PowerMAX OS Guide to Real-Time Services

concurrent

0890479-110
November 2004

To report an error or comment on a specific portion of the manual, photocopy the page in question and mark the correction or comment on the copy. Mail the copy (and any additional comments) to Concurrent Computer Corporation, 2881 Gateway Drive, Pompano Beach, FL 33069. Mark the envelope **“Attention: Publications Department**.” This publication may not be reproduced for any other reason in any form without written permission of the publisher.

Printed in U. S. A.

Revision History:	Level:	Effective With:
Original Release -- November 1995	000	PowerUX Release 2.1.1
Previous Release -- August 2001	090	PowerMAX OS Release 5.1
Previous Release -- August 2003	100	PowerMAX OS Release 6.0/6.1
Current Release -- November 2004	110	PowerMAX OS Release 6.2

Preface

Scope of Manual

This manual provides an overview of the real-time services provided by the frequency-based scheduler and the performance monitor. It explains how to use the associated real-time command processor and library routines. It also describes the data monitoring services.

Structure of Manual

This manual consists of eight chapters, four appendixes, a glossary, and an index. A brief description of the chapters and appendixes is presented as follows:

- Chapter 1 provides an introduction to this guide and an overview of the real-time services.
- Chapter 2 provides an overview of the frequency–based scheduler.
- Chapter 3 explains how to use a real-time clock, an edge-triggered interrupt, and a user-supplied real-time device as the timing source for a frequency-based scheduler.
- Chapter 4 provides an overview of the performance monitor.
- Chapter 5 explains the procedures for using the real–time command processor, **`rtcp`**, and provides reference information for each of its commands.
- Chapter 6 describes the subprograms included in the **`RT_Interface`** package.
- Chapter 7 describes the routines included in the real–time library for C, **`/usr/lib/librt.a`**.
- Chapter 8 describes the subroutines included in the real–time library for FORTRAN, **`/usr/lib/libF77rt.a`**.
- Appendix A contains an example **`rtcp`** script.
- Appendix B provides explanations of the errors that may be reported by **`rtcp`**.
- Appendix C contains an example program that shows how to use the C library interface to the frequency–based scheduler and the performance monitor.

The glossary contains definitions of technical terms that are important to understanding the concepts presented in this book.

The index contains an alphabetical reference to key terms and concepts and numbers of pages where they occur in the text.

Syntax Notation

The following notation is used throughout this manual:

italic	Books, reference cards, and items that the user must specify appear in *italic* type. Special terms may also appear in *italic*s.
`list bold`	User input appears in **`list bold`** type and must be entered exactly as shown. Names of directories, files, commands, options and system manual page references also appear in **`list bold`** type.
`list`	Operating system and program output such as prompts and messages and listings of files and programs appear in `list` type.
`[ ]`	Brackets enclose command options and arguments that are optional. You do not type the brackets if you choose to specify such options or arguments

Referenced Publications

The following publications are referenced in this document:

0890429	*System Administration Manual Volume 1*
0890430	*System Administration Manual Volume 2*
0890466	*PowerMAX OS Real-Time Guide*
0890423	*PowerMAX OS Programming Guide*
0890428	*User’s Guide*
0890497	C/C++ *Reference Manual*
0890516	*MAXAda Reference Manual*
0890240	*hf77 Fortran Reference Manual*

Contents

Chapter 4 Overview of the Performance Monitor

Chapter 5 Using Rtcp

Chapter 6 The Ada Interfaces to RT Services

Chapter 7 The C Library Interface

Chapter 8 The FORTRAN Library Interface

List of Screens

List of Illustrations

List of Tables

1
Introduction

1
Introduction

This chapter describes the focus of this guide and provides an overview of the real-time services provided by the frequency-based scheduler and the performance monitor. It also provides an overview of the data monitoring services.

Focus of Guide

This manual provides an overview of the frequency-based scheduler and performance monitor services and the related utilities and libraries. It describes the peripherals that can be used as timing sources for the frequency-based scheduler. It also describes the interfaces to the data monitoring services.

Frequency-Based Scheduler

A frequency-based scheduler (hereinafter also referred to as FBS) is a task synchronization mechanism that enables you to run processes at frequencies that you specify. Frequencies can be based on high-resolution clocks, an external interrupt source, or completion of a cycle. The frequency-based scheduler provides a mechanism for initiating processes at the specified frequency. The processes are then scheduled via the standard PowerMAX OS priority-based scheduler. You can easily configure a frequency-based scheduler to meet the needs of specific applications. A detailed description of the frequency-based scheduler is provided in Chapter 2.

Convenient access to the major functions associated with frequency-based scheduling is provided by the real-time command processor **rtcp** and the real-time tool NightSim.™ Use of **rtcp** to perform operations associated with the frequency-based scheduler is explained in Chapter 5. Use of NightSim is explained in the *NightSim Quick Reference*.

Access is also provided through libraries of routines that can be called from application programs written in Ada, C and FORTRAN 77. Use of the library interfaces to the frequency-based scheduler is explained in Chapters 6, 7, and 8.

Performance Monitor

The performance monitor is a mechanism that enables you to monitor use of the CPU by processes that are scheduled on a frequency-based scheduler. Values obtained can help you to determine whether you need to redistribute processes among processors for improved load balancing and processing efficiency. A detailed description of the performance monitor and its capabilities is provided in Chapter 4.

Convenient access to the major functions associated with the performance monitor is provided by the real-time command processor **rtcp** and the real-time tool NightSim. Use of **rtcp** to perform operations associated with the performance monitor is explained in Chapter 5. Use of NightSim is explained in the *NightSim Quick Reference*.

Access is also provided through libraries of routines that can be called from application programs written in Ada, C, and FORTRAN 77. Use of the library interfaces to the performance monitor is explained in Chapters 6, 7, and 8.

2
Overview of the FBS

2
Overview of the FBS

This section provides an overview of the frequency–based scheduler. It contains a description of the scheduler and the capabilities it provides, an explanation of configuration parameters, and a description of the user interface.

What Is the Frequency–Based Scheduler?

The frequency–based scheduler is a task synchronization mechanism that enables you to run processes at frequencies that you specify. Frequencies can be based on high-resolution clocks, an external interrupt source, or completion of a cycle. The frequency-based scheduler provides a mechanism for initiating processes at the specified frequency. The processes are then scheduled via the standard PowerMAX OS priority-based scheduler.

The frequency–based scheduler provides you with the ability to:

- Define FBS frequency in terms of the duration of a minor cycle and the number of minor cycles per major frame
- Specify the scheduling parameters with which processes are scheduled
- Control all scheduling features from one processor (that is, schedule and query a process on any processor)
- Detect frame overruns for all FBS–scheduled processes
- Obtain the status of a single FBS–scheduled process, all FBS–scheduled processes on a single processor, or all FBS–scheduled processes on all processors
- Remove one or all FBS–scheduled processes from a scheduler
- Reschedule an FBS–scheduled process
- Start, stop, and resume scheduling on a frequency–based scheduler
- Connect a timing source to and disconnect it from a frequency–based scheduler
- Control use of the real–time clock device as the timing source for a frequency–based scheduler
- Configure up to 100 frequency–based schedulers system–wide in a single processor or multiprocessor environment
- Use both frequency–based scheduling and static priority scheduling simultaneously
- Set the soft overrun limit for an FBS-scheduled process

- Query the soft overrun limit and the total number of soft overruns incurred by an FBS-scheduled process

How Is Scheduler Frequency Defined?

You configure a frequency–based scheduler, in part, by defining the number of minor cycles that compose a major frame. Minor cycles and major frames have associated with them a duration of time that you can define by using a timing source for the scheduler. The timing source can be the end of a minor cycle, a real–time clock, an edge–triggered interrupt, or a user–supplied device. Procedures for using each of the devices as a timing source are explained in detail in Chapter 3.

If you use end–of–cycle scheduling, scheduling is triggered when the last process that is scheduled during the current minor cycle of the current major frame completes its processing.

If you use a real–time clock as the timing source, you define the duration of a minor cycle by specifying the number of clock counts per minor cycle and the number of microseconds per clock count. You determine the duration of a major frame by multiplying the duration of a minor cycle by the number of minor cycles per major frame. If, for example, you configure a scheduler with 100 minor cycles per major frame and you use as the timing source a real–time clock with a clock count of 10,000 and a clock count duration of one microsecond, each minor cycle has a duration of 10,000 microseconds, or 0.01 second, and each frame a duration of one second.

How Are Processes Scheduled?

You schedule processes to run at a certain frequency by specifying the first minor cycle in which the process is to be wakened in each major frame (called the starting base cycle) and the frequency with which it is to be wakened (called the period). If, for example, you schedule “Process 1” with a starting base cycle of zero and a period of two, the process will be wakened once every two minor cycles, starting with the first minor cycle in the frame. If you schedule “Process 2” with a starting base cycle of one and a period of four, that process will be wakened once every four minor cycles, starting with the second minor cycle in the frame. If you then schedule “Process 3” with a starting base cycle of two and a period of two, that process will be wakened once every two minor cycles, starting with the third minor cycle in the frame. On a frequency–based scheduler configured with 100 minor cycles per major frame, these processes will be wakened as illustrated in Table 2-1.

Table 2-1. Process Scheduling

Minor Cycle	Processes Wakened
0	Process 1
1	Process 2
2	Process 1, Process 3
3	

Table 2-1. Process Scheduling (Cont.)

Minor Cycle	Processes Wakened
4	Process 1, Process 3
5	Process 2
...	
97	Process 2
98	Process 1, Process 3
99	

The maximum frequency with which you can schedule a process is once per minor cycle (a period of one); the minimum frequency is once per major frame (in the case of the example, a period of 100).

A process runs until it calls an FBS library routine that causes it to sleep until the frequency–based scheduler wakes it again. The frequency–based scheduler wakes those sleeping processes that are scheduled to be wakened in the current minor cycle of the current major frame and repeats the process for each minor cycle in the current frame. It continues to repeat the entire process on every major frame until the scheduler is disabled. A scheduler configured with 100 minor cycles per major frame, a minor cycle duration of 10,000 microseconds (0.01 second), and a major frame duration of one second wakes processes as illustrated in Table 2-2.

Table 2-2. Scheduler Operation

Major Frame	Time (sec.)	Minor Cycle	Processes Wakened
0	0	0	Process 1
	0.01	1	Process 2
	0.02	2	Process 1, Process 3
	...		
	0.97	97	Process 2
	0.98	98	Process 1, Process 3
	0.99	99	
1	1.00	0	Process 1
	1.01	1	Process 2
	1.02	2	Process 1, Process 3
	...		
	1.97	97	Process 2
	1.98	98	Process 1, Process 3

Table 2-2. Scheduler Operation (Cont.)

Major Frame	Time (sec.)	Minor Cycle	Processes Wakened
	1.99	99	
	...		
n	n.00	0	Process 1
	n.01	1	Process 2
	n.02	2	Process 1, Process 3
	...		
	n.97	97	Process 2
	n.98	98	Process 1, Process 3
	n.99	99	

As illustrated in Table 2-2, when the current major frame is zero and the current minor cycle is zero, the scheduler wakes “Process 1.” After 0.01 second, it wakes “Process 2”; after 0.02 second, it wakes “Process 1” and “Process 3”; and so on. At one second, when the current major frame becomes one, the current minor cycle becomes zero again, and the scheduler wakes “Process 1.” After 0.01 second, it wakes “Process 2”; after 0.02 second, it wakes “Process 1” and “Process 3”; and so on. The scheduler continues repeating this process for as long as it is enabled.

Tolerating Frame Overruns

A process might not always run at the frequency that you have specified. A frame overrun occurs when a scheduled process does not finish its processing before it is scheduled to run again. Frame overruns can be classified into two categories:

- Hard overruns
- Soft overruns.

Hard overruns are catastrophic failures of the scheduled process. Soft overruns are catastrophic failures only if the process reached its limit on the number of soft overruns tolerated. Each scheduled process has a soft overrun limit, defaulting to 0.

Letting a process survive a reasonable number of soft overruns makes the system more flexible and efficient. Some soft overruns result from random, unpredictable, or external events unlikely to recur. Other soft overruns result from only minor frame overruns. Soft overruns give the scheduled process a chance to recover from a frame overrun and return to synchronization.

The OS counts both soft and hard overruns for each scheduled process, but only hard overruns for each scheduler. Other processes can get these counts by querying the scheduled process or scheduler.

When scheduling a process, you can specify that the scheduler be stopped by the kernel when that process running under it causes a hard overrun. If you do not specify that the scheduler should stop when this process causes a hard overrun, then the scheduler will continue to run, regardless of how many hard overruns this process accumulates.

When scheduling a process, you can also specify a consecutive soft overrun limit count that this process will tolerate and have processed as soft overruns by the kernel. Note that the default value for this consecutive soft overrun limit is zero. With the consecutive soft overrun limit set to zero, ALL overruns incurred by this process will be treated as hard overruns (see below).

In addition to the per-process consecutive soft overrun limit value, there is also a system-wide consecutive overrun limit value. This system-wide limit has a default value of 2, and is configured via the **FBSMAXMISSEDFBSWAITS** kernel tunable. The value for this tunable should be set to a value that when exceeded, indicates a serious problem with a scheduled process, a simulation or the system.

When a scheduled process overruns a frame and is not blocked in **`fbswait(2)`** when a frame interrupt occurs, the kernel then makes a decision whether to treat this overrun as a soft or hard overrun. The following steps are taken:

- The process's own consecutive soft overrun counter is incremented.
- If the process's own consecutive soft overrun counter reaches or exceeds either the per-process soft overrun counter or the system-wide consecutive overrun limit value, then this overrun will be treated as a hard overrun. Both the process's and the scheduler's hard overrun counters will be incremented.
- Otherwise, this overrun will be treated as a soft overrun, as long as this process makes a **`fbswait(2)`** call before its own consecutive soft overrun counter does not exceed the per-process soft overrun counter or the system-wide overrun limit value.

When the overrun is treated as a soft overrun, then that process will not block the next time that it calls **`fbswait(2)`**. In this case, it will return immediately from the **`fbswait(2)`** call with a status value of 2.

When the overrun is treated as a hard overrun, then that process will block the next time it calls **`fbswait(2)`**. When the next normally scheduled FBS wakeup for that process occurs, then this process will return out of the **`fbswait(2)`** call, returning a status value of 0. Note that a status of 2 is NOT returned in the hard overrun case.

Installation and Configuration Requirements

Before using the frequency-based scheduler, you must ensure that the **`fbs`** package is installed on your system. This package provides kernel support for the frequency-based scheduler, the performance monitor, and **`rtcp(1)`**. For an explanation of the procedures for installing software packages, refer to the applicable platform *PowerMAX OS Release Notes* and the **`pkgadd(1M)`** system manual page.

You must also ensure that the frequency-based scheduler module (**fbs**) is configured into the kernel. By default, the **fbs** module is not configured. You can use the **config(1M)** utility to (1) determine whether or not the **fbs** module is enabled in your kernel, (2) enable the **fbs** module, and (3) rebuild the kernel. Note that you must be the root user to enable or disable a module and rebuild the kernel. After rebuilding the kernel, you must then reboot your system. For an explanation of the procedures for using **config(1M)**, refer to the "Configuring and Building the Kernel" chapter of *System Administration Volume 2*.

The frequency-based scheduler has associated with it the following system tunable parameters:

FBSMNI
: The maximum number of frequency–based schedulers that can be configured at one time system–wide. The default value for this quantity is 10. You cannot specify a value greater than 100.

FBSUNSCHEDMAX
: The maximum number of unscheduled processes that is permitted on a frequency-based scheduler. The default value for this quantity is -1, which indicates that the maximum number of unscheduled processes permitted per scheduler is equal to the maximum number of scheduled processes permitted on the scheduler. This number is specified when the scheduler is configured.

  A value other than -1 may be specified. This new value will be the maximum number of unscheduled processes permitted for all schedulers.

FBSMAXMISSEDFBSWAITS
: The maximum number of consecutive major frames an FBS-scheduled process is allowed to miss calling **fbswait()** before a catastrophic failure is assumed. The default value is 2.

You can use the **config** utility to (1) determine whether the values of these parameters have been modified for your system, (2) change the value of either of these parameters, and (3) rebuild the kernel. Note that you must be the root user to change the value of a tunable parameter and rebuild the kernel. After rebuilding the kernel, you must then reboot your system.

Coupled FBS Timing Devices

Systems that wish to take advantage of Coupled FBS timing devices must also ensure that the remote device file system kernel module (rdevfs) is configured into the kernel along with the fbs module. By default, the rdevfs module is not configured.

If an integral real-time clock is to be used as a Coupled FBS timing device, then the real-time clock kernel module (rtc) should be configured into the kernel.

If a RCIM device is to be used as a Coupled FBS timing device, then the rcim, rtc and eti kernel modules should also be configured into the kernel. For standalone systems and for the host SBC in a Closely-Coupled cluster, the **config** utility's "RT Features" item under the "Realtime Configure Menu" provides an easy way to enable all of these kernel modules.

Client SBCs in a Closely-Coupled cluster can be configured for supporting either Closely-Coupled or RCIM Coupled timing devices by using a **vmebootconfig** subsystem option. For more information on how to configure client SBCs with Coupled FBS support, see the "Configuring Coupled FBS Support" section in the Cl*osely-Coupled Programming Guide*.

For a discussion about RCIM Coupled and Closely-Coupled FBS timing devices, see the section "Using a Coupled FBS Timing Device" in Chapter 3 of this manual.

User Interface

Use of the frequency-based scheduler is accommodated by the following: (1) **rtcp**, the real–time command processor; (2) NightSim, a real-time tool that provides a graphical user interface to the frequency-based scheduler and the performance monitor; and (3) a set of library routines that can be called from application programs written in Ada, C, or FORTRAN 77. Each interface is introduced in the sections that follow.

Rtcp

The real-time command processor **Rtcp** lets you to do key operations associated with the frequency-based scheduler by entering commands from the keyboard or invoking a script. These operations include configuring a scheduler, scheduling programs, saving a scheduler configuration, setting up a timing source, running a simulation, and querying status.

NightSim

NightSim provides the same capabilities as the real-time command processor **rtcp(1)**. It allows you to perform the entire range of functions associated with the frequency-based scheduler. You can perform the major functions of configuring a scheduler, setting up a timing source, scheduling programs, saving and restoring a scheduler configuration, running a simulation and viewing scheduling data. Complete information on NightSim is provided in the *NightSim Quick Reference*.

Libraries

The **RT_Interface** package and the C **librt** and FORTRAN **libF77rt** libraries contain subroutines that enable you to perform the entire range of functions associated with the scheduler. You can perform the key functions of configuring a scheduler, setting up a timing source, scheduling programs, running a simulation, and retrieving scheduling data. You can also obtain information about the scheduler itself (for example, the minor cycle and major frame counts, the number of frame overruns, the active CPUs). Other functions include those that 1) enable a process that you have scheduled on a frequency–based scheduler to put itself to sleep and 2) enable any process to wake a process that is in the frequency–based scheduler sleep state. All of the subroutines that are contained in the **RT_Interface** package and the C and FORTRAN libraries are described in detail in Chapters 6, 7 and 8.

It is important to note that in PowerMAX OS, some of the scheduling and querying interfaces in the **RT_Interface** package and the C and FORTRAN libraries are obsolete. The reasons are explained as follows. In PowerMAX OS, scheduling priorities are specific to a System V scheduler class or associated POSIX scheduling policy. Some of the scheduling interfaces that are being maintained for compatibility with the CX/UX operating system do not provide the means for specifying a scheduler class or policy. These interfaces are as follows:

Table 2-3. Scheduling Interfaces

Ada	C	FORTRAN
PGM_Schedule	**pgmschedule**	**pgmschedule**
PGM_Reschedule	**pgmreschedule**	**pgmreschedule**

If you schedule or reschedule a process on a frequency-based scheduler by using one of these interfaces, the process is scheduled under the POSIX **SCHED_RR** scheduling policy (fixed-priority class). The priority that you specify must lie within the range of priorities associated with this policy. With these interfaces, you cannot schedule a process under the POSIX **SCHED_OTHER** scheduling policy (time-sharing class). (Scheduler classes, POSIX scheduling policies, and priorities are fully explained in the "Process Scheduling and Management" chapter of the *PowerMAX OS Programming Guide*.)

Some of the querying interfaces that return a process's scheduling priority and are being maintained for compatibility with CX/UX do not provide the means for returning the scheduler class or policy with which the priority is associated. These interfaces are as follows:

Ada:	C:	FORTRAN:
FBS_Query	**fbsquery**	**fbsquery**
PGM_Query	**pgmquery**	**pgmquery**

If you have an existing application that uses the obsolete interfaces listed here, it is recommended that you change your application to use (1) the scheduling interfaces that allow you to specify a scheduling policy and priority and (2) the querying interfaces that return both the policy and priority. The obsolete interfaces and the interfaces with which you should replace them are presented in Table 2-4.

Table 2-4. Obsolete Interfaces

Function	Obsolete Interfaces	Replacement Interfaces
Schedule a process on an FBS	**pgmschedule(3rt)** **pgmschedule(3F77rt)** **PGM_Schedule**	**sched_pgmadd(3rt)** **schedpgmadd(3F77rt)** **Sched_PGM_Add**
Reschedule a process	**pgmreschedule(3rt)** **pgmreschedule(3F77rt)** **PGM_Reschedule**	**sched_pgmresched(3rt)** **schedpgmresched(3F77rt)** **Sched_PGM_Reschedule**
Query processes on an FBS	**fbsquery(3rt)** **fbsquery(3F77rt)** **FBS_Query**	**sched_fbsqry(3rt)** **schedfbsqry(3F77rt)** **Sched_FBS_Query**
Query a process on an FBS	**pgmquery(3rt)** **pgmquery(3F77rt)** **PGM_Query**	**sched_pgmqry(3rt)** **schedpgmqry(3F77rt)** **Sched_PGM_Query**

Procedures for using all of the interfaces presented in Table 2-4 are explained in detail in Chapters 6, 7, and 8.

Privileges

PowerMAX OS supports a privilege mechanism through which processes are allowed to perform sensitive operations or override system restrictions. Some of the operations associated with the frequency-based scheduler require special privileges. These operations include configuring and removing a scheduler, changing the permissions assigned to a scheduler, and scheduling and rescheduling programs. Attaching a timing source to a scheduler also requires special privilege if the Enhanced Security Utilities are installed and running. Specific information related to these privilege requirements is presented in the appropriate sections of this manual. For additional information on privileges, refer to the *PowerMAX OS Programming Guide* and the **intro(2)** system manual page.

Debugging FBS–Scheduled Processes

You can debug processes that have been scheduled on a frequency–based scheduler by using NightView,™ a general-purpose, source-level debugger.

To be able to debug an Ada, a C, or a FORTRAN executable program, you must compile the source program by specifying the **–g** option. For information on use of the Ada compiler, refer to the *MAXAda Reference Manual*. For information on use of the Concurrent C compiler, refer to the *Concurrent C Reference Manual*. For information on use of the FORTRAN 77 compiler, refer to the *hf77 Fortran Reference Manual*.

The NightView commands that you can use to debug FBS-scheduled processes are briefly described as follows:

attach attach to a running process. This command allows you to debug a process that is already running

detach detach from an attached process. This command allows you to release an attached process from the control of the debugger.

To use NightView to debug an FBS–scheduled process, you must supply the process ID (PID). You can easily obtain the PID for an FBS–scheduled process by using the **ps(1)** command. You can obtain the PID for a selected process name by using the C library routine **nametopid(3rt)** or the FORTRAN library routine **nametopid(3F77rt)**. Use of each of these routines is explained in the corresponding system manual pages. If you are using the **RT_Interface** package, you can obtain the PID for the current process by invoking the **POSIX_1003_1.getpid** subprogram.

For NightView to attach to a running process, the debugger's effective user and group ID must match the effective user and group ID of the process controlled by the debugger.

When a debugger attaches to an FBS–scheduled process or when an attached FBS-scheduled process hits a breakpoint, the associated FBS and all processes scheduled under it are stopped.

For additional information on the procedures for using the NightView **attach** and **detach** commands, refer to the *NightView User's Guide*.

Integrity of the Coupled FBS Support

There are a few situations in which the integrity of the Coupled FBS support cannot be guaranteed. While the Coupled FBS support does attempt to recover from various events, such as a single host crashing or inter-host messaging errors, there may be situations when the Coupled FBS support may not be able to recover properly. When such situations occur, the operating system will log error messages to the console and to the system log file. In addition, rtcp and the real-time libraries will report problems. If the system is giving indications that there is a problem with the Coupled FBS support, it may be necessary to reboot all of the hosts that are registered with the same set of Coupled FBS timing devices.

3
Timing Sources for an FBS

3
Timing Sources for an FBS

The frequency–based scheduler provides the capability of using a real–time clock, an edge–triggered interrupt, or a user–supplied device as the timing source for a scheduler. This chapter contains the procedures for using all these types of devices. Use of a real–time clock is explained in “Using a Real–Time Clock”; use of an edge–triggered interrupt is explained in “Using an Edge-Triggered Interrupt”; and use of a user–supplied real–time device is explained in “Using a User–Supplied Real–Time Device.”

Using a Real–Time Clock

In this section, three types of information are provided to ease use of a real–time clock as the timing source for a frequency–based scheduler. An overview of the real–time clock device, **rtc**, is presented in “Understanding the Real–Time Clock Device.” A description of the user–interface to the device is provided in “Understanding the User Interface.” An outline of the general procedures for using the device is presented in “General Procedures for Using a Real–Time Clock.”

Understanding the Real–Time Clock Device

The real–time clock device, **rtc**, is designed to be used for a variety of timing and frequency control functions. It provides a range of clock count values and a set of resolutions that taken together produce many different timing intervals—a feature that makes it particularly appropriate for frequency–based scheduling.

On Series 6000 and PowerMAXION systems, the real–time clock controller is integral to the system. Each CPU board has one real–time clock controller. On Model 6200 and 6800 systems, five real–time clocks are provided on the first CPU board (board 0). Three real–time clocks are provided on each additional CPU board. The HVME real–time clock controller is not supported. On PowerMAXION systems, five real-time clocks are provided on each of the four CPU boards.

On Series 6000 systems, device special files for real–time clocks have names of the form **/dev/rrtc/*mcn***, where ***m*** specifies a controller number that ranges from zero to three and corresponds to the CPU board on which the clock resides; ***c*** stands for clock, and **n** specifies a real–time clock number that ranges from zero to four on the first CPU board and zero to two on additional CPU boards. The names of the device special files on the first board must be as follows:

/dev/rrtc/0c0

/dev/rrtc/0c1

/dev/rrtc/0c2

/dev/rrtc/0c3

/dev/rrtc/0c4

The names of the device special files on the second board must be as follows:

/dev/rrtc/1c0

/dev/rrtc/1c1

/dev/rrtc/1c2

The names of the device special files on the third board must be as follows:

/dev/rrtc/2c0

/dev/rrtc/2c1

/dev/rrtc/2c2

The names of the device special files on the fourth board must be as follows:

/dev/rrtc/3c0

/dev/rrtc/3c1

/dev/rrtc/3c2

On PowerMAXION systems, device special files for real–time clocks have names of the form **/dev/rrtc/***m***c***n*, where *m* specifies a controller number that ranges from zero to three and corresponds to the CPU board on which the clock resides; **c** stands for clock, and *n* specifies a real–time clock number that ranges from zero to four on each CPU board. The names of the device special files on the first board must be as follows:

/dev/rrtc/0c0

/dev/rrtc/0c1

/dev/rrtc/0c2

/dev/rrtc/0c3

/dev/rrtc/0c4

The names of the device special files on the second board must be as follows:

/dev/rrtc/1c0

/dev/rrtc/1c1

/dev/rrtc/1c2

/dev/rrtc/1c3

/dev/rrtc/1c4

The names of the device special files on the third board must be as follows:

/dev/rrtc/2c0

/dev/rrtc/2c1

/dev/rrtc/2c2

/dev/rrtc/2c3

/dev/rrtc/2c4

The names of the device special files on the fourth board must be as follows:

/dev/rrtc/3c0

/dev/rrtc/3c1

/dev/rrtc/3c2

/dev/rrtc/3c3

/dev/rrtc/3c4

On Series 6000 and PowerMAXION systems, each real–time clock is connected to a particular pin on the interrupt terminator board. The hardware controls the interrupt priority associated with each pin. The real-time clock interrupts are handled on the CPU board on which they reside. They cannot be routed to other CPU boards.

On Power Hawk Series 600/700/900 systems, there are integral real-time clocks located on the CPU board and additional clocks available through the Real-Time Clocks and Interrupts Module (RCIM), if installed.

On Power Hawk Series 600 systems, the integral clocks consist of tick timers and Zilog Z8536 timers. On Power Hawk Series 700/900 systems, the integral clocks consist of only tick timers. The number of each available varies based on system type. Only the tick timers can be used as the timing source for a frequency-based scheduler. For complete information on both types of timers refer to the **rtc(7)** system manual page and also to the system header file **/usr/include/sys/rtc.h**.

If a Real-Time Clocks and Interrupts Module (RCIM) is installed, it provides additional real-time clocks. There are four real-time clocks available to each SBC (single-board computer) that has an RCIM. When multiple SBCs are connected via an RCIM chain, up to four RTCs may be designated to be distributed, i.e. its interrupts are sent to all connected systems. A distributed RTC may be located on any SBC within the RCIM chain.

The kernel tunable **RCIM_DISTRIB_RTCS** specifies which RTCs are distributed.

On Power Hawk Series 600/700/900 systems, the device special files for the real-time clocks are as follows (where ***n*** specifies the clock number):

/dev/rrtc/0c*n*	rtc (character) special files - tick timer
/dev/rrtc/1c*n*	rtc (character) special files - Z8536 (Power Hawk Series 600 systems only)
/dev/rrtc/2c*n*	rtc (character) special files - RCIM

NOTE

> To use a real–time clock on a PowerMAX OS system on which the Enhanced Security Utilities are installed, device special files must be created in the **/dev/rrtc** directory. Refer to the "Trusted Facility Management" chapter of *System Administration Volume 1* for an explanation of the procedures for using device files when the Enhanced Security Utilities are installed.

A real–time clock operates in one of two modes: default mode or direct mode. If the clock is in default mode, you can control the following:

- Whether the clock counts up or down
- What the value of the clock count is
- What the resolution per clock count is
- Whether the clock automatically starts counting again when the clock count reaches its terminal count (zero or 65,535)

If the clock is in direct mode, you can directly program the hardware registers. Doing so requires information on the system timing chip and its registers. The information needed can be obtained by special request. Directly programming the hardware registers also requires information that is provided in the system manual page **rtc(7)**. Note that on Power Hawk systems, direct mode is not supported on the tick timers.

You can use a real–time clock for triggering events in the high–resolution callout queue. By default, a real-time clock is configured into the system for this purpose it is **/dev/rrtc/0c0**). When you configure a real-time clock for use with the high-resolution callout queue, you cannot use it for any other purpose. Additional information on the high-resolution callout queue is presented in the *PowerMAX OS Real-Time Guide*. Note that using a real-time clock with the high-resolution callout queue does not affect the resolution of other clocks on the same controller.

Understanding the User Interface

Use of a real–time clock with a frequency–based scheduler is accommodated by the real–time utility **rtcp**; by the FBS FORTRAN and C library routines; and by the **RT_Interface** package. It is also accommodated by NightSim, the real-time tool that provides a graphical user interface to the frequency based scheduler and performance monitor service (use of NightSim is fully explained in the *NightSim Quick Reference*).

The **rtcp** commands needed to attach a timing source to and detach it from a scheduler and to set, start, and stop a real–time clock are **ats, dts, stc, rc,** and **sc**. These commands are explained in Chapter 5. These commands are used in conjunction with the commands to start and stop a simulation, **start** and **stop**, which are also explained in Chapter 5.

The FORTRAN and C library routines that relate to use of a real–time clock for frequency–based scheduling are **fbsattach, fbssetrtc, fbsrunrtc, fbsintrpt**, and **fbsdetach**. The corresponding routines contained in the Ada package are

FBS_Attach, FBS_Setrtc, FBS_Runrtc, FBS_Intrpt, and **FBS_Detach**. These routines are explained in Chapter 6, Chapter 7, and Chapter 8.

A set of system calls can be used directly to control a real–time clock and to use it for frequency–based scheduling and other purposes. The calls are explained in detail in the system manual page **rtc(7)**.

It is recommended that you use **`rtcp`** or the routines contained in the FORTRAN library, the C library, or the **RT_Interface** package as your interface to the real–time clock. The **`rtcp`** command is the easiest to use because it isolates you from most of the initialization tasks.

Watch-Dog Timer Function

The fifth RTC on the first board can be used as watch-dog timer or as an interrupting real time clock. When used as an interrupting clock the output of the RTC that indicates a time out continues to be connected to the interrupt control logic. However, when the RTC is being used as a watch-dog timer, its interrupt is disabled via software issuing a disarm interrupt command to the RTC’s interrupt level. The RTC’s time-out output is also connected to the logic Processor Control and Status Register (PCSR).

The RTC used in the watch-dog timer function is programmed by the application using the facilities provided under PowerMAX OS. For more information on software control capability of the RTC, refer to the manual pages section -rtc.

It is recommended that the RTC watch-dog timer be used in the default mode and programmed to have a clock resolution of 1 millisecond. This time gives a time out range of from 1 millisecond to 65.535 seconds. In the event of a time-out, the hardware generates the SRESET signal to the PPC604 processor. This signal causes the processor to save the machine state in its Save and Restore Registers (SRR) and start execution of a soft reset exception. This execution’s execution starts at physical location 0x00000100. The exception handler then tests the MODULE_NO_GO register flag to find out if the cause of the soft reset is the watch-dog timer time-out. If it is, processing of the soft reset continues by resetting of the MODULE_NO_GO bit followed by a reset of the SRESET register bit in the PCSR. Control is now passed to a user defined exception handler.

Using the watch-dog function, the application can monitor the health of its processes. To accomplish this the application must program for the watch-dog interrupt in the following manner:

1. The fifth real time clock's interrupt must be disabled on the processor's interrupt controller. The application does this by mapping the interrupt controller's enable register using the shared memory mechanism.The physical addresses for the interrupt enable registers on the PowerMAXION are:

```
0x96200020 local processor
0x9D000020 processor 0
0x9D100020 processor 1
0x9D200020 processor 2
0x9D300020 processor 3
```

The fifth real time clock interrupt is disabled by resetting bit 17 in the 32-bit enable register.

The application must take care to not change other bits in the interrupt controller's enable register. This can be achieved by reading the enable register, masking out only bit 17, and re-writing the contents back to the enable register.

The application has the responsibility of re-enabling this interrupt once use of the watch-dog timer is complete. This is achieved by setting bit 17 in the enable register. Failure to do so will preclude the fifth real time clock on processor board one from being used as a timer.

2. The interrupt signal from the fifth real-time clock must be routed to the PPC604 processor. The application does this by mapping the processor's 16 bit control and status register (PCSR) using the shared memory mechanism. The physical addresses for the PCSR's are as follows:

```
0xB2000000 processor 0
0xB2000008 processor 1
0xB6000000 processor 2
0xB6000008 processor 3
```

Routing of the fifth real-time clock interrupt is achieved by setting bit 11 in the PCSR for the respective processor board. The application must take care to not change other bits in the PCSR. This can be achieved by reading the register, setting bit 11, and re-writing the contents back to the register.

The application has the responsibility of restoring bit 11 of the PCSR to 0 once use of the watch-dog timer function is complete. Failure to do so will preclude the fifth real time clock on processor board one from being used as a timer.

3. The application must connect and enable the user level interrupt routine. This is achieved using the **`iconnect(3C)`** and **`ienable(3C)`** routines. The application must also lock all memory resources used by the user level interrupt routine. These resources include shared memory regions, library text and data, process text and data. See the *PowerMAX OS Real-Time Guide* (publication number 0890466) for a description of user level interrupts.

The fifth real time clock must be programmed by the application with the correct count and frequency. PowerMAX OS supplies a user interface to the real-time clocks.

General Procedures for Using a Real–Time Clock

Whether you elect to use **`rtcp`** or the routines contained in the FORTRAN library, the C library, or the Ada package as your interface to the real–time clock, the general procedures for using a real–time clock for frequency–based scheduling are the same. The following steps are required:

STEP 1: Attach a real–time clock to a frequency–based scheduler

STEP 2: Establish the duration of a minor cycle by specifying the clock count value and the resolution per clock count

STEP 3: Start the real–time clock counting

STEP 4: Start the simulation

STEP 5: Stop the simulation

STEP 6: Stop the real–time clock counting

STEP 7: Detach the real–time clock

The **rtcp** commands and the FORTRAN, Ada, and C routines that correspond to each step are presented in Table 3-1.

Table 3-1. Tasks, Commands, and Routines Related to Steps

Step	Command	FORTRAN Routine	Ada Routine	C Routine
1	ats	fbsattach	FBS_Attach	fbsattach
2	stc	fbssetrtc	FBS_Setrtc	fbssetrtc
3	rc	fbsrunrtc	FBS_Runrtc	fbsrunrtc
4	start	fbsintrpt	FBS_Intrpt	fbsintrpt
5	stop	fbsintrpt	FBS_Intrpt	fbsintrpt
6	sc	fbsrunrtc	FBS_Runrtc	fbsrunrtc
7	dts	fbsdetach	FBS_Detach	fbsdetach

Refer to Chapter 5 for descriptions of the **rtcp** commands and to Chapter 6, Chapter 7, and Chapter 8 for explanations of the routines included in the Ada package and the C and FORTRAN libraries.

NOTE

To use a real–time clock as the timing source for a frequency–based scheduler on a PowerMAX OS system on which the Enhanced Security Utilities are installed, you must have enough privilege to open the device. Refer to the “Trusted Facility Management” chapter of *System Administration Volume 1* for an explanation of the procedures for using devices when the Enhanced Security Utilities are installed.

Using an Edge-Triggered Interrupt

This section contains the information needed to use an edge–triggered interrupt as the timing source for a frequency–based scheduler. An overview of the edge–triggered interrupt, **eti**, is presented in "Using an Edge-Triggered Interrupt." A description of the user–interface to the device is provided in "Understanding the User Interface."

Understanding the Edge–Triggered Interrupt

The edge–triggered interrupt device, **`eti`**, provides a means for the computer system to detect an external interrupt coming into the system from any user device that generates a signal pulse. It can be used as the timing source for a frequency–based scheduler.

On 6000 and PowerMAXION systems, edge–triggered interrupts are integral to the system. Four edge–triggered interrupts are provided for each CPU board. One to four CPU boards may be configured; as a result, the number of edge–triggered interrupts per system ranges from four to 16.

All the edge-triggered interrupts, by default, are automatically configured despite the number of CPU boards installed on a system. However, at least one CPU board must be installed in the system. If you do not wish the edge-triggered interrupts to be configured in your system, you can edit the **`/etc/conf/sdevice.d/eti`** file and change the value in the **`conf`** field to **`N`**.

On Series 6000 PowerMAXION systems, device special files for the integral edge–triggered interrupts have names of the form **`/dev/reti/eti`***n*, where *n* specifies an edge–triggered interrupt number ranging from zero to 15. The numbers **0**–**3** are the edge–triggered interrupts on CPU board 0; **4**–**7** are the edge–triggered interrupts on CPU board 1; **8**–**11** are the edge–triggered interrupts on CPU board 2; and **12**–**15** are the edge–triggered interrupts on CPU board 3. If a CPU board in a specified slot is marked down or is not present, the numbering scheme is not affected. If a system contains a CPU board in slot 0 and a CPU board in slot 3, for example, the edge–triggered interrupts on the first board are numbered **0**–**3**, and the edge–triggered interrupts on the second board are numbered **12**–**15**

On Series 6000 PowerMAXION systems, each edge–triggered interrupt is connected to a particular pin on the terminator board. Edge-triggered interrupts are handled on the CPU board on which they reside. They cannot be routed to other CPU boards.

For detailed information on the edge–triggered interrupt hardware and the conditions that are required for using it, refer to the system manual page **`eti(7)`**, the *HN6200 Architecture Manual*, the *HN6800 Architecture Manua*l, or the *PowerMAXION Architecture Manual*.

On Power Hawk Series 600/700/900 systems, edge-triggered interrupts are provided by the Real-Time Clocks and Interrupts Module (RCIM), if installed. There are four edge-triggered interrupts available to each SBC (single-board computer) that has an RCIM. When multiple SBCs are connected via an RCIM chain, up to four ETIs may be designated to be distributed, i.e. its interrupts are sent to all connected systems. A distributed ETI may be located on any SBC within the RCIM chain.

The kernel tunable **`RCIM_DISTRIB_ETIS`** specifies which ETIs are distributed.

On Power Hawk Series 600/700/900 systems, ETI device special files are only available if an RCIM module is installed. They have the following format:

/dev/reti/eti0*n*	eti (character) special files

For more information, refer to the system manual page **eti(7)**.

Understanding the User Interface

Use of an edge–triggered interrupt as the timing source for a frequency–based scheduler is accommodated by the real–time services utility **rtcp**; by the FBS FORTRAN and C library routines; and by the **RT_Interface** package. It is also accommodated by NightSim, the real-time tool that provides a graphical user interface to the frequency based scheduler and performance monitor service (use of NightSim is fully explained in the *NightSim Quick Reference*).

To use an edge–triggered interrupt as the timing source for a scheduler, you must attach it to the desired scheduler and ensure that it is already generating interrupts when you start the simulation. The **rtcp** commands needed to attach and detach a timing source are **ats** and **dts**. Use of these commands is explained in Chapter 5. The corresponding FORTRAN and C library routines are **fbsattach** and **fbsdetach**. The FORTRAN routines are explained in Chapter 8, and the C routines are explained in Chapter 7. The corresponding routines contained in the Ada package are **FBS_Attach** and **FBS_Detach**. These routines are explained in Chapter 6

The edge–triggered interrupt can be directly controlled by using the following standard PowerMAX OS system calls: **open(2)**, **close(2)**, and **ioctl(2)**.

NOTE

This device does not support the **read(2)** and **write(2)** system calls.

A set of **ioctl** commands enables you to perform a variety of operations that are specific to the device. These commands are summarized as follows:

ETI_ARM	arm the edge–triggered interrupt
ETI_DISARM	disarm the edge–triggered interrupt
ETI_ENABLE	enable the edge–triggered interrupt
ETI_DISABLE	disable the edge–triggered interrupt
ETI_INFO	obtain information about the specified edge-triggered interrupt.
ETI_REQUEST	generate a software–requested interrupt along the edge–triggered interrupt. Note that the edge–trig-

	gered interrupt must previously have been armed and enabled. Not available with distributed slave device.
ETI_ATTACH_SIGNAL	attach the specified signal number to the edge–triggered interrupt. The signal is generated on every interrupt.
ETI_VECTOR	place the edge–triggered interrupt vector number in the specified location. Note that any device that is being attached to a frequency–based scheduler must support this command.

Detailed descriptions of these commands and the specifications required for using them are presented in the system manual page **`eti(7)`**.

Using a User–Supplied Real–Time Device

You may wish to use your own device as the timing source for a frequency–based scheduler under one of the following conditions:

- You desire a minor cycle duration that is greater than 655 seconds.
- You wish scheduling to be triggered by asynchronous events.

  (Note that in this case, you must provide the hardware and the software to service the event.)

To use your own device, you must ensure the following:

- That your device driver supports the IOCTLVECNUM **`ioctl`** call (IOCTLVECNUM is defined in **`<sys/ioctl.h>`**)
- That your device generates a series of interrupts

NOTE

When a user–supplied timing source is attached to a frequency–based scheduler, its interrupt service routine will no longer be executed. You must specify a timing source that does <u>not</u> require re-enabling of interrupts within the interrupt service routine; otherwise, the device will not be able to generate a series of interrupts as required by the scheduler. When the timing source is detached from the scheduler, its interrupt service routine will handle subsequent interrupts.

Use of the **`ioctl`** call is explained in “Specifying the Ioctl Call.”

Specifying the Ioctl Call

The IOCTLVECNUM **ioctl** call is made to obtain the interrupt vector number of the device. It requires the following specifications:

```
#include <sys/types.h>
#include <sys/ioctl.h>

ioctl(fildes, IOCTLVECNUM, arg)
int fildes;
int *arg;
```

Arguments are defined as follows:

fildes	the file descriptor for the device
IOCTLVECNUM	the command to place the interrupt vector number of the device in the location pointed to by *arg*
arg	a pointer to the location to which the interrupt vector number of the device will be returned.

The IOCTLVECNUM **ioctl** call returns the interrupt vector number of the device if the operation is successful; it returns **–1** if this **ioctl** call is not supported.

Note that when a user–supplied timing source is detached from a frequency–based scheduler, no corresponding **ioctl** call is made. Resetting the timing source to its default state is the responsibility of the user.

Using a Coupled FBS Timing Device

This section contains information about using Coupled FBS timing devices. There are a variety of system configurations where Coupled FBS timing devices may be used. A Coupled timing device may be used to couple together FBS schedulers that are located on more than one computer system (host). All schedulers that are attached to the same Coupled FBS timing device will start, stop and resume their executions together on the same frame and cycle, using the Coupled FBS timing device as the interrupt source.

Device Registration

Using a Coupled FBS timing device is not that much different from using any other non-Coupled timing device. However, to use a device as a Coupled FBS timing device, it must first be registered on the host where the device actually resides. After registration, you may proceed as you would with any other non-Coupled FBS timing device.

To register a device using **rtcp**, use the **rd** (register device) command. Once this has been done, the device is then available for use on all registered hosts and it can then be used with other **rtcp** commands.

To register a device using the C or FORTRAN real-time libraries, use the **fbs_register_rdev** function. Once this has been done, the device is available for use on all registered hosts, and it can then be used with other C or FORTRAN real-time library functions.

One FBS scheduler per-host may be attached to a Coupled FBS timing device. Note that the appropriate **`/dev/rdev`** entry should be used as the attach path name, even on the host where the timing device actually resides (where the device interrupts originate).

Understanding Coupled FBS Timing Devices

There are two basic components to Coupled timing devices:

- the propagation of the timing device interrupt to all attached schedulers
- the sending and receiving of communication messages between attached schedulers and the Coupled FBS timing device support code

  One example of a communication message would be for requesting that the timing device start sending interrupts to all the attached schedulers. This start type of message would be sent from an attached scheduler to the host where the timing device interrupt originates. Another example would be the communication messages that are sent between hosts during the timing device registration and un-registration operations.

The mechanism used for propagating the timing device interrupt and the mechanism used for sending and receiving communication messages depends upon the type of Coupled FBS timing device that has been registered.

There are two types of Coupled FBS timing devices:

RCIM Coupled timing device

For this type of timing device, the device must be a RCIM real-time clock or edge-triggered interrupt, and the device must be configured to distribute its interrupts through the RCIM cable.

Additionally, all remote hosts where this device is registered must be configured to receive this device’s distributed interrupt through the RCIM cable.

A RCIM Coupled timing device propagates the timing device interrupt through the RCIM cable, with this interrupt signal directly interrupting each attached scheduler’s host system.

The associated communication messages that are sent between hosts are always sent with networking messages.

In order to use a RCIM Coupled timing device, all of the hosts where this device is registered must be accessible via standard TCP/IP networking, and each host must be attached to the same RCIM cable. Any combination of standalone, Closely-coupled or Loosely-coupled hosts/SBCs may be used, as long as these two requirements are met.

Closely-Coupled timing device

Closely-Coupled timing devices may only be used by hosts (SBCs) within the same Closely-coupled cluster.

For these timing devices, the SBC messaging support is always used to send communication messages between hosts.

Closely-Coupled FBS timing devices may be either RCIM or non-RCIM devices. The Closely-Coupled timing device interrupts that are propagated to all the attach schedulers are sent in one of two ways: by SBC messages or by a distributed RCIM interrupt through the RCIM cable.

The timing device interrupt will be propagated with SBC messages for all non-RCIM devices, such as integral real-time clocks. SBC messages will also be used to propagate the timing device interrupts for RCIM devices that are not configured to distribute their interrupts through the RCIM cable.

However, RCIM devices that are configured to have their interrupts distributed through the RCIM cable will have their timing device interrupts sent directly through the RCIM cable to the attached schedulers in the cluster. As is the case for RCIM Coupled FBS timing devices, when the RCIM device is configured as a distributed interrupt, all remote SBCs where the device is registered must have an RCIM that is attached to the same RCIM cable, and each remote SBC must have its RCIM configured to receive this timing device's interrupt through the cable.

Note that it is usually more efficient to use a distributed interrupting RCIM device as the timing source, since this avoids the system overhead of sending an SBC message for each Coupled FBS timing device interrupt.

The Remote Device File System

When a timing device is registered, a corresponding device file entry will be added to the remote device file system (**rdevfs**) on every host where the device has been registered.

The **rdevfs** file system provides local access to timing devices that are registered as Coupled FBS timing devices and that may actually reside on a remote host. Coupled FBS timing device files are located in **/dev/rdev** subdirectories, where the subdirectory name indicates the name of the host where the device actually resides.

For devices that are registered with the **rtcp rd** command, or the **fbs_register_rdev** function, the subdirectory name will be the name of the local host that was specified in the list of registration hostnames. For example, if the list of registration hostnames was **rudi**, **cosmo** and **endor** and the device actually resides on **endor**, then the **/dev/rdev/<*hostname*>** subdirectory for that device will be '**/dev/rdev/endor**'.

A given subdirectory **/dev/rdev/<*hostname*>** contains device file entries, each of which represents a device that was registered as a Coupled FBS timing device. A device file entry is of the form '**device<*n*>**', where <***n***> is the numeric indication of the device's ordinal number, for example **1** for the device registered, **2** for the second device registered, and so on.

Thus, the Coupled FBS timing device file name is of the form '**/dev/rdev/<*hostname*>/device<*n*>**'. This is the file name that should be used when attaching a timing source to a FBS scheduler.

When a Coupled FBS timing device is unregistered, the corresponding **rdevfs** device file entry will be removed on all hosts where the device was previously registered.

NOTE

When timing devices are registered with either the obsolete **fbs_register_cluster_device** function, or with the obsolete **rtcp reg** command, the '**/dev/rdev**' subdirectory names are of the form '**sbc<*x*>**', where **<*x*>** is the SBC board ID of the SBC where the timing device actually resides. Thus, the timing device file entries in this case will be of the form '**/dev/rdev/sbc<*x*>/device<*n*>**'.

Understanding the User Interface

Use and management of Coupled FBS timing devices is supported by rtcp and the FBS C and FORTRAN real-time libraries, librt and libF77rt, respectively.

The **rtcp** commands needed to register and unregister timing devices are **rd** and **urd**, respectively. The **ats rtcp** command is used to attach a coupled FBS timing device to a FBS scheduler. The **gtc**, **rc**, **sc**, and **stc** commands may be used to get, start, stop and set values for a Coupled FBS timing device, if the actual device is a real-time clock. Other devices that are used as Coupled FBS timing devices must be directly manipulated (initialized, etc.) with ioctl(2) calls on the host where that device actually resides, using the real device file name (not the **/dev/rdev** name) to open(2) the device

When a scheduler is attached to a Coupled FBS timing device, the **rtcp vs** command will output additional information about the name of the host where the device actually resides, the real name of the timing device on that host, a list of hostnames where the device is registered, and a list of hostnames of the hosts that currently have a FBS scheduler attached to that device. Additionally, if the timing device is a Closely-Coupled timing device, the **vs rtcp** command will also output the SBC board ID of the SBC where the device actually resides, and a SBC board ID mask of the SBCs that currently have a scheduler attached to this timing device. The **rtcp vr** command or the **fbsinfo_rdev** function call may be used to obtain information about a **/dev/rdev/<*hostname*>/device<*n*>** timing device without requiring that a scheduler be currently attached to the device. C and FORTRAN real-time library functions that manage timing devices are **fbs_register_rdev**, **fbs_unregister_rdev** and **fbsinfo_rdev**. The **fbsattach** function can be used to attach a Coupled FBS timing device to an FBS scheduler, using the **/dev/rdev/<*hostname*>/device<*n*>** pathname. The **fbsgetrtc**, **fbsrunrtc** and **fbssetrtc** functions may be used to get, start, stop or set values for a Coupled FBS timing device if the actual device is a real-time clock.

It is strongly recommended that you use the **rtcp** commands or the C or FORTRAN real-time library functions as your interface to the Coupled FBS timing devices.

NOTE

The **fbs_register_cluster_device** and **fbs_unregister_cluster_device** functions, and also the **`rtcp reg`** and **`unreg`** commands are obsolete. They are currently being provided only for backward compatibility with previous PowerMAX OS releases. Users are highly encouraged to make use of the newer **fbs_register_rdev**, **fbs_unregister_rdev** and **fbsinfo_rdev** functions, as well as the **`rtcp rd`**, **`urd`** and **`vr`** commands.

It should also be mentioned that the **fbsinfo_rdev** function and the **`rtcp vr`** command will not work on Coupled FBS timing devices that were registered with the **fbs_register_cluster_device** function call or the **`rtcp reg`** command.

Scheduler Synchronization

All frequency based schedulers that are attached to the same Coupled FBS timing device are stopped, started and resumed together, thus maintaining the same number of frame/cycle interrupts delivered to each scheduler residing on different hosts.

However, there are two exceptions when all schedulers will not necessarily stop on the same frame/cycle count:

- when a LWP within a frequency-based scheduler hits a breakpoint that has been set from within a debugger utility, such as **`NightView(1)`**,
- when an overrun occurs and the local scheduler must be stopped.

In these cases the local scheduler is stopped immediately, ignoring any future frame/cycle interrupt notifications. Until the kernel is able to send a stop device communication message to the host where the Coupled FBS timing device actually resides and the device interrupts are then set to be ignored, any additional interrupts generated by the Coupled FBS timing device up to that point will still be propagated to all attached schedulers, thus making the already stopped scheduler potentially out of sync with the other attached schedulers, in terms of frame and cycle count values.

Using RCIM Edge-Triggered Interrupts and Real-Time Clocks

Power Hawk Series 600/700/900 systems that have a Real-Time Clocks and Interrupts Module (RCIM) installed may make use of the real-time clocks and edge-triggered interrupts located on this board as timing devices for frequency-based schedulers.

For more information on these topics, see “Understanding the Real–Time Clock Device“on page 3-1 and “Understanding the Edge–Triggered Interrupt“on page 3-8.

As a Local Timing Device

Each real-time clock or edge-triggered interrupt may be used locally as a standard FBS timing device that is used only by a frequency based scheduler located on that same SBC board.

In this case, the device path names that should be used by frequency-based scheduler applications for real-time clocks are:

```
/dev/rrtc/2c0
/dev/rrtc/2c1
/dev/rrtc/2c2
/dev/rrtc/2c3
```

And the path names that should be used for edge-triggered interrupts are:

```
/dev/reti/eti00
/dev/reti/eti01
/dev/reti/eti02
/dev/reti/eti03
```

When using these devices as standard timing devices, the information contained in the section “Using a Real–Time Clock” and “Using an Edge-Triggered Interrupt” will also apply to these devices.

As a Coupled FBS Timing Device

As mentioned in the section, "Understanding Coupled FBS Timing Devices", RCIM real-time clocks and edge-triggered interrupts may be used as Coupled FBS timing devices.

When used as a RCIM Coupled timing device, the RCIM device must be configured to distribute its interrupts through the RCIM cable on the host where the device resides. All other remote hosts where a distributed RCIM device is registered must be configured to receive this device’s interrupts through the RCIM cable. Any set of hosts that are connected to the same RCIM cable and that can also communicate among each other with standard TCP/IP networking can make use of the same RCIM Coupled timing devices.

When a RCIM device is used as a Closely-Coupled timing device, then all hosts that are registered to make use of the Closely-Coupled timing device must reside within the same Closely-Coupled cluster system. This is because the inter-host communication messages that are used to support the Coupled FBS timing device are passed by using the SBC VME messaging interface. The RCIM device does not necessarily have to be configured to distribute its interrupts through the RCIM cable. In this case, the device interrupts are propagated by using the SBC messaging support to pass messages to the other SBCs as notification of a device interrupt. However, if the RCIM device is configured to distribute its interrupts through the RCIM cable, then SBC messages are not needed for passing the device interrupt notifications. In this case, the interrupts are sent directly to each registered host through the RCIM cable, and for this reason, all other remote hosts where the device is registered must be configured to receive this device’s interrupts through the RCIM cable. Since the distributed interrupt capability provides a much faster and lower overhead method than using SBC messaging for interrupt propagation, it is recommended that the distributed interrupt feature be used when ever possible.

Configurations with Limited RCIM Hardware

In some situations, it might be desirable to connect multiple Closely-Coupled clusters together with RCIM boards and a RCIM cable, where within each cluster there might be just one SBC with a connected RCIM board and cable. In this type of configuration, the other SBCs within each cluster that are not configured with a RCIM board would not usually be able to participate in a Coupled FBS simulation with the other RCIM-configured SBCs.

In an effort to provide some amount of limited support to all SBCs in this type of mixed RCIM and non-RCIM SBC configurations, a device registration of a **/dev/distrib_intr[*n*]** device file will be provided solely for this purpose.

The **/dev/distrib_intr[*n*]** device files are used for receiving incoming interrupts from remote or local RCIM devices that are configured to distribute their interrupts through the RCIM cable. These device files should not usually be registered as Coupled timing devices; instead, the actual RCIM device where the interrupts originate from should usually be registered as a RCIM coupled FBS timing device.

However, for this type of situation, by allowing the registration of the **/dev/distrib_intr[*n*]** device file that receives the incoming interrupts through the RCIM cable, a Closely-Coupled timing device will be created that is available on all SBCs within that one cluster. In this case, the kernel will treat this registered device as a non-distributed, edge-triggered interrupt (ETI), closely-coupled timing device. All incoming interrupts received by this device will be propagated to all participating SBCs/hosts within that cluster through the use of SBC messages.

Note that FBS schedulers on the SBCs within the cluster should still attach themselves to the corresponding **/dev/rdev/[hostname]/device[*n*]** device file, and <u>not</u> to the **/dev/distrib_intr[*n*]** device file.

In this type of configuration, the start and stop scheduler operations will have the affect of enabling or disabling the propagation of interrupts received by this registered Closely-Coupled timing device within that local cluster.

However, note that the actual timing device that is distributing its interrupts through the RCIM cable is **not** controlled by the start/stop scheduler operations issued locally within the each cluster. Therefore, co-ordination between clusters for the processing of these incoming RCIM distributed interrupts is entirely up to the user.

Typically, the real RCIM timing device would also be registered as a RCIM Coupled timing device, and any scheduler attached to this device could be used to start and stop the real device interrupts.

For example, in order to start all attached schedulers on all clusters together in a synchronized fashion, within each cluster one would first issue a start scheduler operation on a scheduler that is attached to the registered **/dev/distrib_intr[*n*]** Closely-Coupled timing device. Then the real RCIM Coupled timing device that is distributing its interrupts through the RCIM cable to each of the clusters could be started by a scheduler that is attached to the real RCIM Coupled timing device.

As a Distributed Interrupt Device Without Coupled FBS Support

The Coupled FBS support provides customers with an easy way to setup and use a distributed RCIM interrupting device across multiple hosts. However, keeping all attached schedulers synchronized across the registered hosts does cause additional system overhead, particularly at scheduler start, resume and stop event points.

For those applications that require the fastest possible scheduler start, stop and resume operations across all attached schedulers, it is possible to directly make use of a RCIM distributed interrupt device without using the Coupled FBS support. While this approach causes a loss of functionality and also places all of the cross-host synchronization responsibilities on the customer, it does reduce the amount of system overhead that is associated with running in a Coupled FBS environment.

When directly using the distributed interrupt RCIM device, the user does not register the device with **fbs_register_rdev**, but instead the application code on the local host will open, control and attach its scheduler to the **/dev/rrtc/2c<*n*>** or **/dev/reti/eti0<*n*>** file, and the remote host application's FBS scheduler must attach itself to the corresponding **/dev/distrib_intr<*n*>** device file. (See the **rcim(7)** system manual page for details on **distrib_intr** device files.)

When the Coupled FBS timing device support is not used, then all start, stop, resume, overrun and breakpoint stops must be coordinated entirely by the user. For example, since a stop scheduler fbsintrpt call affects only the local scheduler, the other schedulers attached to this same distributed interrupt will not also stop as a result of this one scheduler's stop operation.

The recommended method for keeping all attached schedulers synchronized on the same frame/cycle counts is to directly start and stop the actual distributed RCIM timing device on the host where the device resides, and to not make use of the usual per-scheduler fbsintrpt start, stop and resume functions (or the **rtcp start** and **stop** commands).

The following is an example method for starting together all of the schedulers that attach themselves to the same distributed RCIM interrupt source:

1. On the host where the RCIM timing device actually resides, open(2) the appropriate **/dev/rrtc/2c<*n*>** or **/dev/reti/eti0<*n*>** device file. Then ensure that this device is not currently generating interrupts by making the appropriate ioctl(2) calls.

2. Attach all schedulers to the distributed RCIM interrupt, using the appropriate **/dev/distrib_intr<*n*>** device file. An ETI_ARM and ETI_ENABLE ioctl(2) call must be made on this device file in order for the distributed interrupts to be received.

3. Issue a fbsintrpt start function call (or rtcp start command) for all the attached schedulers. This call give each scheduler the ability to receive and process each incoming Coupled timing device interrupt.

4. On the host where the RCIM timing device actually resides, issue the appropriate ioctl(2) calls to start the distributed RCIM device running (interrupting).

The FBS Daemon

When Coupled FBS support is configured into the kernel, the system will automatically startup a FBS user-level daemon when the system goes into **`run level 3`**.

This FBS user-level daemon is used for the sending and receiving of communication messages between registered hosts in a Coupled FBS environment. Many of these received message requests result in the execution of various system service calls on the local host on behalf of a remote host.

While this FBS daemon is automatically invoked by the Coupled FBS support within the **`/etc/init.d/fbs`** script, it should be mentioned that there are several FBS daemon options that the system administrator may wish to alter from their default values, depending upon system and application requirements.

Specifically, two types of options that the system administrator may wish to modify are the FBS daemon's scheduling class and priority, and the enabling of the Coupled FBS timing device cleanup processing (discussed in the section "Existing Device Registration Cleanup" on the next page). For more information on the FBS daemon options, see the system manual page **`fbsd(1M)`**.

Coupled FBS Timing Device Error Recovery

The following subsections discuss several aspects regarding recovering from various types of errors that can occur when using Coupled FBS timing devices. Since Coupled FBS timing devices are used across multiple hosts, certain events, such as inter-host message communication errors or having a host crash, can cause problems with the successful functioning of these types of timing devices.

Failed Registrations

When a **fbs_register_rdev** or **`rtcp rd`** device registration fails in the middle of the registration process, some hosts may have successfully registered that device locally on their system, while other hosts may have failed to complete their own local registrations of this Coupled FBS timing device due to reasons such as inter-host communication errors or improper device configuration a of distributed RCIM device. In this case, the Coupled FBS support will automatically attempt to completely back out any device registration information from any host that may have already completed its local device registration processing.

Therefore, upon return from a failed device registration function call or **`rtcp`** registration command, the caller may assume that the local host and all remote hosts have been purged of any information regarding the device's registration.

Existing Device Registration Cleanup

When there are one or more Coupled FBS timing devices that are registered on a local host, the set of remotely registered hosts will contain kernel information about that Coupled FBS timing device, as well as inter-host communication connections and possibly a FBS scheduler that is attached to that timing device.

If the host where the timing device actually resides crashes or is taken down with active local device registrations, then all the registered remote hosts will no longer be able to properly use that Coupled FBS timing device. However, at that point, the **`rdevfs(4)`** device file entry for that remote timing device will still exist on each remote host, and their local scheduler (if any) will still remain attached to the remote Coupled FBS timing device.

As a means to aid in removing this defunct remote Coupled FBS timing device from all the hosts where this device was registered, the Coupled FBS support provides some automatic cleanup of timing devices through the FBS daemon.

When the FBS daemon's **`-c`** option is specified (see the **`fbsd(1M)`** system manual page), then the FBS daemon will attempt at invocation time, to cleanup any left-over local devices that were still registered as Coupled FBS timing devices the last time that this local host either crashed or was shutdown. For each local device that was registered, a communication message will be sent to each remote host where the Coupled FBS timing device was registered.

When each remote host's FBS daemon receives this communication message, it will attempt to detach any scheduler that is still attached to this no longer valid Coupled FBS timing device, and then to un-register this Coupled FBS timing device on its local system. Upon successful un-registration, the associated **`rdevfs(4)`** device file will no longer exist on that remote host. Note that although any attached scheduler will be detached from the device, the scheduler itself will not be automatically removed from the remote host. The removal of the previously attached scheduler must be manually done with a **`rtcp rms`** command, or by the customer's application code with a **fbsremove** function call.

By default, this option is specified/enabled in the **`/etc/init.d/fbs`** script. If this option is not specified at FBS daemon invocation time, then no attempts to cleanup the remote hosts will be done, and some remote hosts may be left with attached schedulers and stale **`rdevfs(4)`** device files on their systems.

Unregistration of a Coupled FBS Timing Device

One requirement for successfully un-registering a local device from being a Coupled FBS timing device is that no FBS schedulers may be currently attached to that device. If a remote host where a FBS scheduler is attached to that device crashes, then the Coupled FBS support code on the host where the device resides will still contain information regarding the remote scheduler's attachment to this timing device.

When a user or application on the local host attempts to un-register this timing device, the un-registration will fail, since there will still be kernel information that indicates that there is an attached scheduler, even though this scheduler really no longer exists.

For RCIM Coupled timing devices, support has been added in this area. When a timing device un-registration call is made, the kernel Coupled FBS support code will automatically detect when an attached scheduler belongs to a host that is either no longer responding or has been rebooted. In this case, the kernel un-registration code will remove the internal information regarding this scheduler's attachment and then continue on with this timing device's un-registration processing.

Unfortunately, this support cannot be provided for Closely-Coupled timing devices. This is due to the inability of the Coupled FBS kernel support code to reliably detect when an attached scheduler belongs to a remote SBC that has been rebooted. Therefore, for Closely-Coupled timing devices, the local host/SBC must be rebooted to remove this Closely-Coupled timing device registration when this situation occurs.

4
Overview of the Performance Monitor

4
Overview of the Performance Monitor

This chapter provides an overview of the performance monitor. It contains a description of the performance monitor and the capabilities it provides, and it explains the user interface.

What Is the Performance Monitor?

The performance monitor is a feature of PowerMAX OS that allows you to monitor use of the CPU by processes or LWPs that are scheduled on a frequency-based scheduler. The performance monitor relies on the high-resolution timing facility to obtain its timing values (complete information on timing facilities is provided in the *PowerMAX OS Real-Time Guide*).

The performance monitor provides you with the ability to:

- Obtain performance monitor values by process or processor
- Control all performance monitoring features from one processor (that is, enable performance monitoring for any processor)
- Start and stop performance monitoring by process or processor
- Clear performance monitor values by process or processor

You also have the ability to set the timing mode under which the performance monitor is to run. You can select one of two modes: one that includes time spent servicing interrupts in performance monitor timing values and one that excludes time spent servicing interrupts from those values.

When the performance monitor timing mode is set to include interrupt time, a process's user and system times will total the elapsed time that accrues when the process is the currently running process. This elapsed time includes time spent servicing interrupts. Time spent servicing interrupts is added to the process's system time.

When the performance monitor timing mode is set to exclude interrupt time, a process's user and system times will total the time that accrues when the process is the currently running process. This time excludes time spent servicing interrupts.

Whether the timing mode is set to include or exclude interrupt time, context switch time is always included in the new process's system time.

What Values Are Monitored?

The performance monitor keeps track of the time that a process spends running from the time that it is wakened by a frequency–based scheduler until it calls **fbswait**. Time is measured in microseconds. One instance of a process's being wakened by a scheduler is referred to as an iteration or a cycle. Performance monitor values for FBS–scheduled processes are reported both in terms of cycles, or iterations, and in terms of major frames. They reflect what has happened since the last time that performance monitor values were cleared and performance monitoring was enabled.

When performance monitoring is enabled for a single FBS–scheduled process or for all FBS–scheduled processes on a processor, the following types of values are maintained for each process:

Total iterations, cycles	The number of times that the process has been wakened by the scheduler
Last time	The amount of time that the process has spent running from the last time that it has been wakened by the scheduler until it has called **fbswait**
Total time	The total amount of time that the process has spent running in all cycles
Minimum cycle time	The least amount of time that the process has spent running in a cycle
Minimum cycle cycle	The number of the minor cycle in which the minimum cycle time has occurred
Minimum cycle frame	The number of the major frame in which the minimum cycle time has occurred
Maximum cycle time	The greatest amount of time that the process has spent running in a cycle
Maximum cycle cycle	The number of the minor cycle in which the maximum cycle time has occurred
Maximum cycle frame	The number of the major frame in which the maximum cycle time has occurred
Minimum frame time	The least amount of time that the process has spent running during a major frame
Minimum frame frame	The number of the major frame in which the minimum frame time has occurred
Maximum frame time	The greatest amount of time that the process has spent running during a major frame
Maximum frame frame	The number of the major frame in which the maximum frame time has occurred

Number of overruns	The number of times that the process has caused a frame overrun

Monitoring Idle and Spare Time

The performance monitor provides you with the capability of monitoring a processor's idle and spare time. Idle time refers to the time that the CPU is not busy. Spare time is composed of the following:

- Idle time
- CPU time of processes that are not scheduled on a frequency–based scheduler
- CPU time of FBS–scheduled processes for which performance monitoring has not been enabled

It is important to note that PowerMAX OS allows you to monitor a processor's spare time at the same time that you are monitoring its idle time. By monitoring a processor's idle time, you can determine the amount of CPU time that is available to be allocated to additional processes. By subtracting the values obtained for idle time from those obtained for spare time, you can obtain an estimate of the amount of CPU time that is being allocated to processes that are not part of a simulation. You may be able to increase the number of tasks included in a simulation that you are running or run additional simulations.

Procedures for monitoring idle and spare time are described in "How Is Idle Time Monitored?" and "How Is Spare Time Monitored?"

How Is Idle Time Monitored?

You can monitor a particular processor's idle time if you add the process **/idle** to a frequency–based scheduler and schedule it on the desired processor. You can monitor idle time for a number of different processors by adding **/idle** to a selected frequency–based scheduler more than once and scheduling it on a different processor each time. You can also add **/idle** to more than one frequency–based scheduler. It is important to note, however, that you can schedule **/idle** on a particular processor only once.

To add **/idle** to a frequency–based scheduler, you can execute the **rtcp** command **sp**; make a call to **sched_pgmadd** from a C program; make a call to **schedpgmadd** from a FORTRAN program; or make a call to **Sched_PGM_Add** from an Ada program. An explanation of the **sp** command is provided in Chapter 5. Explanations of the **Sched_PGM_Add**, **sched_pgmadd** and **schedpgmadd** routines are provided in Chapter 6, 7, and 8. You can also use NightSim to add **/idle** to a frequency-based scheduler. For complete information on NightSim, refer to the *NightSim Quick Reference*.

When you add **/idle** to a frequency–based scheduler, the only parameter that you must specify is the CPU. The default scheduling priority for **/idle** is zero. The starting base cycle is zero, and the period is one. **/Idle** will be scheduled every minor cycle, starting with the first minor cycle in each major frame.

NOTE

If you are using the subroutine **Sched_PGM_Add**, **sched_pgmadd(3rt)**, or **schedpgmadd(3F77rt)** to add **/idle** to a scheduler, you can set only one bit in the bit mask that specifies the processor. To add **/idle** to a scheduler and schedule it on more than one processor, you must call the subroutine repeatedly, specifying a different processor on each call.

After **/idle** is scheduled, the unique frequency–based scheduler identifier that is known as the process's slot number is returned. You can subsequently use the slot number to identify **/idle** when you are performing tasks related to the frequency–based scheduler or the performance monitor.

You can obtain scheduling information for **/idle** in the same way that you obtain it for other FBS–scheduled processes—by executing the **rtcp** command **vp**; by making a call to **sched_fbsqry** or **sched_pgmqry** from a C program; by making a call to **schedfbsqry** or **schedpgmqry** from a FORTRAN program; or by making a call to **Sched_FBS_Query** or **Sched_PGM_Query** from an Ada program. An explanation of the **vp** command is provided in Chapter 5. Explanations of the Ada, C, and FORTRAN routines are provided in Chapter 6, 7, and 8. You can also use NightSim to obtain scheduling information for **/idle**.

If you enable performance monitoring for the processor(s) on which **/idle** has been scheduled or for the process, itself, you can obtain all of the performance monitor values that are described in "What Values Are Monitored?" Procedures for enabling performance monitoring and retrieving performance monitor values for FBS–scheduled processes are described in detail in Chapter 6, 7, and 8.

How Is Spare Time Monitored?

You can monitor a processor's spare time by adding the process **/spare** to a selected frequency–based scheduler and scheduling it on the desired processor. As in the case of /**idle**, you can monitor spare time for a number of different processors by adding **/spare** to a selected frequency–based scheduler more than once and by scheduling it on a different processor each time. You can also add **/spare** to more than one frequency–based scheduler. It is important to note, however, that you can schedule **/spare** on a particular processor only once.

When you add **/spare** to a frequency–based scheduler, the only parameter that you must specify is the CPU. The default scheduling priority is zero. The starting base cycle is zero, and the period is one. **/Spare** will be scheduled every minor cycle, starting with the first minor cycle in each major frame.

NOTE

If you are using the subroutine **Sched_PGM_Add**, **sched_pgmadd(3rt)**, or **schedpgmadd(3F77rt)** to add **/spare** to a scheduler, you can set only one bit in the bit mask that specifies the processor. To add **/spare** to a scheduler and schedule it on more than one processor, you must call the subroutine repeatedly, specifying a different processor on each call.

After **/spare** is scheduled, the unique frequency–based scheduler identifier that is known as the process's slot number is returned. You can subsequently use the slot number to identify **/spare** when you are performing tasks related to the frequency–based scheduler or the performance monitor.

You can obtain scheduling information for **/spare** in the same way that you obtain it for other FBS–scheduled processes—by executing the **rtcp** command **vp**; by making a call to **sched_fbsqry** or **sched_pgmqry** from a C program; by making a call to **schedfbsqry** or **schedpgmqry** from a FORTRAN program; or by making a call to **Sched_FBS_Query** or **Sched_PGM_Query** from an Ada program. An explanation of the **vp** command is provided in Chapter 5. Explanations of the Ada, C, and FORTRAN routines are provided in Chapter 6, 7, and 8. You can also use NightSim to obtain scheduling information for **/spare**.

If you enable performance monitoring for the processor(s) on which **/spare** has been scheduled or for the process, itself, you can obtain all of the performance monitor values that are described in "What Values Are Monitored?" Procedures for enabling performance monitoring and retrieving performance monitor values for FBS–scheduled processes are described in Chapter 6, 7, and 8.

Optimizing the Performance of a Simulation

One of the benefits of using multiprocessor systems for real–time processing is that you can optimize the performance of a simulation by distributing processes among several processors.

By using the frequency–based scheduler to schedule the programs that make up a simulation, you can use the performance monitor to determine the extent to which the FBS–scheduled processes are utilizing a CPU and to find out whether or not they are running at the frequency that you have specified.

A program is scheduled on a processor when you add it to a frequency–based scheduler. The processor on which it is scheduled is determined by the CPU bias that you specify when you add it to the scheduler. After your programs have been scheduled, you can enable performance monitoring on one or more processes or processors and run your simulation. By examining the performance monitor values that are maintained for each FBS–scheduled process, you can determine the following:

- The processors to which the processes have been assigned
- The amount of time that the processes have spent running
- The processes that have not run at their assigned frequency

If you find that a process is not running at its assigned frequency, you should examine its frequency, the amount of CPU time that is being used by the other processes, and the CPU biases for all processes. Note that if a process's CPU bias identifies more than one processor, you cannot determine how much time the process has spent on a particular CPU specified in the bit mask because of dynamic load balancing. To avoid dynamic load balancing, specify only one processor in the bit mask. By using performance monitoring, you can then tell how much time a process has spent on its assigned CPU. You can redistribute processes as necessary.

You can also enable performance monitoring for a processor's idle and spare time. Procedures for doing so are explained in "Monitoring Idle and Spare Time." By examining the amount of idle and spare time on each processor, you will be able to identify the processors that have the lightest load and calculate the additional amount of CPU time that can be used for scheduling real–time processes.

You can determine the processor assignments that are optimal for your simulation by analyzing the performance monitor values for FBS–scheduled processes and for idle and spare time on selected processors. As necessary, you can redistribute your FBS–scheduled processes by changing their CPU biases. It is important to note that in order to do so, you must first remove the process from the scheduler on which it has been scheduled and then again add it to a scheduler. For an overview of the frequency–based scheduler and the interfaces that accommodate its use, refer to Chapter 2.

Monitoring Unscheduled Processes

The performance monitor provides you with the additional capability of monitoring the performance of unscheduled processes. Unscheduled processes are those that are not wakened by the scheduler and do not call **fbswait**; they are not scheduled to run at a certain frequency. To be able to obtain performance monitor values for such processes, you must first add them to a frequency–based scheduler and specify a starting base cycle of zero and a period of zero. The other scheduling parameters that you must specify include the process's scheduling priority and the CPU on which it is to be scheduled. You can optionally specify an octal value to be passed to a process that is scheduled on a frequency–based scheduler. The "halt on overrun" flag does not apply to an unscheduled process.

You can add unscheduled processes to a frequency–based scheduler by executing the **rtcp** command **sp**; by making a call to **sched_pgmadd** from a C program; by making a call to **schedpgmadd** from a FORTRAN program; or by making a call to **Sched_PGM_Add** from an Ada program. An explanation of the **sp** command is provided in Chapter 5. Explanations of the **Sched_PGM_Add, sched_pgmadd,** and **schedpgmadd** routines are provided in Chapter 6, 7, and 8. You can also use NightSim to add unscheduled processes to a frequency-based scheduler. For complete information on NightSim, refer to the *NightSim Quick Reference*.

After a process is scheduled, the unique frequency–based scheduler identifier that is known as the process's slot number is returned. You can subsequently use the slot number to identify the process when you are performing tasks related to the frequency–based scheduler or the performance monitor.

You can obtain scheduling information for unscheduled processes in the same way that you obtain it for other FBS–scheduled processes—by executing the **rtcp** command **vp**;

by making a call to **sched_fbsqry** or **sched_pgmqry** from a C program; by making a call to **schedfbsqry** or **schedpgmqry** from a FORTRAN program; or by making a call to **Sched_FBS_Query** or **Sched_PGM_Query** from an Ada program. An explanation of the **vp** command is provided in Chapter 5. Explanations of the Ada, C, and FORTRAN routines are provided in Chapter 6, 7, and 8. You can also use NightSim to obtain scheduling information for unscheduled processes.

The performance monitor values that are maintained for unscheduled processes include the following:

- Last time
- Total time
- Minimum frame time and the number of the frame in which it occurred
- Maximum frame time and the number of the frame in which it occurred

You can obtain these values if you enable performance monitoring for the processor(s) on which the processes have been scheduled or if you enable it for the processes, themselves. Procedures for enabling performance monitoring and retrieving performance monitor values are described in detail in Chapter 6, 7, and 8.

Installation and Configuration Requirements

Before using the performance monitor, you must ensure that the **fbs** package is installed on your system. This package provides kernel support for the frequency-based scheduler, the performance monitor, and **rtcp(1)**. For an explanation of the procedures for installing software packages, refer to the *PowerMAX OS Version 4.1 Release Notes* and the **pkgadd(1M)** man page.

You must also ensure that the frequency-based scheduler module (**fbs**) is configured into the kernel. By default, the **fbs** module is not configured. You can use the **config(1M)** utility to (1) determine whether or not the **fbs** module is enabled in your kernel, (2) enable the **fbs** module, and (3) rebuild the kernel. Note that you must be the root user to enable a module and rebuild the kernel. After rebuilding the kernel, you must then reboot your system. For an explanation of the procedures for using **config(1M)**, refer to the "Configuring and Building the Kernel" chapter of *System Administration Volume 2*.

Use of the performance monitor facility also requires that the high-resolution timing facility be configured into the kernel. The high-resolution timing facility provides a means of measuring each process's or LWP's execution time. It is fully described in the *PowerMAX OS Real-Time Guide*. The performance monitor facility uses the times that are gathered for each process by the high–resolution timing facility to obtain its timing values. As specified by the user, these values may include or exclude time spent servicing interrupts.

By default, the high-resolution timing facility is not configured into the kernel. To configure the high-resolution timing facility, you must change the value of the HIGHRESTIMING system tunable parameter from 0 to 1. You can use the **config** utility to (1) determine whether the high-resolution timing facility is configured into you kernel, (2) change the value of the HIGHRESTIMING tunable parameter, and (3) rebuild the kernel. Note that you must be the root user to change the value of a tunable parameter and rebuild the kernel. After rebuilding the kernel, you must then reboot your system.

User Interface

Use of the performance monitor is accommodated by the following: (1) **rtcp**, the real–time command processor; (2) NightSim, a real-time tool that provides a graphical user interface to the frequency-based scheduler and the performance monitor; and (3) a set of library routines that can be called from application programs written in Ada, C, and FORTRAN 77. Each interface is introduced in the sections that follow.

Rtcp

Rtcp, the real-time command processor, allows you to perform key operations associated with the performance monitor by entering commands from the keyboard or invoking a script. These operations include clearing performance monitor values, starting and stopping performance monitoring, setting the timing mode, and querying values. An overview of the real–time command processor and the procedures for using it is provided in Chapter 5.

NightSim

NightSim provides the same capabilities as the real-time command processor **rtcp(1)**. It allows you to perform the entire range of functions associated with the performance monitor. You can perform the major functions of selecting a scheduler, clearing performance monitor values, enabling and disabling performance monitoring, setting the timing mode, and viewing values. Complete information on NightSim is provided in the *NightSim Quick Reference*.

Libraries

The **RT_Interface** package and the C **librt** and FORTRAN **libF77rt** libraries contain subroutines that enable you to perform the entire range of functions associated with the performance monitor. You can perform the key functions of selecting a scheduler, clearing existing performance monitor values, enabling and disabling performance monitoring, setting the timing mode, and retrieving performance monitor values. You can also clear performance monitor values for a single process and retrieve performance monitor values for a single process or a specified list of processes. All of the subroutines that are contained in the **RT_Interface** package and the C and FORTRAN libraries are described in detail in Chapters 6, 7, and 8.

Privileges

PowerMAX OS supports a privilege mechanism through which processes are allowed to perform sensitive system operations or override system restrictions. One of the operations associated with the performance monitor requires that you have the P_RTIME privilege. That operation is selecting the timing mode under which the performance monitor is to run. Specific information related to this privilege requirement is presented in the appropriate sections of this manual. For additional information on privileges, refer to the *PowerMAX OS Programming Guide* and the **`intro(2)`** system manual page.

5
Using Rtcp

5
Using Rtcp

This chapter provides an overview of the real–time command processor, **`rtcp`**. It contains a description of the command processor and the capabilities it provides, an explanation of the modes for executing its commands, and reference information for each command.

What Is the Real–Time Command Processor?

The real–time command processor is a program that acts as a command interpreter. It provides easy access to the major services associated with the frequency–based scheduler and the performance monitor. The real–time command processor allows you to perform key operations by entering commands from the keyboard or invoking a script from the shell command line; it reads the commands and interprets them as requests to execute the related services.

Real–time command processor commands associated with the frequency–based scheduler enable you to perform such key operations as the following: configuring a scheduler, scheduling programs, saving a scheduler configuration, setting up a timing source, running a simulation, and querying status. An overview of the frequency–based scheduler is provided in Chapter 2. It is recommended that you read this chapter prior to using the real–time command processor for the first time.

Real–time command processor commands associated with the performance monitor enable you to perform the following operations: clearing performance monitor values, starting and stopping performance monitoring, and querying values. Overview of the performance monitor is provided in Chapter 4. It is recommended that you also read this chapter prior to using the real–time command processor for the first time.

The real–time command processor has two modes of execution: direct mode and interactive mode. These modes are described in detail in “What Are the Modes of Execution?”

The real–time command processor also has a help facility that makes it possible for you to obtain on–line information about commands and arguments. Procedures for using the help facility are explained in “Getting Help.”

Real–time processor commands are listed and described in Table 5-1. Reference information and procedures for executing each command are provided in “Using Rtcp Commands.”

Table 5-1. Real–Time Processor Commands

Command	Description
ats	Attach timing source to an FBS
chs	Change permissions for an FBS
cs	Configure an FBS
dts	Detach timing source from an FBS
rms	Remove an FBS
svs	Save scheduler configuration
vc	View minor cycle/major frame count
vr	View a **rdevfs** file configuration
vs	View scheduler configuration
rc	Start real–time clock
rd	Register a Coupled FBS device
sc	Stop real–time clock
stc	Set real–time clock values
gtc	Get real–time clock values
start	Start scheduling on an FBS
reg	Register a Closely-Coupled FBS timing device
resume	Resume scheduling on an FBS
stop	Stop scheduling on an FBS
rmp	Remove a process from an FBS
rsp	Reschedule a process
sp	Schedule a process on an FBS
unreg	Unregister a Closely-Coupled FBS timing device
urd	Unregister a Coupled FBS device
vp	View processes on an FBS
pm	Start/stop performance monitoring
cpm	Clear performance monitor values
vcm	View or modify performance monitor timing mode
vpm	View performance monitor values
ex	Exit real–time command processor
he	Display help information

What Are the Modes of Execution?

The real–time command processor provides two modes for executing commands: direct mode and interactive mode. Procedures for using direct mode are explained in “Using Direct Mode.” Procedures for using interactive mode are explained in “Using Interactive Mode.”

Using Direct Mode

Direct mode enables you to invoke real–time command processor commands from the shell command line. You can do so in three ways: (1) by invoking the real–time command processor with a command name and its arguments at the system command prompt; (2) by invoking the real–time command processor at the system command prompt and redirecting the standard input to come from a file instead of the terminal keyboard; and (3) by invoking a script at the system command prompt.

The first method requires that you use the following format for specifying commands:

> **rtcp** *command* [–*option* [*argument*]] [–*option* [*argument*]] . . .

Note that you are allowed to enter only one command on the command line at a time. If you need more than one line to enter a command and its arguments, enter a backslash (\) to cause the line to be continued.

The second method requires that you create a file that contains the real–time command processor commands that you wish to execute. You may do so by using a text editor of your choice or by executing the **svs** (Save Scheduler Configuration) command. Use of the **svs** command is explained in “Svs – Save Scheduler Configuration.”

If you use a text editor to create the file, you must enter each command on a separate line; you may use either of the following formats:

> **rtcp** *command* [–*option* [*argument*]] [–*option* [*argument*]] . . .
>
> or
>
> *command* [–*option* [*argument*]] [–*option* [*argument*]] . . .

If you need more than one line to enter a command and its arguments, enter a backslash (\) to cause the line to be continued.

After you have created the file, invoke the real–time command processor, and redirect the standard input to come from the file by entering a line similar to the following:

```
rtcp < rtcp_input_file
```

The third method also requires that you create a file that contains the real–time command processor commands that you wish to execute. The first line in the file can contain a command or the following text:

> **#!***program_name* where *program_name* specifies the name of the file that contains the shell to be invoked. Typically the name of the file specified is **/sbin/sh** (sig-

nifying the Bourne shell) or **/sbin/csh** (signifying the C shell). It is important to note that the Bourne shell is invoked if the first line of the file contains a command.

Each command must be entered on a separate line according to the following format:

rtcp *command* [–*option* [*argument*]] [–*option* [*argument*]] ...
rtcp *command* [–*option* [*argument*]] [–*option* [*argument*]] ...
...
rtcp *command* [–*option* [*argument*]] [–*option* [*argument*]] ...

If you need more than one line to enter a command and its arguments, enter a backslash (\) to cause the line to be continued.

After you have created the file, you can use the following method to execute it from the shell command line:

Make it an executable file by using the **chmod(1)** command, and then invoke it from the command line as you would any other command (for information on use of the **chmod(1)** command, refer to the corresponding system manual page). Examples are provided by the following:

```
chmod 755 rtcp_script
rtcp_script
```

See Appendix A for an example of a real–time command processor script. For additional information on developing and executing shell scripts, refer to the *User's Guide*.

Using Interactive Mode

Interactive mode makes it possible for you to invoke the real–time command processor, itself, from the shell command line and then to enter the desired commands from within the command processor. To invoke the real–time command processor, first type the following at the system command prompt:

rtcp

The real–time command processor prompt is then displayed as follows:

```
rtcp>
```

At the prompt, type real–time processor commands by using the following format:

rtcp>*command* [–*option* [*argument*]] [–*option* [*argument*]] ...

If you need more than one line to enter a command and its arguments, enter a backslash (\) to cause the line to be continued.

In most instances, if the command is successfully executed, a message is displayed; where applicable, configuration data, scheduling information, or performance monitor values are displayed. Messages and data associated with the commands are included in the reference information that is presented in "Using Rtcp Commands." If an error occurs, a message indicating the nature of the error is displayed. Error messages that may be displayed are listed and described in Appendix B.

When you wish to exit the command processor and return to the shell, type the following:

rtcp>**ex**

The system command prompt is again displayed.

Getting Help

You can access the real–time command processor's help facility in the direct or the interactive mode by using the **he** command. The help information that is provided includes the following:

- A list and brief description of all real–time command processor commands
- A description of each command and the format for entering it
- An explanation of all of the command arguments To display a list of all commands, enter the **he** command as follows:

  he

Commands are displayed as illustrated in Screen 5-1.

```
% rtcp he
       rtcp commands
ats - attach timing source to FBS      chs - modify FBS permissions
cs  - configure FBS                    dts - detach timing source from FBS
rms - remove FBS                       svs - save FBS configuration to a file
vc  - view current frame/cycle count   vs  - view FBS configuration

rc  - run real-time clock              sc  - stop real-time clock
stc - set real-time clock values       gtc - get real-time clock values

start - start FBS                      resume - resume FBS
stop  - stop FBS

rmp - remove a process on a FBS        rsp - reschedule a process on a FBS
sp  - schedule a process on a FBS      vp  - view scheduled process on FBS

cpm - clear performance monitor tables pm  - start/stop performance monitor
vcm - view/modify PM timing mode       vpm - view performance

he  - help
ex  - exit rtcp

%
```

Screen 5-1. Displaying Commands

To display a description of a particular command, enter the **he** command, and specify the name of the command as argument. An example is provided by the following:

he ats

Help information for the **ats** command is displayed as follows:

```
Attach timing source to a FBS

rtcp ats -s scheduler -d device | -e
```

To display an explanation of command arguments, enter the **he** command, and specify the word **option** as argument:

he option

The first screen of help information for arguments is displayed as illustrated in Screen 5-2.

```
% rtcp he option
      rtcp parameters

-a               remove program from FBS and terminate
-b {F|R|O}       scheduling policy
-c cpu_bias      CPU bias (* = all CPUs) (default = current CPU)
-d name          devicename or filename
-e               EOC flag
-f frequency     number of minor cycles to next wakeup  (default = 1)
-i fpid          process fpid number  (default = -1)
-m start_cycle   1st minor cycle to wakeup  (default = 0)
-n proc_name     process name
-o {halt|nohalt} halt FBS on overrun flag (default = nohalt)
-p priority      process priority
-s scheduler     FBS scheduler key
-t {in|ex}       include or exclude interrupt time in pm monitor
-v parameter     process initiation parameter
-x {av|mi|ma|al} performance monitor display option (default = average)

Enter 'he op2' for more parameters

%
```

Screen 5-2. Displaying the First Screen of Arguments

If you wish to display the second screen of help information for arguments, enter the **he** command, and specify **op2** as argument:

he op2

The second screen of information is displayed as illustrated in Screen 5-3.

```
% rtcp he op2
      rtcp parameters

-C cycles/frame  number of minor cycles per major frame
-D duration      clock tick duration  (default = 10us)
-G gid           effective group ID for FBS  (default = current user)
-I permissions   IPC permissions for FBS in octal  (default = 0600)
-L soft_limit    soft overrun limit (default=0)
-M progs/cycle   maximum number of processes per minor cycle
-N progs/fbs     maximum number of processes per FBS
-O clock_ticks   number of clock ticks per minor cycle
-P {ON|OFF}      enable/disable performance monitor (default = OFF)
-R {-1 | 0 | 1}  reset process flag  (default = 0)
-U uid           effective user ID for FBS (default = current user)

%
```

Screen 5-3. Displaying the Second Screen of Arguments

Using Rtcp Commands

This section provides reference information for all of the commands that are supported by the real–time command processor. Commands are presented in the order in which they are described in the help facility (refer to “Getting Help” for a description of the help facility). For each command, the following information is provided:

- A description of the command
- The format for entering the command
- Detailed descriptions of each argument
- An example of the output from the command

Figure 5-1 illustrates the approximate order in which you might execute the commands associated with the frequency–based scheduler.

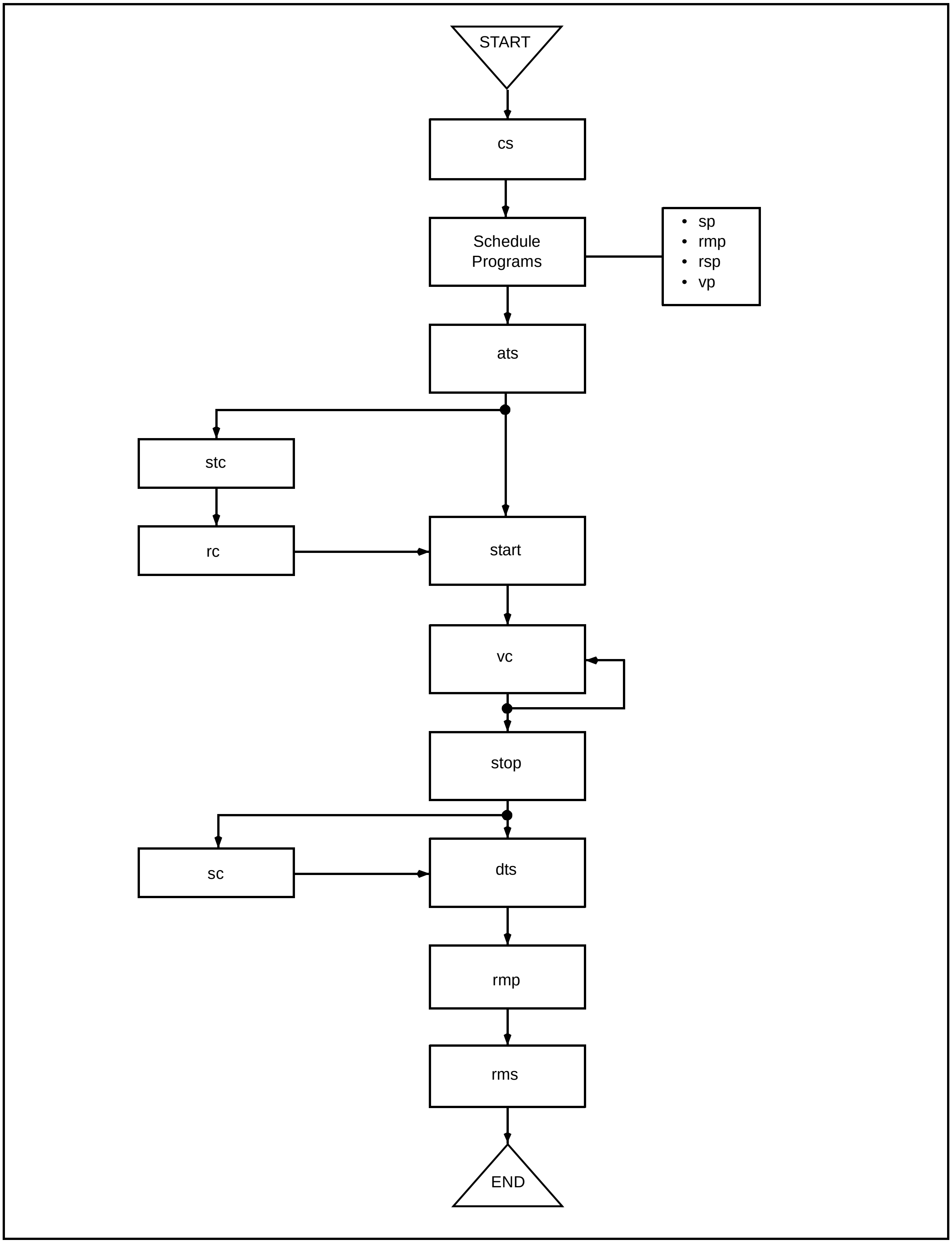

Figure 5-1. FBS Command Sequence

Figure 5-2 illustrates the approximate order in which you might execute the commands associated with the performance monitor.

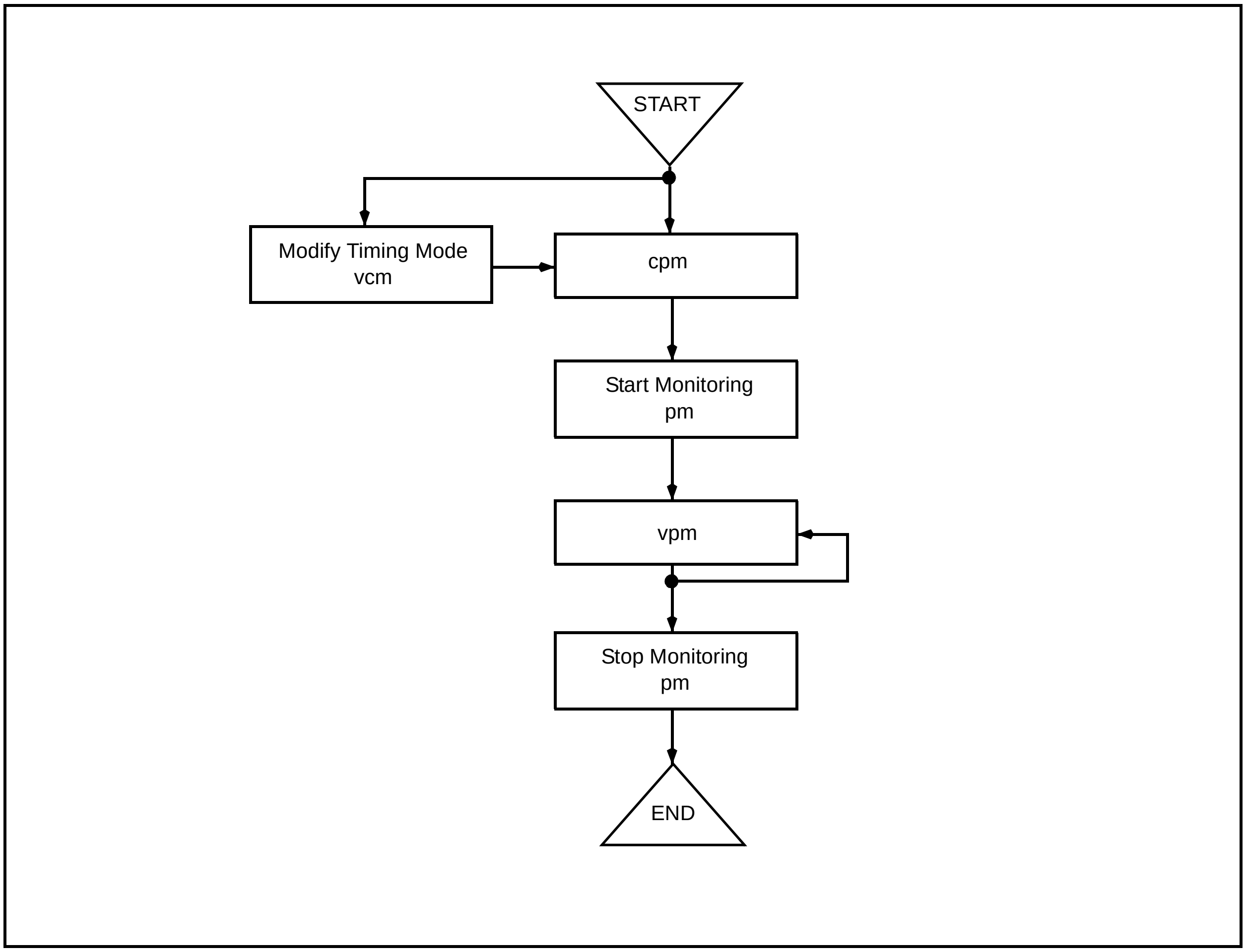

Figure 5-2. Performance Monitor Command Sequence

Ats – Attach Timing Source to an FBS

The **`ats`** command enables you to attach a timing source to a frequency–based scheduler or to specify end–of–cycle scheduling. In the latter case, scheduling is triggered when the last process scheduled during the current minor cycle completes its processing.

NOTE

To use a real–time clock as the timing source for a frequency–based scheduler on a PowerMAX OS system on which the Enhanced Security Utilities are installed, you must have enough privilege to open the device. Refer to the "Trusted Facility Management" chapter of *System Administration Volume 1* for an explanation of the procedures for using devices when the Enhanced Security Utilities are installed.

The format for entering the command, descriptions of the corresponding arguments, and the resulting screen display are presented in the sections that follow.

Format

`ats –s` *scheduler* **`–d`** *device* | **`–e`**

Arguments

Arguments are described as follows.

`–s` *scheduler* — This argument identifies the frequency–based scheduler for which the timing source is to be attached or end–of–cycle scheduling specified. The scheduler must previously have been configured. The *scheduler* argument specifies the numeric key associated with the desired scheduler; it can be any positive integer value.

`–d` *device* | **`–e`** — The **`–d`** *device* argument specifies the path name of the real–time clock, edge–triggered interrupt, or user–supplied device that is being used as the timing source for the specified scheduler. If you are using a real–time clock or an edge–triggered interrupt, you must enter a path name of a certain form. Refer to Chapter 3 for detailed information on the form associated with each type of device. If you are using a user–supplied device, the path name must be a valid UNIX path name. (Refer to Chapter 3 for an explanation of the procedures for using a user–supplied device).

The **`–e`** option specifies end–of–cycle scheduling. In this case, execution of the processes in the next minor cycle will occur when the last process scheduled to run in the current minor cycle finishes its processing for the cycle.

If you are using a Coupled FBS timing device, you must enter the path name of the **`rdevfs`** device file. Refer to Chapter 3 for detailed information on the rdevfs device files that are associated with Coupled FBS timing devices.

Screen Display

If the specified timing source is successfully attached to the scheduler or if end–of–cycle scheduling is successfully enabled, the following message is displayed:

```
Scheduler attached
```

Chs – Change Permissions for an FBS

The **chs** command enables you to change the permissions assigned for a frequency–based scheduler. In order to change the permissions associated with a scheduler, you must have the P_OWNER privilege or have an effective user ID that is equal to that of the creator/owner of the frequency-based scheduler.

If the Enhanced Security Utilities are installed and running, the following conditions must also be met:

- The calling process and the frequency-based scheduler must have identical security levels, or the process must have both P_MACREAD and P_MACWRITE privileges.

The format for entering the command, descriptions of the corresponding arguments, and the resulting screen display are presented in the sections that follow.

Format

chs –s *scheduler* **[–I** *permissions***]** **[–G** *gid***]** **[–U** *uid***]**

Arguments

Arguments are described as follows.

–s *scheduler* — This argument identifies the frequency–based scheduler for which the permissions are to be changed. The scheduler must previously have been configured. The *scheduler* argument specifies the numeric key associated with the desired scheduler; it can be any positive integer value.

–I *permissions* — This argument defines the permissions required for operations related to the specified scheduler (see the system manual page for **intro(2)** for information on permissions associated with the frequency–based scheduler). The *permissions* argument specifies three octal digits—the first indicates permissions granted to the owner, the second those granted to the group, and the third those granted to other users. The octal method for changing permissions associated with a scheduler is the same as that used for specifying *mode* with the **chmod** command (for assistance in using this method, see the system manual page for **chmod(1)**). The default, **600,** grants read and alter (write) permission to the owner only.

–G *gid* — This argument specifies the effective group ID of the selected frequency–based scheduler.

–U *uid* — This argument specifies the effective user ID of the selected frequency–based scheduler.

Screen Display

If the permissions assigned to the scheduler are successfully changed, the following message is displayed:

```
Scheduler permissions changed
```

Cs – Configure an FBS

The **cs** command enables you to create a frequency–based scheduler. Note that to execute this command, the calling process must have the P_RTIME privilege (for additional information on privileges, refer to the *PowerMAX OS Programming Guide* and the **intro(2)** system manual page). It is important to note that the number of schedulers that can be configured at one time cannot exceed the value of FBSMNI, which is the maximum number of schedulers permitted on your system (see Chapter 2 for a description of system tunable parameters).

To create a scheduler, you must specify a key, which is a user–chosen numeric identifier with which the scheduler will be associated. You must also define the following:

1. The number of minor cycles that will compose a major frame on the scheduler
2. The maximum number of tasks that can be scheduled during one minor cycle
3. The maximum number of tasks that can be scheduled on the scheduler at one time

A frequency–based scheduler has associated with it two types of permission that control users' ability to perform scheduler operations: read and alter (write). Read permission is required to perform query operations. Alter permission is required to do the following:

- Schedule, remove, and reschedule programs
- Attach a timing source to and detach it from a scheduler
- Start, stop, and resume scheduling

Permissions are assigned when you create the scheduler. They are specified in the same way in which permissions associated with files are assigned. Refer to the system manual page for **chmod(1)** for assistance in specifying permissions. Refer to the system manual page **intro(2)** for additional information on the permissions associated with a frequency–based scheduler.

When you execute the **cs** command, a unique, positive frequency–based scheduler identifier and corresponding data structure will be created for the specified key if both of the following conditions are met:

- The key is not already associated with a frequency–based scheduler identifier
- The number of frequency–based schedulers already configured is less than the maximum number of schedulers allowed on your system

The newly created frequency–based scheduler identifier will be displayed on your terminal screen.

When you specify a key that is already associated with a frequency–based scheduler, the corresponding frequency–based scheduler identifier will be displayed on your terminal screen if all of the following conditions are met:

- The number of minor cycles specified by the **–C** *cycles/frame* argument matches the number of minor cycles associated with the existing scheduler
- The maximum specified by the **–M** *progs/cycle* argument is less than or equal to the maximum number of processes per minor cycle associated with the existing scheduler
- The maximum specified by the **–N** *progs/fbs* argument is less than or equal to the maximum number of processes allowed on the existing scheduler at one time

If these conditions are not met, an error message will be displayed on the screen.

The **–R** *reset* argument enables you to control the manner in which FBS–scheduled processes are handled when you specify the key for an existing scheduler. If the value of *reset* is the default zero, such processes remain on the scheduler. If the value is **1**, they are removed from the scheduler but allowed to continue executing. If the value is **–1**, they are removed from the scheduler and terminated.

The format for entering the command, descriptions of the corresponding arguments, and the resulting screen display are presented in the sections that follow.

Format

cs *–s scheduler* **–C** *cycles/frame* **–M** *progs/cycle* **–N** *progs/fbs* **[–I** *permissions***]** \
[–R *reset***]**

Arguments

Arguments are described as follows.

–s *scheduler* — This argument specifies a key for the frequency–based scheduler that you wish to create. The key is a user–chosen numeric identifier with which the scheduler will be associated. The value of *scheduler* can be any positive integer value. Note that the number of schedulers that can be configured at one time cannot exceed the value of FBSMNI, which is the maximum number of frequency–based schedulers permitted on your system (see Chapter 2 for a description of system tunable parameters).

–C *cycles/frame* — This argument specifies the number of minor cycles that compose a frame on the specified frequency–based scheduler.

–M *progs/cycle* — This argument specifies the maximum number of programs that can be scheduled to execute during one minor cycle.

–N *progs/fbs* — This argument specifies the maximum number of programs that can be scheduled on the specified scheduler at one time. This value must be less than or equal to the product that is obtained by multiplying the values specified for the *cycles/frame* and the *progs/cycle* arguments.

–I *permissions* — This argument defines the permissions required for operations related to the identified scheduler (see the system manual page for **intro(2)** for information on permissions associated with the frequency–based scheduler). The *permissions* argument specifies three octal digits—the first indicates permissions granted to the owner, the second those granted to the group, and the third those granted to other users. The octal method for setting permissions associated with a scheduler is the same as that used for specifying *mode* with the **chmod** command (for assistance in using this method, see the system manual page for **chmod(1)**). The default, **600,** grants read and alter (write) permission to the owner only.

–R *reset* — This argument specifies the manner in which processes currently scheduled on the specified scheduler are to be handled. The value of the *reset* argument can be 1, –1, or 0. Specify the default **0** if you wish to allow these processes to remain on the scheduler. Specify **1** if you wish to remove these processes from the scheduler but allow them to continue executing. Specify **–1** if you wish to remove these processes from the scheduler and terminate them.

Screen Display

If the scheduler is successfully configured, information similar to the following is displayed:

```
Scheduler 10 has FBS ID of 3
```

Descriptions of the fields presented in this display follow.

Scheduler

This field displays the user–specified key for the selected frequency–based scheduler. It is important to note that this value is required by most of the real–time command processor commands.

FBS ID

This field displays the unique, positive integer value representing the identifier for the selected frequency–based scheduler.

Dts – Detach Timing Source from an FBS

The **dts** command enables you to detach the timing source from a frequency–based scheduler or to disable end–of–cycle scheduling. If the timing source is a real-time clock, it is recommended that you stop the clock prior to invoking this routine. You can do so by making a call to **sc** (see page 5-22 for an explanation of this command).

The format for entering the command, a description of the corresponding arguments, and the resulting screen display are presented in the sections that follow.

Format

dts –s *scheduler*

Arguments

This command requires one argument, which is described as follows.

–s *scheduler*	This argument specifies the frequency–based scheduler for which the timing source is to be detached or end–of–cycle scheduling disabled. The scheduler must previously have been configured. The *scheduler* argument specifies the numeric key associated with the desired scheduler; it can be any positive integer value.

Screen Display

If the timing source is successfully detached from the scheduler or end–of–cycle scheduling is successfully disabled, the following message is displayed:

```
Scheduler detached
```

Rms – Remove an FBS

The **rms** command enables you to remove a frequency–based scheduler. Prior to executing this command, you must ensure that the timing source for the scheduler has been detached or that end–of–cycle scheduling has been disabled (see “Dts – Detach Timing Source from an FBS” for information on use of the **dts** command).

Note that to remove a scheduler, the calling process must have the P_OWNER privilege or an effective user ID that is equal to that of the owner/creator of the frequency-based scheduler.

If the Enhanced Security Utilities are installed and running, the following conditions must also be met:

- The calling process and the frequency-based scheduler must have identical security levels, or the process must have both P_MACREAD and P_MACWRITE privilege.

The format for entering the command, descriptions of the corresponding arguments, and the resulting screen display are presented in the sections that follow.

Format

rms –s *scheduler* [**–a**]

Arguments

Arguments are described as follows.

–s *scheduler* — This argument specifies the frequency–based scheduler that you wish to remove. The scheduler must previously have been configured. The *scheduler* argument specifies the numeric key associated with the desired scheduler; it can be any positive integer value.

–a — This option specifies that all processes currently scheduled on the specified scheduler are to be removed from the scheduler and terminated. If this option is not specified, all processes currently scheduled on the specified scheduler are removed but continue executing.

Screen Display

If the specified scheduler is successfully removed, the following message is displayed:

```
Scheduler removed
```

Svs – Save Scheduler Configuration

The **svs** command enables you to store configuration and scheduling data for a selected frequency–based scheduler in a file. When you execute this task, the data that you have specified in executing the real–time processor commands to configure a scheduler (**cs**), schedule programs on it (**sp**), attach a timing source to it (**ats**), and set the real–time clock (**stc**) are written to the file that you specify.

The format for entering the command, descriptions of the corresponding arguments, and the resulting screen display are presented in the sections that follow.

Format

svs –s *scheduler* **–d** *output_file_name*

Arguments

Arguments are described as follows.

–s *scheduler* — This argument specifies the frequency–based scheduler for which you wish to store configuration and scheduling data. The scheduler must previously have been configured. The *scheduler* argument specifies the numeric key associated with the desired scheduler; it can be any positive integer value.

–d *output_file_name* — This argument specifies a standard UNIX path name identifying the file in which you wish configuration data to be stored. The *output_file_name* argument can be a full or relative path name of up to 1024 characters.

Screen Display

If the scheduler configuration is successfully saved to a file, no message is displayed.

Vc – View Minor Cycle/Major Frame Count

The **vc** command enables you to view the current minor cycle and major frame count values for a frequency–based scheduler. These values help you to determine the progress of a simulation.

The format for entering the command, a description of the corresponding arguments, and the resulting screen display are presented in the sections that follow.

Format

vc –s *scheduler*

Arguments

This command requires one argument, which is described as follows.

–s *scheduler* — This argument specifies the frequency–based scheduler for you which wish to view the current cycle and frame counts. The scheduler must previously have been configured. The *scheduler* argument specifies the numeric key associated with the desired scheduler; it can be any positive integer value.

Screen Display

If the command is successfully executed, information similar to the following is displayed:

```
Major frame = 65 Minor cycle = 25
```

Descriptions of the fields presented in this display follow.

Major frame

This field displays the number of the current major frame for the simulation running on the selected scheduler.

Minor cycle

This field displays the number of the current minor cycle for the simulation running on the selected scheduler.

Vs – View Scheduler Configuration

The **vs** command enables you to view configuration and status information related to a selected frequency–based scheduler. Such information includes the following:

- The key and FBS identifier associated with the scheduler

- The number of minor cycles per major frame, the maximum number of programs per minor cycle, and the maximum number of programs per scheduler
- The user and group IDs of the owner and creator of the scheduler
- The permissions assigned for the scheduler
- The total number of overruns for all processes on the scheduler
- An indication of whether the scheduler is in the run or the stop state
- The CPUs that are active in the system and the CPUs for which performance monitoring has been enabled
- The path name of the device that has been attached to the scheduler

The format for entering the command, a description of the corresponding arguments, and the resulting screen display are presented in the sections that follow.

Format

vs –s *scheduler*

Arguments

This command requires one argument, which is described as follows.

–s *scheduler*	This argument specifies the frequency–based scheduler for which you wish to view current information. The scheduler must previously have been configured. The *scheduler* argument specifies the numeric key associated with the desired scheduler; it can be any positive integer value.

Screen Display

If the command is successfully executed, configuration and status information similar to the following is displayed:

```
Scheduler 99 has FBS ID of 0:    Cycles per frame = 101
Max programs per cycle = 10:    Max programs per fbs = 110
owner uid   = 9999:    owner gid   = 101
creator uid = 9999:    creator gid = 101:
total overruns = 0:    access mode = 600:    flags word = 1
active CPU mask = '----xxxx':    active PM CPU mask = '----x--x'
interrupt device name = /dev/rrtc/0c2
FBS is currently running
```

The following additional information will also be provided if the scheduler is attached to a Closely-Coupled timing device:

```
Closely-Coupled timing device.
Device interrupt source on host: endor
Real device name = /dev/rrtc/0c2
Registered on hosts: endor rudi cosmo orbity
Attached schedulers on hosts: rudi cosmo
SBC id where device resides: 1
SBC id mask of attached FBSs: 0x6
```

The following additional information will also be provided if the scheduler is attached to a RCIM Coupled timing device:

```
RCIM Coupled timing device.
Device interrupt source on host: endor
Real device name = /dev/rrtc/2c1
Registered on hosts: endor rudi cosmo orbity
Attached schedulers on hosts: rudi cosmo
```

Descriptions of the fields presented in this display follow.

Scheduler

This field contains the user–specified key for the selected frequency–based scheduler.

FBS ID

This field contains the unique, positive integer value representing the identifier for the selected frequency–based scheduler.

Cycles per frame

This field indicates the number of minor cycles that compose a major frame on the selected scheduler.

Max programs per cycle

This field indicates the maximum number of programs that can be scheduled in a minor cycle on the selected scheduler.

Max programs per fbs

This field contains the maximum number of programs that can be scheduled on the selected scheduler at one time.

owner uid

This field contains the user ID of the scheduler’s owner.

owner gid

This field contains the group ID of the scheduler’s owner.

creator uid

This field contains the user ID of the scheduler’s creator.

creator gid

This field contains the group ID of the scheduler’s creator.

total overruns

This field indicates the total number of overruns for all processes on the selected scheduler.

access mode

This field contains an octal value indicating the permissions assigned to the selected scheduler.

flags word

If a timing source has been attached to the selected scheduler or if end–of–cycle scheduling has been enabled, this field displays **1**; otherwise, it displays **0**.

active CPU mask

This field contains a mask of the CPUs that are active in the system. The rightmost position corresponds to the first logical CPU. The letter **x** signifies that a CPU is active; the dash (–) signifies that it is not.

active PM CPU mask

This field contains a mask of the CPUs for which performance monitoring has been enabled. The rightmost position corresponds to the first logical CPU. The letter **x** signifies that performance monitoring has been enabled on a CPU; the dash (–) signifies that it has not.

interrupt device name

If a timing source has been attached to the selected scheduler, this field contains the full path name of the device. If end–of–cycle scheduling has been enabled, this field contains the following: `EOC triggering`.

SBC id where device resides

If this scheduler is attached to a Closely-Coupled timing device, then this field contains the SBC board ID where the actual timing device resides.

SBC id mask of attached FBSs

If this scheduler is attached to a Closely-Coupled timing device, then this field contains a SBC board ID bitmask of all SBCs that currently have a frequency-based scheduler attached to this timing device.

real device name

If this scheduler is attached to a Coupled FBS timing device, then this field contains the actual device filename of the timing device on the host where that device is located. If this device is also a RCIM Coupled timing device, then this field will contain the name of the distributed interrupt device:

/dev/reti|eti0[*n*] or **/dev/rrtc/2c[*n*]**

Closely-Coupled timing device

This line will be output if the scheduler is attached to a Closely-Coupled timing device.

RCIM Coupled timing device

This line will be output if the scheduler is attached to a RCIM Coupled timing device.

Device interrupt source on host

When the scheduler is attached to a Coupled FBS timing device then this field contains the hostname of the host where the timing device actually resides.

This line does not appear for Coupled timing devices that were registered with the obsolete **fbs_register_cluster_device** function or **`rtcp reg`** command.

Registered on hosts

When the scheduler is attached to a Coupled FBS timing device then this field contains a list of hostnames where the timing device is registered for use.

This line does not appear for Coupled timing devices that were registered with the obsolete **fbs_register_cluster_device** function or **`rtcp reg`** command.

Attached schedulers on hosts

When the scheduler is attached to a Coupled FBS timing device then this field contains a list of hostnames of the hosts that currently have schedulers attached to this timing device.

This line does not appear for Coupled timing devices that were registered with the obsolete **fbs_register_cluster_device** function or **`rtcp reg`** command.

Rc – Start Real–Time Clock

The **rc** command enables you to start the real–time clock that has been specified as the timing source for a selected frequency–based scheduler (see “Ats – Attach Timing Source to an FBS” for an explanation of the command for attaching a timing source to a scheduler, **`ats`**). Note that you must first have set the count and resolution values for the real–time clock by executing the **`stc`** command (see “Stc – Set Real–Time Clock” for an explanation of this command).

The format for entering the command, a description of the corresponding arguments, and the resulting screen display are presented in the sections that follow.

Format

`rc –s` *scheduler*

Arguments

This command requires one argument, which is described as follows.

`–s` *scheduler* — This argument specifies the frequency–based scheduler for which you wish to start the attached real–time clock. The scheduler must previously have been configured. The *scheduler* argument specifies the numeric key associated with the desired scheduler; it can be any positive integer value.

Screen Display

If the real–time clock is successfully started, the following message is displayed:

```
Clock started
```

Sc – Stop Real–Time Clock

The **sc** command enables you to stop the real–time clock that has been specified as the timing source for a selected frequency–based scheduler.

The format for entering the command, a description of the corresponding arguments, and the resulting screen display are presented in the sections that follow.

Format

sc –s *scheduler*

Arguments

This command requires one argument, which is described as follows.

–s *scheduler* — This argument specifies the frequency–based scheduler for which you wish to stop the attached real–time clock. The scheduler must previously have been configured. The *scheduler* argument specifies the numeric key associated with the desired scheduler; it can be any positive integer value.

Screen Display

If the real–time clock is successfully stopped, the following message is displayed:

```
Clock stopped
```

Stc – Set Real–Time Clock

The **stc** command enables you to establish the duration of a minor cycle by setting the count and duration values for a real–time clock that has been specified as the timing source for a selected frequency–based scheduler.

The format for entering the command, descriptions of the corresponding arguments, and the resulting screen display are presented in the sections that follow.

Format

stc –s *scheduler* [**–D** *clock_duration*] **–O** *clock_count*

Arguments

Arguments are described as follows.

–s *scheduler*	This argument specifies the frequency–based scheduler to which the real–time clock has been attached. The scheduler must previously have been configured. The *scheduler* argument specifies the numeric key associated with the desired scheduler; it can be any positive integer value.
–D *clock_duration*	This argument specifies the duration in microseconds of one clock count. The value of *clock_duration* must be one of the following: 1, 10, 100, 1000, 10000. The default value is 10.
–O *clock_count*	This argument specifies the number of clock counts per minor cycle. The value of *clock_count* can range from one to 65535.

Screen Display

If the real–time clock is successfully set, the following message is displayed:

```
Clock set
```

Gtc – Display Real–Time Clock Settings

The **gtc** command enables you to view the current count and duration values for a real–time clock that has been specified as the timing source for a selected frequency–based scheduler. The clock must previously have been set by using the **stc** command.

The format for entering the command, a description of the corresponding arguments, and the resulting screen display are presented in the sections that follow.

Format

gtc –s *scheduler*

Arguments

This command requires one argument, which is described as follows.

–s *scheduler*	This argument specifies the frequency–based scheduler to which the real–time clock has been attached. The scheduler must previously have been configured. The *scheduler* argument specifies the numeric key associated with the desired scheduler; it can be any positive integer value.

Screen Display

If the real–time clock is currently set, a message similar to the following is displayed:

```
Clock count = 50: duration = 1000
```

Start – Start Scheduling on an FBS

The **start** command enables you to start scheduling processes on a frequency–based scheduler. When you execute this command, the minor cycle, major frame, and overrun count values are set to zero.

Prior to executing this command, you must have executed the **ats** command to attach a timing source to the scheduler or to specify end–of–cycle scheduling (see "Ats – Attach Timing Source to an FBS" for an explanation of this command). If you have specified a real–time clock as the timing source for the scheduler, scheduling will not start until you have set and started the clock (see "Stc – Set Real–Time Clock" and "Rc – Start Real–Time Clock," respectively, for explanations of the **stc** and **rc** commands). If you have specified an edge–triggered interrupt or a user–supplied device as the timing source, it must already be generating interrupts in order for scheduling to start.

The format for entering the command, a description of the corresponding arguments, and the resulting screen display are presented in the sections that follow.

Format

start –s *scheduler*

Arguments

This command requires one argument, which is described as follows.

–s *scheduler*	This argument specifies the frequency–based scheduler on which you wish to start scheduling. The scheduler must previously have been configured. The *scheduler* argument specifies the numeric key associated with the desired scheduler; it can be any positive integer value.

Screen Display

If scheduling on the specified scheduler is successfully started, the following message is displayed:

```
FBS started
```

Resume – Resume Scheduling on an FBS

The **resume** command enables you to resume scheduling of processes on a selected frequency–based scheduler with the major frame, minor cycle, and overrun count values the same as they were when you stopped the scheduler.

The format for entering the command, a description of the corresponding arguments, and the resulting screen display are presented in the sections that follow.

Format

resume –s *scheduler*

Arguments

This command requires one argument, which is described as follows.

-s *scheduler*	This argument specifies the frequency–based scheduler for which you wish to resume scheduling. The scheduler must previously have been configured. The *scheduler* argument specifies the numeric key associated with the desired scheduler; it can be any positive integer value.

Screen Display

If scheduling on the specified scheduler is successfully resumed, the following message is displayed:

```
FBS resumed
```

Stop – Stop Scheduling on an FBS

The **stop** command enables you to stop scheduling of processes on a selected frequency–based scheduler.

The format for entering the command, a description of the corresponding arguments, and the resulting screen display are presented in the sections that follow.

Format

stop -s *scheduler*

Arguments

This command requires one argument, which is described as follows.

-s *scheduler*	This argument specifies the frequency–based scheduler for which you wish to stop scheduling. The scheduler must previously have been configured. The *scheduler* argument specifies the numeric key associated with the desired scheduler; it can be any positive integer value.

Screen Display

If scheduling on the specified scheduler is successfully stopped, the following message is displayed:

```
FBS stopped
```

Rmp – Remove a Process from an FBS

The **rmp** command enables you to remove a process from a frequency–based scheduler. You can identify the process that you wish to remove by using one of the following methods:

- Specify the name of the process and the CPU on which it is scheduled.
- Specify the process's frequency–based scheduler process identifier.
- Specify the name of the process, the CPU on which it is scheduled, and its frequency–based scheduler process identifier.

NOTE

The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency–based scheduler identifier.

The format for entering the command, descriptions of the corresponding arguments, and the resulting screen display are presented in the sections that follow.

Format

`rmp -s` *scheduler* {**`-n`** *proc_name* **`-i`** *fpid*} [**`-c`** *cpu_bias*] [**`-a`**]

Arguments

Arguments are described as follows.

`-s` *scheduler* — This argument specifies the frequency–based scheduler on which the process is scheduled. The scheduler must previously have been configured. The *scheduler* argument specifies the numeric key associated with the desired scheduler; it can be any positive integer value.

`-n` *proc_name* — This argument specifies a standard UNIX path name identifying the process to be removed from the specified scheduler. The *proc_name* argument can be a full or relative path name of up to 1024 characters. If you do not specify this argument, you must provide the frequency–based scheduler process identifier by specifying the **`-i`** *fpid* argument.

`-i` *fpid* — This argument specifies the unique frequency–based scheduler process identifier for the process to be removed. This value is displayed when you successfully schedule a program by executing the **`sp`** command (see "Sp – Schedule a Process on an FBS" for an explanation of this command). If you have not identified the process by name, you must specify this argument. The default value for *fpid* is **–1**. If you accept the default value, you must identify the process by name and CPU.

`-c` *cpu_bias* — This argument enables you to specify the processor(s) to be used in conjunction with the value of the **`-n`** *proc_name* argument to identify the process to be removed from the specified scheduler. The value of *cpu_bias* may be a single CPU ID or a list of CPU IDs. CPU IDs range from zero to seven, where the number **0** represents the first logical CPU, **1** represents the second, and so on. A list of CPU IDs may specify a sequence or a range of numbers––for example, **`-c`** **1,3–5,7**. Note that you must use commas to separate items in the list. You may specify the entire range of CPU IDs (**0–7**) by entering an asterisk (*****). If you do not specify the **–c** *cpu_bias* argument, the default processor is the CPU on which the real–time command processor is currently executing.

–a This option specifies that the process is to be removed from the specified scheduler and terminated. If this option is not specified, the process is removed from the scheduler but allowed to continue executing.

Screen Display

If the specified process is successfully removed from the scheduler, the following message is displayed:

```
Process removed
```

Rsp – Reschedule a Process

The **rsp** command enables you to change the scheduling parameters for a process that has been scheduled on a frequency–based scheduler. You may wish, for example, to change the process's scheduling policy or priority. You may also wish to change the frequency with which the process is scheduled to run. You cannot, however, change the CPU on which it has been scheduled.

If you wish to (1) change a process's scheduling policy to the first-in-first-out (FIFO) or the round-robin scheduling policy or (2) change the priority of a process scheduled under the FIFO or round-robin policy, the following conditions must be met:

- The calling process must have the P_RTIME privilege.
- The effective user ID of the calling process must match the effective user ID of the target process (the process for which the scheduling policy and priority are being set), or the calling process must have the P_OWNER privilege.

If the Enhanced Security Utilities are installed and running, the following additional conditions must be met:

- The Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

If you wish to raise the priority of a process scheduled under the time-sharing policy above a per-process or LWP limit, the following conditions must be met:

- The calling process must have the P_TSHAR privilege.
- The effective user ID of the calling process must match the effective user ID of the target process (the process for which the scheduling priority is being set), or the calling process must have the P_OWNER privilege.

If the Enhanced Security Utilities are installed and running, the following additional conditions must be met:

- The Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

For additional information on privileges, refer to the *PowerMAX OS Programming Guide* and the **intro(2)** system manual page.

You can reschedule a process without first having executed the **rmp** command to remove it from the scheduler (see “Rmp – Remove a Process from an FBS”) or the **stop** command to stop scheduling (see “Stop – Stop Scheduling on an FBS”).

You can identify the process that you wish to reschedule by using one of the following methods:

- Specify the name of the process and the CPU on which it is scheduled.
- Specify the process’s frequency–based scheduler process identifier.
- Specify the name of the process, the CPU on which it is scheduled, and its frequency–based scheduler process identifier.

NOTE

The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency–based scheduler identifier.

The format for entering the command, descriptions of the corresponding arguments, and the resulting screen display are presented in the sections that follow.

Format

rsp -s *scheduler* {**-n** *proc_name* **-i** *fpid*} [**-c** *cpu_bias*] [**-f** *frequency*] \
[**-m** *start_cycle*] [**-b** *policy*] [**-p** *priority*] [**-o** *halt_flag*] [**-L** *soft_limit*]

Arguments

Arguments are described as follows.

-s *scheduler* — This argument specifies the frequency–based scheduler on which the process is scheduled. The scheduler must previously have been configured. The *scheduler* argument specifies the numeric key associated with the desired scheduler; it can be any positive integer value.

-n *proc_name* — This argument specifies a standard UNIX path name identifying the process to be rescheduled. The *proc_name* argument can be a full or relative path name of up to 1024 characters. If you do not specify this argument, you must provide the frequency–based scheduler process identifier by specifying the **-i** *fpid* argument.

-i *fpid* — This argument specifies the unique frequency–based scheduler process identifier for the process to be rescheduled. This value is displayed when you execute the **sp** command (see “Sp – Schedule a Process on an FBS" for an explanation of this command). If you have not identified the process by name, you must specify this argument.

The default value for *fpid* is **-1**. If you use the default value, you must identify the process by name and CPU.

-c *cpu_bias*

This argument enables you to specify the processor(s) to be used in conjunction with the value of the **-n** *proc_name* argument to identify the process to be rescheduled.

The value of *cpu_bias* may be a single CPU ID or a list of CPU IDs. CPU IDs range from zero to seven, where the number **0** represents the first logical CPU, **1** represents the second, and so on. A list of CPU IDs may specify a sequence or a range of numbers––for example, **–c 1,3–5,7**. Note that you must use commas to separate items in the list. You may specify the entire range of CPU IDs (**0–7**) by entering an asterisk (*****).

If you do not specify this argument, the default processor is the CPU on which the real–time command processor is currently executing.

-f *frequency*

This argument enables you to establish the frequency with which the specified process is to be wakened in each major frame. A frequency of one indicates that the specified process is to be wakened every minor cycle; a frequency of two indicates that it is to be wakened once every two minor cycles, a frequency of three once every three minor cycles, and so on. Specify the number of minor cycles representing the frequency with which you wish the process to be wakened. The value of *frequency* can range from one to the number of minor cycles that compose a frame on the scheduler. (The total number of minor cycles per frame is defined by executing the **cs** command, which is explained in "Cs – Configure an FBS.")

-m *start_cycle*

This argument enables you to specify the first minor cycle in which the specified process is to be wakened in each frame. The value of *start_cycle* can range from zero to the total number of minor cycles per frame minus one. The total number of minor cycles per frame is defined by executing the **cs** command, which is explained in "Cs – Configure an FBS."

-b *policy*

This argument enables you to set the POSIX scheduling policy for the specified program. The value of *policy* must be **F, R,** or **O**. Specify **F** to select the first–in–first–out (FIFO) scheduling policy. Specify **R** to select the round–robin scheduling policy. Specify **O** to select the time-sharing scheduling policy.

If you do not specify the **-b** *policy* argument, the default policy is the time-sharing scheduling policy. (For complete information on scheduling policies, refer to the "Process Scheduling and Management" chapter of the *PowerMAX OS Programming Guide*.)

Note: It is recommended that you specify this argument.

-p *priority* — This argument enables you to set the specified process's scheduling priority. The default value is 0.

The range of priority values that you can enter is governed by the value of the policy argument. You can determine the allowable range of priorities associated with each policy (**F, R,** or **O**) by invoking the **run(1)** command from the shell and not specifying any options or arguments (see the corresponding system manual page for an explanation of this command). Higher numerical values correspond to more favorable scheduling priorities.

For complete information on scheduling policies and priorities, refer to the "Process Scheduling and Management" chapter of the *PowerMAX OS Programming Guide*.

-o *halt_flag* — This argument enables you to indicate whether or not the scheduler should be stopped in the event that the specified process causes a frame overrun. The value of *halt_flag* must be either **halt** or **nohalt**. The default is **nohalt**.

-L *soft_limit* — This argument enables you to set the soft overrun limit for the process. The default is 0. If you reschedule a process that already has a non-zero soft overrun limit set and you do not specify a soft overrun limit, the process' soft overrun limit will be set to 0.

Screen Display

If the specified process is successfully rescheduled, scheduling information similar to the following is displayed:

```
CPU   fpid   Priority   Frequency   Start   HaltOnOR   SoftLimit   Process Name

 1    198       53          4         1        N           2       /usr_1/guest/task03
```

Descriptions of the columns presented in this display follow.

CPU

This column contains the identifier for the CPU on which the specified process is scheduled.

fpid

This column contains the unique frequency–based scheduler process identifier for the specified process. This identifier is displayed by the real–time command processor when you schedule a program on the scheduler (see "Sp – Schedule a Process on an FBS" for a description of the **sp** command).

Priority

This column indicates the scheduling priority of the specified process.

Frequency

This column indicates the frequency with which the specified process is scheduled to be wakened in each major frame.

Start

This column indicates the first minor cycle in which the specified process is scheduled to be wakened in each major frame.

HaltOnOR

This column indicates whether or not the “halt on overrun” flag has been set for the specified process.

SoftLimit

This column indicates the process’ soft overrun limit

Process Name

This column contains the full path name of the process that has been scheduled on the selected frequency–based scheduler.

Sp – Schedule a Process on an FBS

The **`sp`** command enables you to create a process and schedule it on a frequency–based scheduler. If you execute this command and you wish to (1) change a process’s scheduling policy to the first-in-first-out (FIFO) or the round-robin policy or (2) change the priority of a process scheduled under the FIFO or the round-robin policy, the following conditions must be met:

- The calling process must have the P_RTIME privilege.
- The effective user ID of the calling process must match the effective user ID of the target process (the process for which the scheduling policy and priority are being set), or the calling process must have the P_OWNER privilege.

If the Enhanced Security Utilities are installed and running, the following additional conditions must be met:

- The Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

If you wish to raise the priority of a process scheduled under the time-sharing policy above a per-process or LWP limit, the following conditions must be met:

- The calling process must have the P_TSHAR privilege.
- The effective user ID of the calling process must match the effective user ID of the target process (the process for which the scheduling priority is being set), or the calling process must have the P_OWNER privilege.

If the Enhanced Security Utilities are installed and running, the following additional conditions must be met:

- The Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

If you wish to modify the process's CPU bias when you invoke this command, the following conditions must be met:

- The real or effective user ID of the calling process must match the real or saved user ID of the process for which the CPU assignment is being changed, or the calling process must have the P_OWNER privilege.
- To add a CPU to a process's CPU bias, the calling process must have the P_CPUBIAS privilege.

If the Enhanced Security Utilities are installed and running, the following additional conditions must be met:

- The Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

For additional information on privileges, refer to the *PowerMAX OS Programming Guide* and the **`intro(2)`** system manual page.

When you execute this command, the real–time command processor returns a unique frequency–based scheduler process identifier. You can subsequently use this identifier to specify the process when you are executing other commands.

The format for entering the command, descriptions of the corresponding arguments, and the resulting screen display are presented in the sections that follow.

Format

`sp -s` *scheduler* **`-n`** *proc_name* [**`-c`** *cpu_bias*] [**`-f`** *frequency*] [**`-m`** *start_cycle*] [**`-b`** *policy*] \
[**`-p`** *priority*] [**`-v`** *parameter*] [**`-o`** *halt_flag*] [**`-L`** *soft_limit*]

Arguments

Arguments are described as follows.

`-s` *scheduler* — This argument specifies the frequency–based scheduler on which the process is to be scheduled. The scheduler must previously have been configured. The *scheduler* argument specifies the numeric key associated with the desired scheduler; it can be any positive integer value.

`-n` *proc_name* — This argument specifies a standard UNIX path name identifying the program that you wish to schedule. The *proc_name* argument can be a full or relative path name of up to 1024 characters.

`-c` *cpu_bias* — This argument specifies the CPU bias for the specified program. The CPU bias determines the processor or processors on which the program can be scheduled.

The value of *cpu_bias* may be a single CPU ID or a list of CPU IDs. CPU IDs range from zero to seven, where the number **0** represents the first logical CPU, **1** represents the second, and so on. A list of CPU IDs may specify a sequence or a range of numbers––for example, **–c 1,3–5,7**. Note that you must use commas to separate items in the list. You may specify the entire range of CPU IDs (**0–7**) by entering an asterisk (*).

If you do not specify this argument, the default processor is the CPU on which the real–time command processor is currently executing.

–f *frequency*
: This argument enables you to establish the frequency with which the specified process is to be wakened in each major frame. A frequency of one indicates that the specified process is to be wakened every minor cycle; a frequency of two indicates that it is to be wakened once every two minor cycles, a frequency of three once every three minor cycles, and so on. Specify the number of minor cycles representing the frequency with which you wish the process to be wakened. The value of *frequency* can range from one to the number of minor cycles that compose a frame on the scheduler. The default value is one. (The total number of minor cycles per frame is defined by executing the **cs** command, which is explained in "Cs – Configure an FBS.")

–m *start_cycle*
: This argument enables you to specify the first minor cycle in which the specified process is scheduled to be wakened in each frame. The value of *start_cycle* can range from zero to the total number of minor cycles per frame minus one. The default value is zero. (The total number of minor cycles per frame is defined by executing the **cs** command, which is explained in "Cs – Configure an FBS.")

–b *policy*
: This argument enables you to set the POSIX scheduling policy for the specified process. The value of *policy* must be **F, R,** or **O**. Specify **F** to select the first–in–first–out (FIFO) scheduling policy. Specify **R** to select the round–robin scheduling policy. Specify **O** to select the time-sharing scheduling policy.

If you do not specify the **-b** *policy* argument, the default policy is the time-sharing scheduling policy. (For complete information on scheduling policies, refer to the "Process Scheduling and Management" chapter of the *PowerMAX OS Programming Guide*.)

Note: It is recommended that you specify this argument.

–p *priority*
: This argument enables you to set the specified process's scheduling priority. The default value is 0.

The range of priority values that you can enter is governed by the value of the policy argument. You can determine the allowable range of priorities associated with each policy (**F, R,** or **O**) by invoking the **run(1)** command from the shell and not specifying any options or arguments (see the corresponding system manual page for an explanation of this command). Higher numerical values correspond to more favorable scheduling priorities.

For complete information on scheduling policies and priorities, refer to the "Process Scheduling and Management" chapter of the *PowerMAX OS Programming Guide*.

-v *parameter* — This argument enables you to pass an integer value to a process that is scheduled on a frequency–based scheduler. The value of *parameter* must be a 32–bit decimal number.

-o *halt_flag* — This argument enables you to indicate whether or not the scheduler should be stopped in the event that the specified program causes a frame overrun. The value of *halt_flag* must be either **halt** or **nohalt**. The default is **nohalt**.

-L *soft_limit* — This argument enables you to set the soft overrun limit for the process. The default is 0.

Screen Display

If the specified process is successfully scheduled on the frequency–based scheduler, information similar to the following is displayed:

```
fpid 199 assigned to process task02
```

Descriptions of the fields presented in this display follow.

fpid

This field displays the unique frequency–based scheduler process identifier assigned to the specified process.

process

This field displays the full path name of the process that has been scheduled on the selected frequency–based scheduler.

Vp – View Processes on an FBS

The **vp** command enables you to display information about a particular FBS-scheduled process or all FBS-scheduled processes on one or more processors on a selected frequency-based scheduler.

You can display information about all FBS–scheduled processes on a specified processor or all processors by specifying the scheduler and the CPU(s) on which the processes are scheduled.

If you wish to display information for a particular FBS–scheduled process, you can identify the process by specifying the name of the process and the CPU on which it is scheduled or by specifying the process's frequency–based scheduler process identifier. In the first case, you can use the default CPU, specify a particular CPU, or specify all CPUs.

NOTE

The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency–based scheduler identifier.

Information displayed for each process includes the following:

- The CPU on which the process is executing
- The frequency–based scheduler process identifier
- The process's scheduling priority
- The frequency (the number of minor cycles indicating the frequency with which the process is wakened in each major frame)
- The starting base cycle (the first minor cycle in which the process is scheduled to be wakened in each major frame)
- The value of the "halt on overrun" flag
- The process' soft overrun limit
- The process's path name

The format for entering the command, descriptions of the corresponding arguments, and the resulting screen display are presented in the sections that follow.

Format

vp –s *scheduler* [[**–n** *proc_name*] [**–i** *fpid*] [**–c** *cpu_bias*]] | [**–c** *cpu_bias*]

Arguments

Arguments are described as follows.

–s *scheduler* — This argument specifies the frequency–based scheduler for which scheduling information is to be displayed. The scheduler must previously have been configured. The *scheduler* argument specifies the numeric key associated with the desired scheduler; it can be any positive integer value.

–n *proc_name* — This argument specifies a standard UNIX path name identifying a particular process for which scheduling information is to be displayed. The *proc_name* argument can be a full or relative path name of up to 1024 characters.

–i *fpid* — This argument specifies the unique frequency–based scheduler process identifier for a particular process for which scheduling information is to be displayed. This value is displayed when you successfully schedule a program by executing the **sp** command (see "Sp – Schedule a Process on an FBS" for an explanation of this command). The default value for *fpid* is **–1**.

–c *cpu_bias* — This argument specifies the processor(s) for which scheduling information is to be displayed.

The value of *cpu_bias* may be a single CPU ID or a list of CPU IDs. CPU IDs range from zero to seven, where the number **0** represents the first logical CPU, **1** represents the second, and so on. A list of CPU IDs may specify a sequence or a range of numbers—for example, **–c 1,3–5,7**. Note that you must use commas to separate items in the list. You may specify the entire range of CPU IDs (**0–7**) by entering an asterisk (*****).

The default processor is the CPU on which the real–time command processor is currently executing.

Screen Display

If the command is successfully executed, scheduling information similar to the following is displayed:

```
CPU    fpid   Priority  Frequency   Start   HaltOnOR  SoftLimit  Process Name

 1      199      55         1         0        Y          0      /usr_1/guest/task02

 1      198      53         4         1        N          3      /usr_1/guest/task03

 2      197      53         2         0        Y          1      /usr_1/guest/task04
```

Descriptions of the columns presented in this display follow.

CPU

This column contains the identifiers for the CPUs on which the respective processes are scheduled.

fpid

This column contains the unique frequency–based scheduler process identifiers for the respective processes. This identifier is displayed by the real–time command processor when you schedule a program on the scheduler (see “Sp – Schedule a Process on an FBS” for a description of the **sp** command).

Priority

This column indicates the scheduling priorities of the respective processes.

Frequency

This column indicates the frequency with which the respective processes are scheduled to be wakened in each major frame.

Start

This column indicates the first minor cycle in which the respective processes are scheduled to be wakened in each major frame.

HaltOnOR

This column indicates whether or not the “halt on overrun” flag has been set for the respective processes.

SoftLimit

This column indicates the process' soft overrun limit.

Process Name

This column contains the full path names of the processes that have been scheduled on the selected frequency–based scheduler.

Cpm – Clear Performance Monitor Values

The **cpm** command enables you to clear performance monitor values for a particular FBS–scheduled process or all FBS–scheduled processes on one or more processors on a selected frequency–based scheduler.

You can clear values for all FBS–scheduled processes on a specified processor or all processors by specifying the scheduler and the CPU(s) on which the processes are scheduled.

If you wish to clear performance monitor values for a particular FBS–scheduled process, you can identify the process by specifying the name of the process and the CPU on which it is scheduled or by specifying the process's frequency–based scheduler process identifier. In the first case, you can use the default CPU, specify a particular CPU, or specify all CPUs.

The format for entering the command, descriptions of the corresponding arguments, and the resulting screen display are presented in the sections that follow.

Format

cpm –s *scheduler* [[**–n** *proc_name*] [**–i** *fpid*] [**–c** *cpu_bias*]] | [**–c** *cpu_bias*]

Arguments

Arguments are described as follows.

–s *scheduler* — This argument specifies the frequency–based scheduler on which the process or processes are scheduled. The scheduler must previously have been configured. The *scheduler* argument specifies the numeric key associated with the desired scheduler; it can be any positive integer value.

–n *proc_name* — This argument specifies a standard UNIX path name identifying a particular process for which values are to be cleared. The *proc_name* argument can be a full or relative path name of up to 1024 characters.

–i *fpid* — This argument specifies the unique frequency–based scheduler process identifier for a particular process for which values are to be cleared. This value is displayed when you execute the **sp** command (see "Sp – Schedule a Process on an FBS" for an explanation of this command). The default value for *fpid* is **–1**.

–c *cpu_bias* — This argument specifies the processor(s) for which performance monitor values are to be cleared.

The value of *cpu_bias* may be a single CPU ID or a list of CPU IDs. CPU IDs range from zero to seven, where the number **0** represents the first logical CPU, **1** represents the second, and so on. A list of CPU IDs may specify a sequence or a range of numbers—for example, **–c 1,3–5,7**. Note that you must use commas to separate items in the list. You may specify the entire range of CPU IDs (**0–7**) by entering an asterisk (*****).

The default processor is the CPU on which the real–time command processor is currently executing.

Screen Display

If the performance monitor values are successfully cleared, the following message is displayed:

```
Performance monitor values cleared
```

Note

This command clears the soft overrun count for all processes specified by the user.

Pm – Start/Stop Performance Monitoring

The **pm** command enables you to start or stop performance monitoring for a particular FBS–scheduled process or all FBS–scheduled processes on one or more processors on a selected frequency–based scheduler.

You can start or stop performance monitoring for all FBS–scheduled processes on a specified processor or all processors by specifying the scheduler and the CPU(s) on which the processes are scheduled.

If you wish to start or stop performance monitoring for a particular FBS–scheduled process, you can identify the process by specifying the name of the process and the CPU on which it is scheduled or by specifying the process's frequency–based scheduler process identifier. In the first case, you can use the default CPU, specify a particular CPU, or specify all CPUs.

The format for entering the command, descriptions of the corresponding arguments, and the resulting screen display are presented in the sections that follow.

Format

pm –s *scheduler* [[**–n** *proc_name*] [**–i** *fpid*] [**–c** *cpu_bias*]] | [**–c** *cpu_bias*] \
[**–P** *pm_flag*]

Arguments

Arguments are described as follows.

-s *scheduler* — This argument specifies the frequency–based scheduler on which the process or processes are scheduled. The scheduler must previously have been configured. The *scheduler* argument specifies the numeric key associated with the desired scheduler; it can be any positive integer value.

-n *proc_name* — This argument specifies a standard UNIX path name identifying a particular process for which performance monitoring is to be started or stopped. The *proc_name* argument can be a full or relative path name of up to 1024 characters.

-i *fpid* — This argument specifies the unique frequency–based scheduler process identifier for a particular process for which performance monitoring is to be started or stopped. This value is displayed when you execute the **sp** command (see "Sp – Schedule a Process on an FBS" for an explanation of this command). The default value for *fpid* is **-1**.

-c *cpu_bias* — This argument specifies the processor(s) for which performance monitoring is to be started or stopped.

The value of *cpu_bias* may be a single CPU ID or a list of CPU IDs. CPU IDs range from zero to seven, where the number **0** represents the first logical CPU, **1** represents the second, and so on. A list of CPU IDs may specify a sequence or a range of numbers—for example, **-c 1,3–5,7**. Note that you must use commas to separate items in the list. You may specify the entire range of CPU IDs (**0–7**) by entering an asterisk (*****).

The default processor is the CPU on which the real–time command processor is currently executing.

-P *pm_flag* — This argument enables you to indicate whether performance monitoring is to be started or stopped. The value of *pm_flag* must be either **ON** or **OFF**. The default is **OFF**.

Screen Display

If performance monitoring is successfully started, the following message is displayed:

```
Performance monitoring enabled
```

If performance monitoring is successfully stopped, the following message is displayed:

```
Performance monitoring disabled
```

Vcm – View/Modify Performance Monitor Timing Mode

The **vcm** command enables you to view or modify the performance monitor timing mode. The timing mode can be set to include or exclude time spent servicing interrupts from the performance monitor timing values. Note that to set the timing mode, the calling process must have the P_RTIME privilege (for additional information on privileges, refer to the **intro(2)** system manual and the *PowerMAX OS Programming Guide).*

CAUTION

The timing mode for the high–resolution timing facility is set system–wide. It affects all processes running on all CPUs.

The format for entering the command, descriptions of the corresponding arguments, and the resulting screen display are presented in the sections that follow.

Format

vcm [**–t** *pm_tmode*]

Arguments

No argument is required to view the performance monitor timing mode.

One argument is required to modify the timing mode; it is described as follows:

–t *pm_tmode* — This argument specifies whether interrupt time is to be included in or excluded from performance monitor timing values. The value of *pm_tmode* must be either **in** or **ex**.

Screen Display

If you successfully execute the **vcm** command to view the performance monitor timing mode, one of the following messages is displayed:

```
PM timing mode includes interrupt times.
```

or

```
PM timing mode excludes interrupt times.
```

Vpm – View Performance Monitor Values

The **vpm** command enables you to display performance monitor values for a particular FBS–scheduled process or all FBS–scheduled processes on one or more processors on a selected frequency–based scheduler.

You can display values for all FBS–scheduled processes on a specified processor or all processors by specifying the scheduler and the CPU(s) on which the processes are scheduled.

If you wish to display performance monitor values for a particular FBS–scheduled process, you can identify the process by specifying the name of the process and the CPU on which it is scheduled or by specifying the process's frequency–based scheduler process identifier. In the first case, you can use the default CPU, specify a particular CPU, or specify all CPUs.

The format for entering the command, descriptions of the corresponding arguments, and the resulting screen displays are presented in the sections that follow.

Format

vpm –s *scheduler* [[**–n** *proc_name*] [**–i** *fpid*] [**–c** *cpu_bias*]] | [**–c** *cpu_bias*] \
[**–x** *pm_output*]

Arguments

Arguments are described as follows.

–s *scheduler* — This argument specifies the frequency–based scheduler on which the process or processes are scheduled. The scheduler must previously have been configured. The *scheduler* argument specifies the numeric key associated with the desired scheduler; it can be any positive integer value.

–n *proc_name* — This argument specifies a standard UNIX path name identifying a particular process for which performance monitor values are to be displayed. The *proc_name* argument can be a full or relative path name of up to 1024 characters.

–i *fpid* — This argument specifies the unique frequency–based scheduler process identifier for a particular process for which performance monitor values are to be displayed. This value is displayed when you execute the **sp** command (see "Sp – Schedule a Process on an FBS" for an explanation of this command). The default value for *fpid* is **–1**.

–c *cpu_bias* — This argument specifies the processor(s) for which performance monitor values are to be displayed.

The value of *cpu_bias* may be a single CPU ID or a list of CPU IDs. CPU IDs range from zero to seven, where the number **0** represents the first logical CPU, **1** represents the second, and so on. A list of CPU IDs may specify a sequence or a range of numbers—for example, **–c 1,3–5,7**. Note that you must use commas to separate items in the list. You may specify the entire range of CPU IDs (**0–7**) by entering an asterisk (*****).

The default processor is the CPU on which the real–time command processor is currently executing.

–x *pm_output* This argument enables you to indicate the type of values that you wish to display. Four types of values may be specified: average, minimum, maximum, or all.

If you select **`average`**, the following values are displayed: the number of iterations, or cycles; the last time; the total time; the average time; and the number of overruns.

If you select **`minimum`**, the following values are displayed: the number of iterations; the minimum cycle time and the number of the minor cycle and the major frame in which it has occurred; and the minimum frame time and the number of the major frame in which it has occurred.

If you select **`maximum`**, the following values are displayed: the number of iterations; the maximum cycle time and the number of the minor cycle and major frame in which it has occurred; and the maximum frame time and the number of the major frame in which it has occurred.

If you select **`all`**, average, minimum, and maximum values are displayed.

All times are reported in microseconds. You may specify the *pm_output* argument by simply entering the first two letters of the argument: **`av`** for average, **`mi`** for minimum, **`ma`** for maximum, or **`al`** for all. The default value is **`average`**.**Screen Display**

If you select **`average`** values and the command is successfully executed, performance monitor values similar to the following are displayed:

					OverRuns	
fpid	Iterations	TimeLast(us)	TotalTime(us)	Average(us)	Hard	Soft
199	480	1023	499200	1040	0	0
198	120	2013	240960	2008	1	2
197	240	1521	378960	1579	0	3

Descriptions of the columns presented in this display follow.

fpid

This column contains the unique frequency–based scheduler process identifiers for the respective processes. This identifier is displayed by the real–time command processor when you schedule a program on the scheduler (see “Sp – Schedule a Process on an FBS” for a description of the **sp** command).

Iterations

This column contains the number of times that the respective processes have been wakened by the frequency–based scheduler since the last time that performance monitor values have been cleared and performance monitoring has been enabled.

TimeLast(us)

This column contains the amount of time that the respective processes have spent running from the last time that they were wakened by the frequency–based scheduler until they called **`fbswait`**.

TotalTime(us)

This column contains the cumulative times that the respective processes have spent running in all cycles, or iterations.

Average(us)

This column contains the average amount of time that the respective processes have spent running in all cycles, or iterations. These values are obtained by dividing the values reported in the Total Time column by the values reported in the Iterations column.

Overruns

Hard This column contains the number of times that the respective processes have caused a catastrophic frame overrun.

Soft This column contains the number of times that the respective processes have caused a non-catastrophic frame overrun.

If you select **`minimum`** values and the command is successfully executed, performance monitor values similar to the following are displayed:

```
                            Minimum Cycle                 Minimum Frame
fpid        Iterations    time(us)    Frame/Cycle     time(us)        Frame
 199            30          1002          6/7            8013           17
 198            30          1943          2/1            3995           22
 197            30          1312          1/2            5314           11
```

Descriptions of the columns presented in this display follow.

fpid

This column contains the unique frequency–based scheduler process identifiers for the respective processes. This identifier is displayed by the real–time command processor when you schedule a program on the scheduler (see "Sp – Schedule a Process on an FBS" for a description of the **`sp`** command).

Iterations

This column contains the number of times that the respective processes have been wakened by the frequency–based scheduler since the last time that performance monitor values have been cleared and performance monitoring has been enabled.

Minimum Cycle time(us)

This column contains the least amount of time that the respective processes have spent running in a cycle, or iteration.

Minimum Cycle Frame/Cycle

These columns contain the number of the major frame and the minor cycle in which the minimum cycle time has occurred.

Minimum Frame time(us)

This column contains the least amount of time that the respective processes have spent running during a major frame.

Minimum Frame Frame

This column contains the number of the major frame in which the minimum frame time has occurred.

If you select **`maximum`** values and the command is successfully executed, performance monitor values similar to the following are displayed:

		Maximum Cycle		Maximum Frame	
fpid	Iterations	time(us)	Frame/Cycle	time(us)	Frame
199	30	1303	2/4	9502	12
198	30	2201	7/5	4391	17
197	30	1917	9/6	7431	24

Descriptions of the columns presented in this display follow.

fpid

This column contains the unique frequency–based scheduler process identifiers for the respective processes. This identifier is displayed by the real–time command processor when you schedule a program on the scheduler (see “Sp – Schedule a Process on an FBS” for a description of the **sp** command).

Iterations

This column contains the number of times that the respective processes have been wakened by the frequency–based scheduler since the last time that performance monitor values have been cleared and performance monitoring has been enabled.

Maximum Cycle time(us)

This column contains the greatest amount of time that the respective processes have spent running in a cycle, or iteration.

Maximum Cycle Frame/Cycle

These columns contain the number of the major frame and the minor cycle in which the maximum cycle time has occurred.

Maximum Frame time(us)

This column contains the greatest amount of time that the respective processes have spent running during a major frame.

Maximum Frame Frame

This column contains the number of the major frame in which the maximum frame time has occurred.

If you select **`all`** values and the command is successfully executed, the screen displays for average, minimum, and maximum values are displayed.

Ex – Exit Real–Time Command Processor

The **ex** command is used only when you are using the real–time command processor in interactive mode. It enables you to exit the command processor and return to the shell.

The command is entered is as follows:

`ex`

If the command is successfully executed, the system command prompt is displayed.

He – Display Help Information

The **`he`** command enables you to display help information for the real–time command processor. You may obtain the following types of information:

- A list of all commands
- An explanation of a particular command
- A list of all command options

The format for entering the command, descriptions of the corresponding arguments, and the resulting screen displays are presented in the sections that follow.

Format

he [*command* | **option** | **op2**]

Arguments

Arguments are described as follows.

command This argument specifies the command for which you wish to obtain an explanation.

option This argument specifies that the first screen of command options is to be displayed.

op2 This argument specifies that the second screen of command options is to be displayed.

Screen Display

If you specify the **he** command without an argument, Screen 5-4 is displayed (Night Hawk systen output shown).

```
% rtcp he
       rtcp commands
ats - attach timing source to FBS       chs - modify FBS permissions
cs  - configure FBS                     dts - detach timing source from FBS
rms - remove FBS                        svs - save FBS configuration to a file
vc  - view current frame/cycle count    vs  - view FBS configuration

rc  - run real-time clock               sc  - stop real-time clock
stc - set real-time clock values        gtc - get real-time clock values

start - start FBS                       resume - resume FBS
stop  - stop FBS

rmp - remove a process on a FBS         rsp - reschedule a process on a FBS
sp  - schedule a process on a FBS       vp  - view scheduled process on FBS

cpu - clear performance monitor tables  pm  - start/stop performance monitor
vcm - view/modify PM timing mode        vpm - view performance

he  - help
ex  - exit rtcp

%
```

Screen 5-4. Output from the he Command

If you specify the **he** command with the *command* argument, help information similar to the following is displayed:

```
Change FBS permissions

rtcp chs –s scheduler –I permissions [–G gid] [–U uid]
```

If you specify the **`he`** command with the **`option`** argument, Screen 5-5 is displayed.

```
% rtcp he option
      rtcp parameters

-a                remove program from FBS and terminate
-b {F|R|O}        scheduling policy
-c cpu_bias       CPU bias (* = all CPUs) (default = current CPU)
-d name           devicename or filename
-e                EOC flag
-f frequency      number of minor cycles to next wakeup  (default = 1)
-i fpid           process fpid number  (default = -1)
-m start_cycle    1st minor cycle to wakeup  (default = 0)
-n proc_name      process name
-o {halt|nohalt}  halt FBS on overrun flag (default = nohalt)
-p priority       process priority
-s scheduler      FBS scheduler key
-t {in|ex}        include or exclude interrupt time in pm monitor
-v parameter      process initiation parameter
-x {av|mi|ma|al}  performance monitor display option (default = average)

Enter 'he op2' for more parameters

%
```

Screen 5-5. Output from the he option Command

If you specify the **`he`** command with the **`op2`** argument, the second screen of arguments will be displayed.

Rd - Register a Coupled FBS Timing Device

The **`rd`** command enables you to register a device on the calling local host as a Coupled FBS timing device. Once registered, the device is then available for use on all hosts where the device is registered. In order to register a device, you must have the P_RTIME privilege as well as enough privilege to open the device file.

The format for entering the command and a description of the corresponding arguments are presented in the sections that follow.

Format

`rd -T c | r -d` *device* **`-H hostname_list`**

Arguments

Arguments are described as follows:

`-T c | r` — This argument specifies the type of timing device to be registered. A **`-T c`** indicates that a Closely-Coupled timing device is being registered, and a **`-T r`** indicates that a RCIM Coupled timing device is being registered.

`-d` *device* — This argument specifies the path name of the device that is being registered as a Coupled FBS timing device. If you are using a real-time clock or RCIM device, then you must enter the path name of this device. Refer to Chapter 3 for detailed information about the path names for these types of devices. If you are using a user-supplied device, the path name must be a valid UNIX path name. Refer to Chapter 3 for an explanation of the procedures for using a user-supplied device.

-H hostname_list This argument specifies the hosts where the device is to be registered. The hostnames should be listed with no blanks and separated with commas. For example:

> **-H** *endor, rudi, cosmo, orbity*

The name of the local host where the device actually resides **must** be in this hostname list. Only those remote hosts that plan to attach a scheduler to this timing device need to be in the **hostname_list**.

When successful, the **rtcp rd** command will output the **/dev/rdev/<hostname>/device<*n*>** path name that should be used on a subsequent **rtcp ats** attach scheduler command. Note that the **/dev/rdev/<hostname>** hostname will be the name of the local host, as it was specified in the **-H hostname_list**.

Urd - Unregister a Coupled FBS timing device

The **urd** command enables you to unregister a Coupled FBS timing device on the calling local host. Once unregistered, the device is no longer available for use on the hosts where the device was previously registered. In order to unregister a device, there may be no schedulers currently attached to the Coupled FBS timing device on any of the hosts where the device is registered. To successfully unregister a Coupled FBS timing device, you must have the P_RTIME privilege as well as enough privilege to open the device file.

The format for entering the command and a description of the corresponding argument are presented in the sections that follow.

Format

> **urd -d** *device*

Arguments

Arguments are described as follows.

-d *device* This argument specifies the path name of the device that is being unregistered as a Coupled FBS timing device. This path name should be the same path name that was originally specified on the previous corresponding '**rd -d** *device*' argument.

Vr - View a Rdevfs File Configuration

The **`vr`** command enables you to view the configuration information for a Coupled FBS timing device.

Unlike the **`vs`** command, the **`rtcp vr`** command may be used to directly obtain information about a **`/dev/rdev/<hostname>/device<`*`n`*`>`** timing device without requiring that a scheduler be currently attached to the device.

The format for entering the command and a description of the corresponding argument is presented in the sections that follow.

Format

`vr -d` *device*

Argument

This command requires one argument, which is described as follows:

`-d` *device* This argument specifies the already existing **`/dev/rdev/<hostname>/device<`*`n`*`>`** path name of the registered Coupled FBS timing device that the caller wishes to obtain information about.

Screen Display

Note that the screen display output from this command is exactly the same as the Coupled FBS timing device portion of the output from the **`vs`** command when the scheduler is attached to a Coupled FBS timing device.

If the command is successfully executed, configuration and status information similar to the following is displayed if the scheduler is attached to a Closely-Coupled timing device:

```
Closely-Coupled timing device.
Device interrupt source on host: endor
Real device name = /dev/rrtc/0c2
Registered on hosts: endor rudi cosmo orbity
Attached schedulers on hosts: rudi cosmo
SBC id where device resides: 1
SBC id mask of attached FBSs: 0x6
```

If the command is successfully executed, configuration and status information similar to the following is displayed if the scheduler is attached to a RCIM Coupled timing device:

```
RCIM Coupled timing device.
Device interrupt source on host: endor
Real device name = /dev/rrtc/2c1
Registered on hosts: endor rudi cosmo orbity
Attached schedulers on hosts: rudi cosmo
```

Descriptions of the fields presented in this display follow.

Closely-Coupled timing device

This line will be output if the scheduler is attached to a Closely-Coupled timing device.

RCIM Coupled timing device

This line will be output if the scheduler is attached to a RCIM Coupled timing device.

Device interrupt source on host

When the scheduler is attached to a Coupled FBS timing device then this field contains the hostname of the host where the timing device actually resides.

This line does not appear for Coupled timing devices that were registered with the obsolete **fbs_register_cluster_device** function or **rtcp reg** command.

Real device name

If this scheduler is attached to a Coupled FBS timing device, then this field contains the actual device filename of the timing device on the host where that device is located. If this device is a RCIM Coupled timing device, then this field will contain the actual name of the distributed interrupt device:

/dev/reti/eti0<*n*> or **/dev/rrtc/2c<*n*>**

Registered on hosts

When the scheduler is attached to a Coupled FBS timing device then this field contains a list of hostnames where the timing device is registered for use.

This line does not appear for Coupled timing devices that were registered with the obsolete **fbs_register_cluster_device** function or **rtcp reg** command.

Attached schedulers on hosts

When the scheduler is attached to a Coupled FBS timing device then this field contains a list of hostnames of the hosts that currently have schedulers attached to this timing device.

This line does not appear for Coupled timing devices that were registered with the obsolete **fbs_register_cluster_device** function or 'rtcp reg' command.

SBC id where device resides

If this scheduler is attached to a Closely-Coupled timing device, then this field contains the SBC board ID where the actual timing device resides.

SBC id mask of attached FBSs

If this scheduler is attached to a Closely-Coupled timing device, then this field contains a SBC board ID bitmask of all SBCs that currently have a frequency-based scheduler attached to this timing device.

Reg – Register a Closely-Coupled Timing Device

The **reg** command enables you to register a device on the calling SBC as a Closely-Coupled timing device. Once registered, the device is then available for use on all SBCs in the cluster. In order to register a device, you must have the P_RTIME privilege as well as enough privilege to open the device file.

The format for entering the command and a description of the corresponding argument are presented in the sections that follow.

Format

reg -d *device*

Arguments

Arguments are described as follows.

-d *device* — This argument specifies the path name of the device that is being registered as a Closely-Coupled timing device. If you are using a real-time clock or RCIM distributed device interrupt, you must enter the path name of a certain form. Refer to Chapter 3 for detailed information on the form associated with these device types. If you are using a user-suppled device, the path name must be a valid UNIX path name. Refer to Chapter 3 for an explanation of the procedures for using a user-supplied device.

When successful, the **rtcp reg** command will output the **/dev/rdev** path name that should be used on subsequent **rtcp ats** attached scheduler commands.

NOTE

The **rtcp reg** command is obsolete, and is provided only for backward compatibility with previous PowerMAX OS releases. Users are encouraged to make use of the **rd** command instead of the **reg** command. Note that the **reg** command only provides for the registration of Closely-Coupled timing devices; the **rd** command provides for the registration of both Closely-Coupled and RCIM Coupled timing devices.

Unreg – Unregister Closely-Coupled Timing Device

The **unreg** command enables you to unregister a device on the calling SBC as a Closely-Coupled timing device. Once unregistered, the device is no longer available for use on all SBCs in the cluster. In order to unregister a device, you must have the P_RTIME privilege as well as enough privilege to open the device file.

The format for entering the command and a description of the corresponding argument are presented in the sections that follow.

Format

`unreg -d` *device*

Arguments

Arguments are described as follows.

`-d` *device* — This argument specifies the path name of the device that is being unregistered as a Closely-Coupled timing device. If you are using a real-time clock or RCIM distributed device interrupt, you must enter the path name of a certain form. Refer to Chapter 3 for detailed information on the form associated with these device types. If you are using a user-suppled device, the path name must be a valid UNIX path name. Refer to Chapter 3 for an explanation of the procedures for using a user-supplied device.

NOTE

The **`rtcp unreg`** command is obsolete, and is provided only for backward compatibility with previous PowerMAX OS releases. User are encouraged to make use of the **`urd`** command instead of the **`unreg`** command.

6
The Ada Interfaces to RT Services

6
The Ada Interfaces to RT Services

The Ada interfaces to the real–time services related to the frequency–based scheduler and the performance monitor are provided as part of the MAXAda product and are located in the MAXAda environment, **/usr/ada/default/vendorlib**. The interfaces to the frequency–based scheduler and the performance monitor are defined in the RT_Interface package. Procedures for using this package are explained in "The RT_Interface Package."

The RT_Interface Package

The RT_Interface package contains subprograms that enable you to perform the entire range of functions associated with the frequency–based scheduler and the performance monitor. The frequency–based scheduler subprograms are presented in "The FBS Subprograms." The performance monitor subprograms are presented in "The Performance Monitor Subprograms."

For each subprogram in the RT_Interface package, the following information is provided:

- A description of the subprogram
- The Ada specification
- Detailed descriptions of each parameter

Procedures for compiling and linking user programs are presented in "Compiling and Linking Procedures."

The FBS Subprograms

The FBS subprograms provide access to the key features of the scheduler. They enable you to perform such basic operations as the following: (1) configure a scheduler; (2) schedule programs on it; (3) set up and connect a timing source to a scheduler; (4) start, stop, and resume scheduling on a scheduler; (5) obtain information about scheduled processes; (6) reschedule and remove scheduled processes; (7) disconnect a timing source; and (8) remove a scheduler.

In the sections that follow, all of the FBS subprograms contained in the RT_Interface package are presented in alphabetical order. Figure 6-1 illustrates the approximate order in which you might invoke the subprograms from an application program.

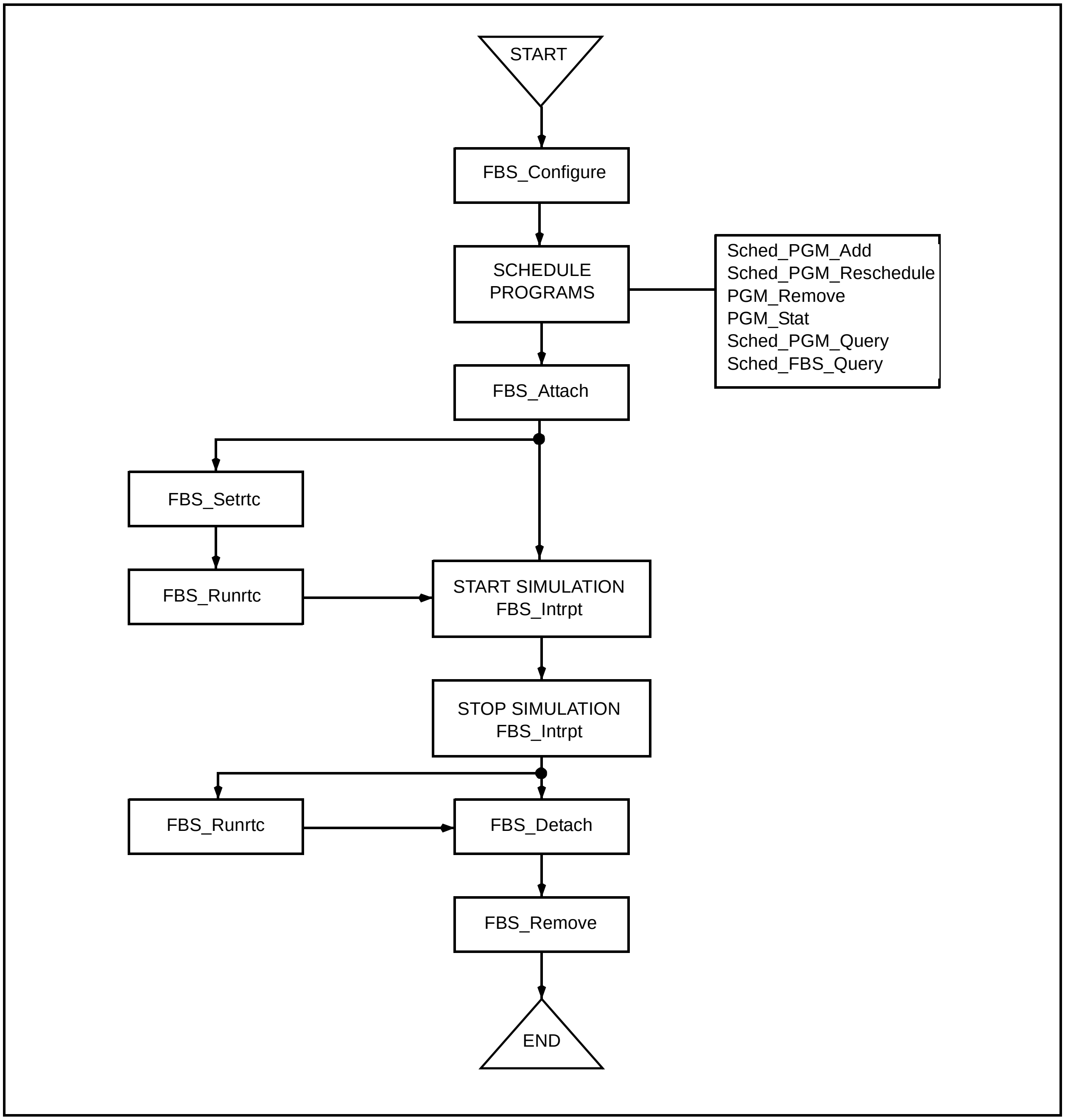

Figure 6-1. Ada Subprogram Call Sequence: FBS

FBS_Access – Change Permissions for an FBS

This subprogram is invoked to change the permissions assigned for a selected frequency–based scheduler. It is important to note that the permissions can be changed only by a process that has the P_OWNER privilege or has an effective user ID that is equal to that of the owner/creator of the frequency–based scheduler.

If the Enhanced Security Utilities are installed and running, the following conditions must be met:

- The calling process and the frequency-based scheduler must have identical security levels, or the process must have both P_MACREAD and P_MACWRITE privileges.
- The calling process must have the P_OWNER privilege or an effective user identification of owner/creator to pass the ownership restriction.

The Ada specification and corresponding parameters are presented in the following sections.

Ada Specification

```
subtype fbs_mode_type is integer;

md_public_alter  : constant fbs_mode_type := 8#002#;
md_public_read   : constant fbs_mode_type := 8#004#;
md_group_alter   : constant fbs_mode_type := 8#020#;
md_group_read    : constant fbs_mode_type := 8#040#;
md_owner_alter   : constant fbs_mode_type := 8#200#;
md_owner_read    : constant fbs_mode_type := 8#400#;
md_public        : constant fbs_mode_type := 8#666#;

procedure FBS_Access    ( scheduler   : in integer;
                          uid         : in integer;
                          gid         : in integer;
                          permissions : in fbs_mode_type;
                          istat       : out integer );
```

Parameters

Parameters are described as follows.

scheduler
: refers to a variable that contains a unique, positive integer value representing the identifier for a frequency–based scheduler. You can obtain this value by making a call to **`FBS_Configure`** (see page 6-7 for an explanation of this subprogram). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing the identifier, you can specify the value – **`1`**.

uid
: refers to a variable that contains an integer value representing the effective user ID of the specified frequency–based scheduler.

gid
: refers to a variable that contains an integer value representing the effective group ID of the specified frequency–based scheduler.

permissions — refers to a variable that contains a bit pattern that defines the permissions associated with the specified frequency–based scheduler.

Permissions are specified by using a combination of the following

md_public_alter
md_public_read
md_group_alter
md_group_read
md_owner_alter
md_owner_read
md_public

You can specify a particular permission by adding or subtracting constants; for example, (**md_public** – **md_group_alter** – **md_public_alter**) yields **644** (owner read/write, group read, others read). Additional information on setting permissions for frequency–based scheduler operations is provided in the system manual page **intro(2)**.

istat — refers to a variable to which **FBS_Access** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

– **1**,– **7** Scheduler is not configured.

– **2** Scheduler does not exist.

– **20** Operation permission is denied to the calling process (see **intro(2)**).

– **22** The effective user ID of the calling process is not equal to the value of **fbs_perm.uid** or **fbs_perm.cuid** in the data structure associated with *scheduler*, and the process does not have the P_OWNER privilege.

FBS_Attach – Attach Timing Source to an FBS

This subprogram is invoked to attach a timing source to a frequency–based scheduler or to specify end–of–cycle scheduling. The timing source can be a real–time clock, an edge–triggered interrupt device, or a user–supplied real–time device.

NOTE

Subprograms contained in the `RT_Interface` package do not provide the functionality to set up and control operation of an edge–triggered interrupt device or a user–supplied device, as they do for a real–time clock. Procedures for using a real–time clock are described in detail in Chapter 3. Procedures for using an edge–triggered interrupt and a user–supplied real–time device are also explained in that chapter.

To use a real–time clock as the timing source for a frequency–based scheduler on a PowerMAX OS system on which the Enhanced Security Utilities are installed, you must have enough privilege to open the device. Refer to the "Trusted Facility Management" chapter of *System Administration Volume 1* for an explanation of the procedures for using devices when the Enhanced Security Utilities are installed.

The Ada specification and corresponding parameters are presented in the following sections. The dynamic string data type `unbounded_string` is defined in the package `Ada.Strings.Unbounded`, which is located in the **`/usr/ada/default/predefined`** environment.

Ada Specification

```
procedure FBS_Attach     ( scheduler   : in integer;
                           CPU         : in integer;
                           devname     : in string;
                           istat       : out integer );

procedure FBS_Attach     ( scheduler   : in integer;
                           CPU         : in integer;
                           devname     : in unbounded_string;
                           istat       : out integer);
```

Parameters

Parameters are described as follows.

scheduler — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler for which the timing source is to be attached or end–of–cycle scheduling specified. You can obtain this value by making a call to **`FBS_Configure`** (see page 6-7 for an explanation of this subprogram). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

CPU — refers to a variable that must contain the value **0**.

devname — refers to a string or dynamically allocated string that contains a null string or the path name of the device that is to be used as the timing source for the specified scheduler. If **devname** contains a null string, end–of–cycle scheduling is specified; that is, execution of the processes in the next minor cycle will occur when the last process scheduled to execute in the current minor cycle finishes its execution for that cycle. If **devname** contains a path name, it may refer to a real–time clock, an edge–triggered interrupt, or a user–supplied device.

If the device is a real–time clock or an edge–triggered interrupt, the path name must be of a certain form. Refer to Chapter 3 for detailed information on the form associated with each type of device.

If the device is a user–supplied device, the path name must be a valid UNIX path name. The device must support the IOCTLVECNUM **ioctl(2)** call (see Chapter 3 for additional information).

istat — refers to a variable to which **FBS_Attach** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

- – **1,– 7** Scheduler is not configured.
- – **2** Scheduler does not exist.
- – **4** **CPU** does not equal zero.
- – **5** Device specified by *devname* does not exist or is not configured.
- – **6** Scheduler has already been attached.
- – **20** Operation permission is denied to the calling process (see **intro(2)**).
- – **23** Device specified by *devname* is already attached to another scheduler.
- – **24** Path name specified by *devname* is too long.
- – **25** Device specified by *devname* does not support the IOCTLVECNUM **ioctl** command.
- – **26** End–of–cycle scheduling cannot be enabled because the scheduler has previously been attached.

FBS_Configure – Configure an FBS

This subprogram is invoked to configure a frequency–based scheduler or to obtain configuration details for a frequency–based scheduler that has already been configured. Note that to configure a scheduler, the calling process must have the P_RTIME privilege (for additional information on privileges, refer to the *PowerMAX OS Programming Guide* and the **intro(2)** system manual page).

If you wish to configure a scheduler, you must specify a *key*, which is a user–chosen numeric identifier for a frequency–based scheduler. You must also specify a *configflg*, which is a word that sets the permission and control flag bits to characterize the scheduler.

The permissions are defined in the system manual page **intro(2)**.

The control flags are described in the header file **<sys/ipc.h>**. They include **IPC_CREAT** and **IPC_EXCL**. Setting the **IPC_CREAT** bit without setting the **IPC_EXCL** bit ensures that a new frequency–based scheduler is created if one corresponding to the value of *key* does not exist; it results in the return of the associated frequency–based scheduler identifier if one does exist and if <u>all</u> of the following conditions are met:

- The number of minor cycles specified by the *cycles* parameter matches the number of minor cycles associated with the existing scheduler
- The maximum specified by the *progs* parameter is less than or equal to the maximum number of processes per minor cycle associated with the existing scheduler
- The maximum specified by the *max* parameter is less than or equal to the maximum number of processes allowed on the existing scheduler at one time

Setting both the **IPC_CREAT** and the **IPC_EXCL** bits results in the creation of a new scheduler if one corresponding to the value of *key* does not exist; it ensures that an error is returned if one does exist.

A unique, nonnegative frequency–based scheduler identifier and corresponding data structure will be created for the specified key if the number of frequency–based schedulers already configured is less than the maximum number of schedulers allowed on your system (see Chapter 2 for a description of system tunable parameters) and if <u>one</u> of the following conditions is met:

- The value of *key* is equal to **IPC_PRIVATE** (that is, zero)
- The value of *key* is not associated with a frequency–based scheduler identifier and (*configflg* **& IPC_CREAT**) is “true”

The Ada specification and corresponding parameters are presented in the following sections.

Ada Specification

```
subtype fbs_mode_type is integer;

md_ipc_private : constant fbs_mode_type := 16#0000#;
md_ipc_creat   : constant fbs_mode_type := 16#0200#;
md_ipc_excl    : constant fbs_mode_type := 16#0400#;

type fbs_reset_flag_type is
  ( reset_with_abort, no_reset, reset );

procedure FBS_Configure  ( key        : in fbs_mode_type;
                           cycles     : in out integer;
                           progs      : in out integer;
                           max        : in out integer;
                           reset      : in out fbs_reset_flag_type;
                           configflg  : in out fbs_mode_type;
                           scheduler  : in out integer;
                           istat      : out integer );
```

Parameters

To create a frequency–based scheduler, you must specify the following parameters as described.

key
: refers to a variable that contains an integer value identifying the frequency–based scheduler that is to be created. Note that the number of schedulers that can be configured at one time cannot exceed the value of FBSMNI, which is the maximum number of frequency–based schedulers permitted on your system (see Chapter 2 for a description of system tunable parameters).

cycles
: refers to a variable that contains an integer value indicating the number of minor cycles that compose a frame on the specified scheduler.

progs
: refers to a variable that contains an integer value indicating the maximum number of programs that can be scheduled to execute during one minor cycle.

max
: refers to a variable that contains an integer value indicating the maximum number of programs that can be scheduled on the specified scheduler at one time. This value must be less than or equal to the product that is obtained by multiplying the values specified for the *cycles* and *progs* parameters.

reset
: refers to a variable that contains an enumeration value indicating whether or not processes currently scheduled on the specified scheduler are to be killed before the scheduler is reconfigured. Acceptable values and corresponding results are presented in Table 6-1.

Table 6-1. Reset Options

Value	Result
reset_with_abort	Kill and remove all processes currently scheduled on the specified scheduler
no_reset	Ignore all processes currently scheduled on the specified scheduler
reset	Remove all processes currently scheduled on the specified scheduler

configflg — refers to a variable that contains a bit pattern indicating the control flags and permissions assigned to the specified scheduler. Control flags and permissions are specified by using a combination of the following constants:

md_ipc_private
md_ipc_create
md_ipc_excl
md_public_alter
md_public_read
md_group_alter
md_group_read
md_owner_alter
md_owner_read
md_public

You can specify a particular permission by adding or subtracting constants; for example, (**md_public** – **md_group_alter** – **md_public_alter** + **md_ipc_create**) yields **1644** octal (create scheduler, owner read/write, group read, others read).

scheduler — refers to a variable to which **FBS_Configure** will return a unique, positive integer value representing the identifier for the specified frequency–based scheduler. It is important to note that this identifier is required by most of the library subprograms.

istat — refers to a variable to which **FBS_Configure** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

– **1**,– **7** Scheduler is not configured.

– **2** Scheduler does not exist.

– **3** Cannot create a new scheduler because the limit on the number of schedulers per system would be exceeded.

– **4** Cannot create a new scheduler with the specified parameters.

– **20** Operation permission is denied to the calling process (see **intro(2)**).

– **22** The calling process does not have the P_RTIME privilege.

– **28** Scheduler for *key* already exists, but **md_ipc_create** and **md_ipc_excl** were specified in *configflg*.

To obtain information for an existing frequency–based scheduler, you must specify the following parameters as described.

key
: refers to a variable that contains an integer value identifying the frequency–based scheduler for which configuration information is to be returned. If this value is zero, the frequency–based scheduler identifier associated with this scheduler must also be provided by using the *scheduler* parameter.

cycles
: refers to a variable that contains the integer value zero, indicating that current configuration information for the specified scheduler is to be returned. **FBS_Configure** will also return to this variable an integer value indicating the number of minor cycles that compose a frame on the specified scheduler.

progs
: refers to a variable to which **FBS_Configure** will return the maximum number of programs that can be scheduled to run during one minor cycle on the specified scheduler.

max
: refers to a variable to which **FBS_Configure** will return the maximum number of programs that can be scheduled on the specified scheduler at one time.

configflg
: refers to a variable to which **FBS_Configure** will return the permissions assigned to the specified scheduler.

scheduler
: refers to a variable to which **FBS_Configure** will return a unique, positive integer value representing the identifier for the specified frequency–based scheduler. If you specify a key of **0**, this variable must contain the related frequency–based scheduler identifier.

FBS_Cycle – Return Minor Cycle/Major Frame Count

This subprogram is invoked to obtain the current minor cycle and major frame count values for a frequency–based scheduler. These values enable you to determine the progress of a simulation.

The Ada specification and corresponding parameters are presented in the following sections.

Ada Specification

```
type cycle_count is record
  minor_cycle_count    : integer;
  major_frame_count    : integer;
end record;


procedure FBS_Cycle      ( scheduler     : in integer;
                           count         : out cycle_count;
                           istat         : out integer );
```

Parameters

Parameters are described as follows.

scheduler
: refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler for which you wish to obtain the current cycle and frame counts. You can obtain this value by making a call to **FBS_Configure** (see page 6-7 for an explanation of this subprogram). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

count
: refers to a record to which **FBS_Cycle** will return integer values indicating the current minor cycle and major frame for the specified scheduler. The **minor_cycle_count** component will contain the number of the cycle. The **major_frame_count** component will contain the number of the frame.

istat
: refers to a variable to which **FBS_Cycle** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

  – **1,– 7** Scheduler is not configured.

  – **2** Scheduler does not exist.

  – **20** Operation permission is denied to the calling process (see **intro(2)**).

FBS_Detach – Detach Timing Source from an FBS

This subprogram is invoked to detach the currently attached timing source from a frequency–based scheduler or to disable end–of–cycle scheduling. If the timing source is a real-time clock, it is recommended that you stop the clock prior to invoking this subprogram. You can do so by making a call to **`FBS_Runrtc`** (see page 6-25 for an explanation of this subprogram).

The Ada specification and corresponding parameters are presented in the following sections.

Ada Specification

```
procedure FBS_Detach    ( scheduler    : in integer;
                          istat        : out integer );
```

Parameters

Parameters are described as follows.

scheduler
: refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler from which you wish to detach the currently attached timing source or for which you wish to disable end–of–cycle scheduling. You can obtain this value by making a call to **`FBS_Configure`** (see page 6-7 for an explanation of this subprogram). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **`1`**.

istat
: refers to a variable to which **`FBS_Detach`** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

– **`1,`**– **`7`** Scheduler is not configured.

– **`2`** Scheduler does not exist.

– **`3`** Scheduler is not attached.

– **`20`** Operation permission is denied to the calling process (see **`intro(2)`**).

FBS_Getrtc – Obtain Current Values for Real–Time Clock

This subprogram is invoked to obtain the current count and resolution values for the real–time clock that is attached to a specified frequency–based scheduler.

The Ada specification and corresponding parameters are presented in the following sections.

Ada Specification

```
type rtc_count_type is new integer range 1.. 65_535;

type rtc_resolution_type is private;

rtc_resolution_1_microsecs         : constant rtc_resolution_type;
rtc_resolution_10_microsecs        : constant rtc_resolution_type;
rtc_resolution_100_microsecs       : constant rtc_resolution_type;
rtc_resolution_1000_microsecs      : constant rtc_resolution_type;
rtc_resolution_10000_microsecs     : constant rtc_resolution_type;

procedure FBS_Getrtc    ( scheduler    : in integer;
                          count        : out rtc_count_type;
                          resolution   : out rtc_resolution_type;
                          istat1       : out integer;
                          istat2       : out integer );
```

Parameters

Parameters are described as follows.

scheduler — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler to which the real–time clock is attached. You can obtain this value by making a call to **FBS_Configure** (see page 6-7 for an explanation of this subprogram). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

count — refers to a variable to which **FBS_Getrtc** will return an integer value indicating the current number of clock counts per minor cycle. This value can range from one to 65535.

resolution — refers to a variable to which **FBS_Getrtc** will return a constant value indicating the duration in microseconds of one clock count. This value will be one of the following:

rtc_resolution_1_microsecs
rtc_resolution_10_microsecs
rtc_resolution_100_microsecs
rtc_resolution_1000_microsecs
rtc_resolution_10000_microsecs

istat1 — refers to a variable to which **FBS_Getrtc** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

– **1**,– **7** Scheduler is not configured.

– **2** Scheduler does not exist.

– **4** Scheduler is not attached.

– **5** An error occurred on the open of the attached device; **istat2** contains the error status of the **open** call.

– **6** An error occurred on the **ioctl** call to the attached device; **istat2** contains the error status of the **ioctl** call.

– **7** Scheduler is not configured.

– **20** Operation permission is denied to the calling process (see **intro(2)**).

If **istat1** contains a value indicating that an error has occurred on an **open** or **ioctl** call, the error status of that call is returned in **istat2**.

istat2 refers to a variable to which **FBS_Getrtc** will return the error status of an **open** or **ioctl** call. See the include file <**errno.h**> for a description of the error.

FBS_Id – Return the FBS Identifier for a Key

This subprogram is invoked to obtain the frequency–based scheduler identifier associated with a particular user–specified key. The key must match the key that was specified when the scheduler was created by making a call to **FBS_Configure**.

The Ada specification and corresponding parameters are presented in the following sections.

Ada Specification

```
subtype fbs_mode_type is integer;

procedure FBS_Id      ( key          : in fbs_mode_type;
                        scheduler    : out integer;
                        istat        : out integer );
```

Parameters

Parameters are described as follows.

key refers to a variable that contains an integer value identifying a frequency–based scheduler; this value must be the same value that was specified for *key* when the scheduler was created by making a call to **FBS_Configure** (see page 6-7 for an explanation of this subprogram).

scheduler refers to a variable to which **FBS_Id** will return an integer value representing the unique frequency–based scheduler identifier associated with the key.

istat — refers to a variable to which **`FBS_Id`** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

– **1**, – **7** Scheduler is not configured.

– **2** Scheduler does not exist.

– **7** Scheduler is not configured.

– **20** Operation permission is denied to the calling process (see **`intro(2)`**).

– **22** The calling process does not have the P_RTIME privilege.

FBS_Info – Return Information for an FBS

This subprogram is invoked to obtain information that is related to a selected frequency–based scheduler but cannot be obtained by invoking other subprograms (for example, **`Sched_FBS_Query`**, **`Sched_PGM_Query`**). Such information includes the following:

- The user and group IDs of the owner and the creator of the scheduler
- The permissions assigned for the scheduler
- The key associated with the scheduler's identifier
- The total number of overruns for all processes on the scheduler
- The CPUs that are active in the system
- The CPUs on which performance monitoring has been enabled
- The FBS–enabled flag
- The path name of the device that has been attached to the scheduler

The Ada specification and corresponding parameters are presented in the following sections. The dynamic string data type `unbounded_string` is defined in the package `Ada.Strings.Unbounded`, which is located in the **`/usr/ada/default/predefined`** environment.

Ada Specification

```
subtype fbs_mode_type is integer;

md_public_alter   : constant fbs_mode_type := 8#002#;
md_public_read    : constant fbs_mode_type := 8#004#;
md_group_alter    : constant fbs_mode_type := 8#020#;
md_group_read     : constant fbs_mode_type := 8#040#;
md_owner_alter    : constant fbs_mode_type := 8#200#;
md_owner_read     : constant fbs_mode_type := 8#400#;
```

```
md_public        : constant fbs_mode_type := 8#666#;

type reserved_words_type is array( integer range 1 .. 30 ) of integer;

type fbs_info_buffer_type is record

    owner_uid              : integer;
    owner_gid              : integer;
    creator_uid            : integer;
    creator_gid            : integer;
    permissions            : fbs_mode_type;
    key                    : integer;
    flags                  : integer;
    reserved_word          : integer;
    overruns               : integer;
    cpu_active_mask        : integer;
    cpu_pm_enabled_mask    : integer;
    enabled_flag           : integer;
    reserved_words         : reserved_words_type;
    end record;

procedure FBS_Info     ( scheduler    : in integer;
                         buf          : out fbs_info_buffer_type;
                         devname      : out unbounded_string;
                         istat        : out integer );
```

Parameters

Parameters are described as follows.

scheduler — refers to a variable that contains a unique, positive integer value representing the identifier for a frequency–based scheduler. You can obtain this value by making a call to **`FBS_Configure`** (see page 6-7 for an explanation of this subprogram). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **`1`**.

buf — refers to a record to which **`FBS_Info`** will return a series of integer values that represent certain types of information about the specified scheduler. The information returned in each component of the record is presented in Table 6-2.

Table 6-2. Contents of buf Record Components

Component	Contents
buf.owner_uid	owner’s user ID
buf.owner_gid	owner’s group ID
buf.creator_uid	creator’s user ID
buf.creator_gid	creator’s group ID
buf.permissions	access modes

Table 6-2. Contents of buf Record Components (Cont.)

Component	Contents
buf.key	key
buf.flags	flags word
buf.reserved_word	reserved for future use
buf.overruns	total number of overruns for all processes on the scheduler
buf.cpu_active_mask	mask of CPUs active in the system
buf.cpu_pm_enabled_mask	mask of CPUs on which performance monitoring has been enabled
buf.enabled_flag	FBS–enabled flag
buf.reserved_words	reserved for future use

devname — refers to a dynamically allocated string to which **`FBS_Info`** will return the path name of the device that is being used as the timing source for the specified frequency–based scheduler. If end–of–cycle scheduling has been specified, *devname* will contain a null string.

istat — refers to a variable to which **`FBS_Info`** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

– **`1`**, – **`7`** Scheduler is not configured.

– **`2`** Scheduler does not exist.

– **`20`** Operation permission is denied to the calling process (see **`intro(2)`**).

FBS_Intrpt – Start/Stop/Resume Scheduling on an FBS

This subprogram is invoked to start, stop, or resume scheduling on a frequency–based scheduler. If you invoke this subprogram to start scheduling, the minor cycle, major frame, and overrun count values are reset. If you invoke it to resume scheduling, these values are not reset.

Prior to invoking **`FBS_Intrpt`**, you must have invoked **`FBS_Attach`** to specify end–of–cycle scheduling or attach a timing source to the frequency–based scheduler on which you are starting scheduling (see page 6-5 for an explanation of **`FBS_Attach`**). If you have specified a real–time clock as the timing source, scheduling will not begin until you have set and started the clock (see page 6-30 and page 6-25 for explanations of **`FBS_Setrtc`** and **`FBS_Runrtc`**, respectively). If you have specified an edge–triggered interrupt device or a user–supplied device as the timing source, it must already be generating interrupts in order for scheduling to start.

The Ada specification and corresponding parameters are presented in the following sections.

Ada Specification

```
type intr_flag_type is
  ( reset_counters_and_arm, disarm, arm );

procedure FBS_Intrpt     ( scheduler      : in integer;
                           intrpt_flag    : in intr_flag_type;
                           istat          : out integer );
```

Parameters

Parameters are described as follows.

scheduler
: refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which you wish to start, stop, or resume scheduling of processes. You can obtain this value by making a call to **`FBS_Configure`** (see page 6-7 for an explanation of this subprogram). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of –**`1`**.

intrpt_flag
: refers to a variable that contains an enumeration value indicating whether scheduling of processes on the specified scheduler is to be started, stopped, or resumed. Acceptable values and corresponding results are presented in Table 6-3.

Table 6-3. Intrpt_flag Options

Value	Result
reset_counters_and_arm	Start scheduling of processes with the initial frame, cycle, and overrun count values set to zero
disarm	Stop scheduling of processes, and save the count values for the current frame and cycle
arm	Resume scheduling of processes with the frame, cycle, and overrun count values set to the values that were saved when the scheduler was last stopped

istat — refers to a variable to which **FBS_Intrpt** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

– **1**, – **7** Scheduler is not configured.

– **2** Scheduler does not exist.

– **3** Scheduler is not attached.

– **20** Operation permission is denied to the calling process (see **intro(2)**).

FBS_Query – Query Processes on an FBS

CAUTION

This interface is obsolete. It is maintained for compatibility with CX/UX, but it returns processes' scheduling priorities without any indication of the scheduling policies with which they are associated. If you have an existing application that uses this interface, it is recommended that you change your application to use **Sched_FBS_Query** (see p. 6-49). For details on obsolete interfaces, refer to Chapter 2, "Overview of the FBS."

This subprogram is invoked to obtain information about processes that have been scheduled on a frequency–based scheduler. Information is returned for all processes scheduled on the user–specified processor(s). Information provided for each process includes the following:

- A mask of the CPU(s) on which the process can execute
- The frequency–based scheduler process identifier
- The scheduling priority
- The period (the number of minor cycles indicating the frequency with which the process is wakened in each major frame)
- The starting base cycle (the first minor cycle in which the process is scheduled to be wakened in each major frame)
- The value of the "halt on overrun" flag

The Ada specification and corresponding parameters are presented in the following sections. The dynamic string data type `unbounded_string` is defined in the package `Ada.Strings.Unbounded`, which is located in the **/usr/ada/default/predefined** environment.

Ada Specification

```
type fbs_query_buffer_element is record
   name          : unbounded_string;
   CPU           : integer;
   slot          : integer;
   priority      : integer;
   period        : integer;
   cycle         : integer;
   abort_flag    : integer;
end record;

 type fbs_query_buffer_type is array( integer range <> )
   of fbs_query_buffer_element;

 procedure FBS_Query       ( scheduler   : in integer;
                              CPU         : in integer;
                              buffer      : in out fbs_query_buffer_type;
                              istat       : out integer );
```

Parameters

Parameters are described as follows.

scheduler
: refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler for which you wish to obtain scheduling information. You can obtain this value by making a call to **`FBS_Configure`** (see page 6-7 for an explanation of this subprogram). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **`1`**.

CPU
: refers to a variable that contains an integer value indicating the processor(s) for which scheduling information is to be obtained. Acceptable values and corresponding results are presented in Table 6-4.

Table 6-4. CPU Options: FBS_Query

Value	Result
0	Scheduling information for processes executing on the processor from which the call is made is returned
-1	Scheduling information for all processes on the scheduler is returned
Bit mask	If (**cpu** & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU), scheduling information for processes executing on CPU *i* is returned

buffer	refers to an array of records to which **`FBS_Query`** will return a dynamically allocated string containing the FBS–scheduled process's name and other scheduling information. **`Buffer`** contains scheduling information for all processes scheduled on the specified CPUs bound by the declared size of the Ada buffer array. The type of information returned in each record component for a single process is presented in Table 6-5.

Table 6-5. Contents of buffer Record Components for a Process

Component for Process p	Contents
buffer(p).name	Pointer to a variable length string that contains the path name of process **p**
buffer(p).CPU	A bit mask indicating the processor(s) on which the process can execute (see Table 6-4 for a description of the bit mask)
buffer(p).slot	The process's frequency–based scheduler process identifier
buffer(p).priority	The process's scheduling priority
buffer(p).period	The number of minor cycles indicating the frequency with which the process is to be wakened in each major frame (period)
buffer(p).cycle	The first minor cycle in which the process is scheduled to be wakened in each major frame (starting base cycle)
buffer(p).abort_flag	The value of the "halt on overrun" flag. A nonzero value indicates that the flag is set. A value of zero indicates that the flag is not set.

istat — refers to a variable to which **`FBS_Query`** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

– **`1`**, – **`7`** Scheduler is not configured.

– **`2`** Scheduler does not exist.

– **`3`** More processes can be queried than can fit in **`buffer`**, but **`buffer`** has been filled to its capacity.

– **`4`** **CPU** value is invalid or out of range.

– **`20`** Operation permission is denied to the calling process (see **`intro(2)`**).

– **27** Service could not allocate enough buffers to perform the query.

FBS_Remove – Remove an FBS

This subprogram is invoked to remove a frequency–based scheduler and to free the data structure associated with it. It is important to note that prior to invoking **FBS_Remove**, you must ensure that the timing source is detached from the scheduler or that end–of–cycle scheduling is disabled (see page 6-12 for information on the use of **FBS_Detach**). It is important to note that **FBS_Remove** will remove all processes scheduled on the specified scheduler. It is recommended, however, that you remove all scheduled processes prior to invoking **FBS_Remove**. You can do so by making a call to **PGM_Remove** (see page 6-35 for information on the use of this subprogram).

Note that to remove a frequency-based scheduler, the calling process must have the P_OWNER privilege or an effective user ID that is equal to that of the owner/creator of the scheduler.

If the Enhanced Security Utilities are installed and running, the following conditions must also be met:

- The calling process and the frequency-based scheduler must have identical security levels, or the process must have both P_MACREAD and P_MACWRITE privileges.

The Ada specification and corresponding parameters are presented in the following sections.

Ada Specification

```
type fbs_remove_reset_type is  ( reset_and_abort, reset_only );

procedure FBS_Remove      ( scheduler  : in integer;
                            reset      : in fbs_remove_reset_type;
                            istat      : out integer );
```

Parameters

Parameters are described as follows.

scheduler — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler that you wish to remove. You can obtain this value by making a call to **FBS_Configure** (see page 6-7 for an explanation of this subprogram). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

reset — refers to a variable that contains an enumeration value indicating the manner in which processes scheduled on the scheduler are to be handled. Acceptable values and corresponding results are presented in Table 6-6.

Table 6-6. Reset Options

Value	Result
reset_and_abort	Kill and remove all processes currently scheduled on the specified scheduler
reset_only	Remove all processes currently scheduled on the specified scheduler

istat — refers to a variable to which **FBS_Remove** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

– **1**,– **7** Scheduler is not configured.

– **2** Scheduler does not exist.

– **6** Scheduler is still attached.

– **20** Operation permission is denied to the calling process (see **intro(2)**).

– **22** The effective user ID of the calling process is not equal to the value of **fbs_perm.uid** or **fbs_perm.cuid** in the data structure associated with s*cheduler*, and the process does not have the P_OWNER privilege.

FBS_Resume - Resume Scheduling on an FBS

The **FBS_Resume** subprogram is invoked to resume scheduling of processes on a frequency-based scheduler at the specified minor cycle, major frame, and overrun count.

Note that to resume scheduling of processes on a frequency-based scheduler, the calling process must have alter permission for the scheduler. If the Enhanced Security Utilities are installed and running, the following conditions must also be met:

- The calling process and the frequency-based scheduler must have identical security levels, or the process must have the P_MACWRITE privilege.

If you wish to resume scheduling of processes on a frequency-based scheduler without altering the scheduler's current frame, cycle, and overrun values, it is recommended that you use the **FBS_Intrpt** subprogram (see page 6-17 for an explanation of this routine).

CAUTION

> The **`FBS_Resume`** subprogram clears performance monitor values for all processes scheduled on the specified scheduler. Changing the frame and cycle count for the scheduler causes the values that are being maintained by the performance monitor to be inaccurate.

The Ada specification and corresponding parameters are presented in the following sections.

Ada Specification

```
procedure FBS_Resume    ( scheduler  : in integer;
                          frame      : in integer;
                          cycle      : in integer;
                          overruns   : in integer;
                          istat      : out integer);
```

Parameters

Parameters are described as follows.

scheduler
: refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which you wish to resume scheduling of processes. You can obtain this value by making a call to **`FBS_Configure`** (see page 6-7 for an explanation of this subprogram). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing the identifier, you can specify the value -1.

frame
: an integer value indicating the major frame in which you wish scheduling of processes to be resumed on the specified scheduler

cycle
: an integer value indicating the minor cycle in which you wish scheduling of processes to be resumed on the specified scheduler. This value can range from zero to the total number of minor cycles per frame minus one. The total number of minor cycles per frame was specified when the scheduler was created by making a call to **`FBS_Configure`** (see page 6-7 for an explanation of this subprogram).

overruns
: an integer value indicating the value to which you wish the overrun count to be set when scheduling resumes on the specified scheduler

  If you do not wish to change the overrun count, you can specify the value **`–1`**.

istat
: refers to a variable to which **`FBS_Resume`** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has

occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

– **1**,– **7** Scheduler is not configured.

– **2** Scheduler does not exist.

– **3** Scheduler is not attached.

– **4** Specified frame or cycle is out of range.

– **20** Operation permission is denied to the calling process (see **intro(2)**).

FBS_Runrtc – Start/Stop Real–Time Clock

This subprogram is invoked to start or stop the counting of a real–time clock that has been attached to a frequency–based scheduler.

The Ada specification and corresponding parameters are presented in the following sections.

Ada Specification

```
type runrtc_flag_type is ( stop_clock, start_clock );

procedure FBS_Runrtc    ( scheduler   : in integer;
                          run_flag    : in runrtc_flag_type;
                          istat1      : out integer;
                          istat2      : out integer );
```

Parameters

Parameters are described as follows.

scheduler — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler for which you wish to start or stop the attached real–time clock. You can obtain this value by making a call to **FBS_Configure** (see page 6-7 for an explanation of this subprogram). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of **–1**.

run_flag — refers to a variable that contains an enumeration value indicating whether the real–time clock is to be started or stopped. Specify **start_clock** to indicate that the clock is to be started. Specify **stop_clock** to indicate that the clock is to be stopped.

istat1 — refers to a variable to which **FBS_Runrtc** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

– **1**,– **7** Scheduler is not configured.

– **2** Scheduler does not exist.

– **4** Scheduler is not attached.

– **5** An error occurred on the open of the attached device; **istat2** contains the error status of the **open** call.

– **6** An error occurred on the **ioctl** call to the attached device; **istat2** contains the error status of the **ioctl** call.

– **20** Operation permission is denied to the calling process (see **intro(2)**).

If **istat1** contains a value indicating that an error has occurred on an **open** or **ioctl** call, the error status of that call is returned in **istat2**.

istat2 refers to a variable to which **FBS_Runrtc** will return the error status of an **open** or **ioctl** call. See the include file <**errno.h**> for a description of the error.

FBS_Sched_Self - Schedule an Ada Task on an FBS

The **FBS_Sched_Self** subprogram is invoked to schedule the calling Ada task on a frequency-based scheduler.

For purposes of discussion, a nontasking application is considered to have a single task, which is called the environment task, that executes the main subprogram. Similarly, a multitasking application also includes the environment task, which executes the main subprogram.

The **FBS_Sched_Self** subprogram may be used by a nontasking or multitasking application; however, if it is used in a multitasking application, the calling task's *weight* must be *bound*. A *bound* task is one that has a dedicated lightweight process (LWP) identified for its execution. (Refer to the "Run-Time Concepts" and "Run-Time Configuration" chapters of the *MAXAda Reference Manual* (0890516) for more information on Ada tasking concepts and configuration.)

It is important to note that **FBS_Sched_Self** does not allow the calling task to set its scheduling policy and priority or its CPU bias. These operations must be performed prior to invoking **FBS_Sched_Self**.

A bound Ada task may set its scheduling policy, priority, and CPU bias by interfacing directly with such system program interfaces as **priocntl(2)** and **mpadvise(3C)**, but use of these interfaces is not (underlined) recommended. It is recommended that you use MAXAda pragmas and options for such operations. The MAXAda pragma TASK_CPU_BIAS, for example, is used to set the CPU bias for a task. Similarly, the MAX-

Ada pragmas TASK_PRIORITY and TASK_QUANTUM are used to set the priority and scheduler class for a task. (Refer to the “Run-Time Configuration” chapter of the *MAXAda Reference Manual* (0890516) for more information.)

Note that a nontasking application may set its scheduling policy, priority, and CPU bias by using **`sched_setscheduler(3C)`** and **`cpu_bias(2)`**.

Note that you cannot use this routine to add **`/idle`** or **`/spare`** to a frequency-based scheduler.

To schedule the calling task on a frequency-based scheduler, the calling task must have alter permission for the scheduler. If the Enhanced Security Utilities are installed and running, the following conditions must also be met:

- The calling task and the frequency-based scheduler must have identical security levels, or the task must have the P_MACWRITE privilege.

You must not change the scheduling policy or priority of task while it is scheduled on a scheduler by using **`sched_setscheduler`** or other program interfaces that allow you to change scheduling policy and priority. The frequency-based scheduler is not aware of changes in scheduling policy and priority that are made by using these interfaces.

If you need to change the scheduling policy or priority of a non-tasking FBS scheduled process, you may do so by using **`Sched_Pgm_Reschedule`** to reschedule it (see page 6-62 for an explanation of this subprogram).

If you need to change the scheduling policy or priority of a bound task, you must first remove it from the scheduler on which it is has been scheduled by using **`Pgm_Remove`** (see page 6-35 for an explanation of this subprogram). You can then use services in the package `Ada.Dynamic_Priorities` to change its priority and **`FBS_Sched_Self`** to schedule it on a scheduler.

The Ada specification and corresponding parameters are presented in the following sections.

Ada Specification

```
type fbs_sched_self_buffer_type is record
     version      : integer;
     param        : integer;
     period       : integer;
     cycle        : integer;
     abort_flag   : integer;
     fpid         : integer;
end record;

procedure FBS_Sched_Self  ( scheduler  : in integer;
                            name       : in string;
                            buffer     : in out fbs_sched_self_buffer_type;
                            istat      : out integer);

procedure FBS_Sched_Self  ( scheduler  : in integer;
                            name       : in unbounded_string;
                            buffer     : in out fbs_sched_self_buffer_type;
                            istat      : out integer);
```

Parameters

Parameters are described as follows.

scheduler	refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler. You can obtain this value by making a call to **FBS_Configure** (see page 6-7 for an explanation of this subprogram). If you wish to reference the frequency–based scheduler on which the calling task is scheduled without knowing its identifier, you can specify a value of **–1**.
name	refers to a string that contains a standard UNIX path name or arbitrary content identifying the program associated with the calling task. A full or relative path name of up to 1023 characters can be specified.
buffer	refers to a record that contains the scheduling parameters with which the calling task is to be scheduled.The type of information returned in each record component for a single process is presented in Table 6-7.

Table 6-7. Contents of buffer Record Components for a Process

Component	Contents
buffer.version	an integer value indicating the version of buffer that is being passed to **FBS_Sched_Self**. The constant FBSSCHED_CUR_VERSION specifies the value to which version should be set for the structure definition presented above. Note that this value is automatically supplied via default record initialization
buffer.param	an integer value to be passed to a task that is scheduled on a frequency-based scheduler. This value can be retrieved by the FBS scheduled task through a call to **RT_Param** (see page 6-49 for an explanation of this subprogram).
buffer.period	an integer value indicating the frequency with which the calling task is to be wakened in each frame. A period of one indicates that the calling task is to be wakened every minor cycle; a period of two indicates that it is to be wakened every two minor cycles, and so on. This value can range from one to the number of minor cycles that compose a frame on the specified scheduler as defined in a call to **FBS_Configure** (see page 6-7 for an explanation of this subprogram).

Table 6-7. Contents of buffer Record Components for a Process (Cont.)

Component	Contents
buffer.cycle	an integer value that specifies the first minor cycle in which the calling task is to be wakened in each frame. This value can range from zero to the total number of minor cycles per frame minus one. The total number of minor cycles per frame is specified in a call to **FBS_Configure** (see page 6-7 for an explanation of this subprogram).
buffer.abort_flag	a flag indicating whether or not the scheduler should be stopped in the event that the calling task overruns its frame. A nonzero value indicates that the scheduler should be stopped
buffer.fpid	on successful return from **FBS_Sched_Self**, this variable contains the unique frequency-based scheduler identifier for the calling task

istat — refers to a variable to which **FBS_Sched_Self** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

– **1** Unrecognized or incompatible version was specified.

– **2** Scheduler does not exist.

– **4** Period or cycle are out of range.

– **5** Name evaluated to **/spare** or **/idle**.

– **7** Scheduler is not configured.

– **20** Operation permission is denied to the calling task (see **intro(2)**).

– **24** The length of path name *name* is too long.

FBS_Setrtc – Set Real–Time Clock

This subprogram is invoked to establish the duration of a minor cycle by setting the count and the resolution values for a real–time clock.

The Ada specification and corresponding parameters are presented in the following sections.

Ada Specification

```
type rtc_count_type is new integer range 1 .. 65_535;

type rtc_resolution_type is private;

rtc_resolution_1_microsecs       : constant rtc_resolution_type;
rtc_resolution_10_microsecs      : constant rtc_resolution_type;
rtc_resolution_100_microsecs     : constant rtc_resolution_type;
rtc_resolution_1000_microsecs    : constant rtc_resolution_type;
rtc_resolution_10000_microsecs   : constant rtc_resolution_type;

procedure FBS_Setrtc    ( scheduler    : in integer;
                          count        : in rtc_count_type;
                          resolution   : in rtc_resolution_type;
                          istat1       : out integer;
                          istat2       : out integer );
```

Parameters

Parameters are described as follows.

scheduler — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler to which a real–time clock has been attached. You can obtain this value by making a call to **FBS_Configure** (see page 6-7 for an explanation of this subprogram). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

count — refers to a variable that contains an integer value indicating the number of clock counts per minor cycle. This value can range from one to 65535.

resolution — refers to a variable that contains a constant value indicating the duration in microseconds of one clock count. This value must be one of the following:

rtc_resolution_1_microsecs
rtc_resolution_10_microsecs
rtc_resolution_100_microsecs
rtc_resolution_1000_microsecs
rtc_resolution_10000_microsecs

istat1 — refers to a variable to which **FBS_Setrtc** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

– **1,** – **7** Scheduler is not configured.

– **2** Scheduler does not exist.

– **4** Scheduler is not attached.

– **5** An error occurred on the **open** of the attached device; **istat2** contains the error status of the **ioctl** call.

– **6** An error occurred on the **ioctl** call to the attached device; **istat2** contains the error status of the **ioctl** call.

– **20** Operation permission is denied to the calling process (see **intro(2)**).

– **31** **Count** value is out of range.

If **istat1** contains a value indicating that an error has occurred on an **open** or **ioctl** call, the error status of that call is returned in **istat2**.

istat2 refers to a variable to which **FBS_Setrtc** will return the error status of an **open** or **ioctl** call. See the include file <**errno.h**> for a description of the error.

FBS_Wait – Wait on an FBS

This subprogram enables a process that is scheduled on a frequency–based scheduler to sleep until its next scheduled minor cycle.

The Ada specification and corresponding parameters are presented in the following sections.

Ada Specification

```
procedure FBS_Wait ( istat : out integer );
```

Parameter

FBS_Wait requires one parameter: **istat**. **Istat** refers to a variable to which **FBS_Wait** will return an integer value indicating whether or not an error has occurred and whether the process has been wakened by the scheduler or by an **fbstrig(2)** call from another process. Values that may be returned are described in Table 6-8.

Table 6-8. Istat Values: FBS_Wait

Value	Description
0	The process has been wakened normally
1	The process has been wakened as the result of an **fbstrig(2)** call
Other nonzero value	An error of a specific type has occurred. The nonzero values that may be returned and the types of errors that they represent are as follows: –1 Scheduler is not configured –3 Process is not scheduled on a frequency–based scheduler –4 Process has been removed from the scheduler

PGM_Query – Query a Process on an FBS

CAUTION

This interface is obsolete. It is maintained for compatibility with CX/UX, but it returns processes' scheduling priorities without any indication of the scheduling policies with which they are associated. If you have an existing application that uses this interface, it is recommended that you change your application to use **Sched_PGM_Query** (see p. 6-56). For details on obsolete interfaces, refer to Chapter 2, "Overview of the FBS."

This subprogram is invoked to obtain information for a particular process that has been scheduled on a frequency–based scheduler. You can identify the process by using one of the following methods:

- Specify the name of the process and the CPU(s) on which it is scheduled.
- Specify the process's frequency–based scheduler process identifier (slot number).
- Specify the name of the process, the CPU on which it is scheduled, and its frequency–based scheduler process identifier.

NOTE

The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency–based scheduler process identifier.

Information that is returned includes the following:

- The process's path name
- The CPU on which the process can execute
- The frequency–based scheduler process identifier
- The scheduling priority
- The period (the number of minor cycles indicating the frequency with which the process is wakened in each major frame)
- The starting base cycle (the first minor cycle in which the process is scheduled to be wakened in each major frame)
- The value of the "halt on overrun" flag

The Ada specification and corresponding parameters are presented in the following sections. The dynamic string data type `unbounded_string` is defined in the package `Ada.Strings.Unbounded`, which is located in the **`/usr/ada/default/predefined`** environment.

Ada Specification

```
procedure PGM_Query      ( scheduler  : in integer;
                           name       : in out unbounded_string;
                           CPU        : in out integer;
                           slot       : in out integer;
                           priority   : out integer;
                           period     : out integer;
                           cycle      : out integer;
                           abort_flag : out integer;
                           istat      : out integer );
```

Parameters

Parameters are described as follows.

scheduler — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process for which you wish to obtain scheduling information has been scheduled. You can obtain this value by making a call to **`FBS_Configure`** (see page 6-7 for an explanation of this subprogram). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of –1.

name — refers to a dynamically allocated string that contains a standard UNIX path name identifying the process for which information is to be returned. A full or relative path name of up to 1024 characters can be specified. If this variable contains blanks, you must provide the frequency–based scheduler process identifier in the *slot* parameter.

CPU — refers to a variable that contains an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the program for which information is to be returned. Acceptable values and corresponding results are presented in Table 6-9.

Table 6-9. CPU Options: PGM_Query

Value	Result
0	The first process named by *name* that is currently running on the processor from which the call is made is specified
-1	The first process named by *name* that is currently running on any processor is specified

Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is running on CPU *i* is specified
	
	If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is specified

slot — refers to a variable that contains an integer value providing the unique frequency–based scheduler process identifier for the process for which information is to be returned. This value is obtained when you make a call to **`PGM_Schedule`** (see page 6-42 for an explanation of this subprogram). This value must be − **`1`** if you wish to identify the program to be queried only by specifying *name* and *cpu*.

priority — refers to a variable to which **`PGM_Query`** will return an integer value indicating the specified process's scheduling priority.

period — refers to a variable to which **`PGM_Query`** will return an integer value indicating the frequency with which the specified program is to be wakened in each major frame. A period of one indicates that the specified program is to be wakened every minor cycle; a period of two indicates that it is to be wakened once every two minor cycles, a period of three once every three minor cycles, and so on.

cycle — refers to a variable to which **`PGM_Query`** will return an integer value indicating the first minor cycle in which the specified process is scheduled to be wakened in each frame

abort_flag — refers to a variable to which **`PGM_Query`** will return an integer value indicating the value of the "halt on overrun" flag. A nonzero value indicates that the flag is set. A value of zero indicates that the flag is not set.

istat — refers to a variable to which **`PGM_Query`** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

– **`1`**,– **`7`** Scheduler is not configured.

– **`2`** Scheduler does not exist.

– **`3`** Process is not scheduled on this scheduler.

– **`4`** **CPU** value is out of range.

– **`20`** Operation permission is denied to the calling process (see **`intro(2)`**).

– **`24`** Path name specified by *name* is too long.

PGM_Remove – Remove a Process from an FBS

This subprogram is invoked to remove a process from a frequency–based scheduler. You can identify the process that you wish to remove by using one of the following methods:

- Specify the name of the process and the CPU on which it is scheduled.
- Specify the process's frequency–based scheduler process identifier (slot number).
- Specify the name of the process, the CPU on which it is scheduled, and its frequency–based scheduler process identifier.

NOTE

The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency–based scheduler process identifier.

The Ada specification and corresponding parameters are presented in the following sections. The dynamic string data type `unbounded_string` is defined in the package `Ada.Strings.Unbounded`, which is located in the **`/usr/ada/default/predefined`** environment.

Ada Specification

```
procedure PGM_Remove    ( scheduler : in integer;
                          name      : in string;
                          CPU       : in integer;
                          slot      : in integer;
                          abrt      : in integer;
                          istat     : out integer );

procedure PGM_Remove    ( scheduler : in integer;
                          name      : in unbounded_string;
                          CPU       : in integer;
                          slot      : in integer;
                          abrt      : in integer;
                          istat     : out integer );
```

Parameters

Parameters are described as follows.

scheduler
: refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process is scheduled. You can obtain this value by making a call to **`FBS_Configure`** (see page 6-7 for an explanation of this subprogram). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **`1`**.

name
: refers to a string or dynamically allocated string that contains a standard UNIX path name identifying the process to be removed from the specified scheduler. A full or relative path name of up to 1024 characters can be specified. If this variable contains blanks, you must provide the frequency–based scheduler process identifier in the *slot* parameter.

CPU
: refers to a variable that contains an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the process to be removed from the specified scheduler. Acceptable values and corresponding results are presented in Table 6-10.

Table 6-10. CPU Options: PGM_Remove

Value	Result
0	The first process named by *name* that is currently running on the processor from which the call is made is removed
-1	The first process named by *name* that is currently running on any processor is removed

Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is running on CPU *i* is removed
	
	If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is removed

slot — refers to a variable that contains an integer value providing the unique frequency–based scheduler process identifier for the process to be removed from the specified scheduler. This value is obtained when you make a call to **`Sched_PGM_Add`** (see page 6-52 for an explanation of this subprogram). This value must be – **`1`** if you choose to identify the program to be removed only by specifying *name* and *cpu*.

abrt — refers to a flag that contains an integer value indicating the manner in which the specified process is be removed from the specified scheduler. A positive value indicates that the process is to be removed from the scheduler but allowed to continue executing. A negative value indicates that the process is to be removed from the scheduler and terminated.

istat — refers to a variable to which **`PGM_Remove`** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

– **`1`**,– **`7`** Scheduler is not configured.

– **`2`** Scheduler does not exist.

– **`3`** Process is not scheduled on the specified scheduler.

– **`4`** **CPU** value is out of range.

– **`20`** Operation permission is denied to the calling process (see **`intro(2)`**).

PGM_Reschedule – Reschedule a Process

CAUTION

This interface is obsolete. It is maintained for compatibility with CX/UX, but its behavior with respect to specification of a process's scheduling priority has changed. If you have an existing application that uses this interface, it is recommended that you change your application to use **Sched_PGM_Reschedule** (see p. 6-59). For details on obsolete interfaces, refer to Chapter 2, "Overview of the FBS."

This subprogram is invoked to change the scheduling parameters for a process that is scheduled on a frequency–based scheduler. You may wish, for example, to change a program's priority or the frequency with which it is scheduled to run. You cannot, however, change the CPU on which it has been scheduled.

To change a process's priority, the following conditions must be met:

- The calling process must have the P_RTIME privilege.
- The effective user ID of the calling process must match the effective user ID of the target process (the process for which the scheduling policy and priority are being set), or the calling process must have the P_OWNER privilege.

If the Enhanced Security Utilities are installed and running, the following additional conditions must be met:

- The Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

For additional information on privileges, refer to the *PowerMAX OS Programming Guide* and the **intro(2)** system manual page.

You can call **PGM_Reschedule** to change the parameters without having called **PGM_Remove** to remove the process from the scheduler (see page 6-35) or **FBS_Intrpt** to stop the simulation (see page 6-17).

You can identify the process that you wish to reschedule by using one of the following methods:

- Specify the name of the process and the CPU on which it is scheduled.
- Specify the process's frequency–based scheduler process identifier (slot number).
- Specify the name of the process, the CPU on which it is scheduled, and its frequency–based scheduler process identifier.

NOTE

> The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency–based scheduler process identifier.

The Ada specification and corresponding parameters are presented in the following sections. The dynamic string data type `unbounded_string` is defined in the package `Ada.Strings.Unbounded`, which is located in the **`/usr/ada/default/predefined`** environment.

Ada Specification

```
procedure PGM_Reschedule   ( scheduler      : in integer;
                             name           : in string;
                             CPU            : in integer;
                             slot           : in integer;
                             priority       : in integer;
                             param          : in integer;
                             period         : in integer;
                             cycle          : in integer;
                             abrt           : in integer;
                             istat          : out integer );

procedure PGM_Reschedule   ( scheduler      : in integer;
                             name           : in unbounded_string;
                             CPU            : in integer;
                             slot           : in integer;
                             priority       : in integer;
                             param          : in integer;
                             period         : in integer;
                             cycle          : in integer;
                             abrt           : in integer;
                             istat          : out integer );
```

Parameters

Parameters are described as follows.

scheduler
: refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process is scheduled. You can obtain this value by making a call to **`FBS_Configure`** (see page 6-7 for an explanation of this subprogram). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **`1`**.

name
: refers to a string or dynamically allocated string that contains a standard UNIX path name identifying the process to be rescheduled. A full or relative path name of up to 1024 characters can be specified. If this variable contains blanks, you must provide the frequency–based scheduler process identifier in the *slot* parameter.

CPU — refers to a variable that contains an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the process to be rescheduled. Acceptable values and corresponding results are presented in Table 6-11.

Table 6-11. CPU Options: PGM_Reschedule

Value	Result
0	The first process named by *name* that is currently running on the processor from which the call is made is rescheduled
-1	The first process named by *name* that is currently running on any processor is rescheduled

Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is running on CPU *i* is rescheduled
	
	If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is rescheduled

slot — refers to a variable that contains an integer value providing the unique frequency–based scheduler process identifier for the process to be rescheduled. This value is obtained when you make a call to **`PGM_Schedule`** (see page 6-42 for an explanation of this subprogram). This value must be – **`1`** if you wish to identify the program to be rescheduled only by specifying *name* and *cpu*.

priority — an integer value indicating the specified process's scheduling priority. A process that has been scheduled using **`PGM_Schedule`** (see p. 6-42 for an explanation of this subprogram) is scheduled under the POSIX **SCHED_RR** scheduling policy. The value specified must lie in the range of priorities associated with this policy. You can obtain the allowable range of priorities by invoking the **`run(1)`** command from the shell and not specifying any options or arguments (see the corresponding system manual page for an explanation of this command). Higher numerical values correspond to more favorable scheduling priorities.

For complete information on scheduling policies and priorities, refer to the "Process Scheduling and Management" chapter of the *PowerMAX OS Programming Guide*.

param — refers to a variable that contains an integer value to be passed to a process that is scheduled on a frequency–based scheduler.

period — refers to a variable that contains an integer value indicating the frequency with which the specified program is to be wakened in each major frame. A period of one indicates that the specified program is to be wakened every minor cycle; a period of two indicates that it is to be wakened once every two minor cycles, a period of three once every three minor cycles, and so on. This value can range from one to the number of minor cycles that compose a frame on the specified scheduler as defined in a call to **`FBS_Configure`** (see page 6-7).

cycle — refers to a variable that contains an integer value indicating the first minor cycle in which the specified process is scheduled to be wakened in each frame. This value can range from zero to the total number of minor cycles per frame minus one. The total number of minor cycles per frame is specified in a call to **`FBS_Configure`** (see page 6-7 for an explanation of this subprogram).

abrt — refers to a flag that contains an integer value indicating whether or not the scheduler should be stopped in the event that the specified process causes a frame overrun. A nonzero value indicates that the scheduler will be stopped.

istat — refers to a variable to which **`PGM_Reschedule`** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

– **`1`**, – **`7`** Scheduler is not configured.

– **`2`** Scheduler does not exist.

– **`3`** Process is not scheduled on the specified scheduler.

– **`4`** *CPU*, *period* or *cycle* value is out of range.

– **`20`** Operation permission is denied to the calling process (see **`intro(2)`**).

– **`29`** There is no space left to perform the reschedule.

– **`33`** The **`sched_setschedular(3C)`** call failed for the scheduled process when attempting to set the scheduling class or priority.

PGM_Schedule – Schedule a Process on an FBS

CAUTION

This interface is obsolete. It is maintained for compatibility with CX/UX, but it returns processes' scheduling priorities without any indication of the scheduling policies with which they are associated. If you have an existing application that uses this interface, it is recommended that you change your application to use **`Sched_PGM_Add`** (see p. 6-52). For details on obsolete interfaces, refer to Chapter 2, "Overview of the FBS."

This subprogram is invoked to create a new process and schedule it on a frequency–based scheduler. When a process is scheduled using this subprogram, it is scheduled under the POSIX **SCHED_RR** scheduling policy (for complete information on scheduling policies and priorities, refer to the "Process Scheduling and Management" chapter of the *PowerMAX OS Programming Guide*). Note that a process can not be scheduled under this policy on a CPU on which servicing of the 60 Hz clock interrupt has been disabled. In such cases, the process will behave as though it were scheduled under the **SCHED_FIFO** policy.

If you wish to set the process's scheduling priority, the following conditions must be met:

- The calling process must have the P_RTIME privilege.
- The effective user ID of the calling process must match the effective user ID of the target process (the process for which the scheduling policy and priority are being set), or the calling process must have the P_OWNER privilege.

If the Enhanced Security Utilities are installed and running, the following additional conditions must be met:

- The Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

If you wish to modify the process's CPU bias when you invoke this subprogram, the following conditions must be met:

- The real or effective user ID of the calling process must match the real or saved user ID of the process for which the CPU assignment is being changed, or the calling process must have the P_OWNER privilege.
- To add a CPU to a process's CPU bias, the calling process must have the P_CPUBIAS privilege.

If the Enhanced Security Utilities are installed and running, the following additional conditions must be met:

- The Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

The Ada specification and corresponding parameters are presented in the following sections. The dynamic string data type `unbounded_string` is defined in the package `Ada.Strings.Unbounded`, which is located in the **`/usr/ada/default/predefined`** environment.

Ada Specification

```
procedure PGM_Schedule    ( scheduler  : in integer;
                            name       : in string;
                            priority   : in integer;
                            param      : in integer;
                            period     : in integer;
                            cycle      : in integer;
                            abrt       : in integer;
                            CPU        : in integer;
                            slot       : out integer;
                            istat      : out integer );
```

```
procedure PGM_Schedule    ( scheduler  : in integer;
                            name       : in unbounded_string;
                            priority   : in integer;
                            param      : in integer;
                            period     : in integer;
                            cycle      : in integer;
                            abrt       : in integer;
                            CPU        : in integer;
                            slot       : out integer;
                            istat      : out integer );
```

Parameters

Parameters are described as follows.

scheduler — refers to a variable that contains a unique, positive integer value representing the identifier for a frequency–based scheduler. You can obtain this value by making a call to **`FBS_Configure`** (see page 6-7 for an explanation of this subprogram). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing the identifier, you can specify the value – **`1`**.

name — refers to a string or dynamically allocated string that contains a standard UNIX path name identifying the program to be scheduled on the scheduler. A full or relative path name of up to 1024 characters can be specified.

priority — an integer value indicating the specified process's scheduling priority. A process that is scheduled using **`PGM_Schedule`** is scheduled under the POSIX **SCHED_RR** scheduling policy. The value specified must lie in the range of priorities associated with this policy. You can obtain the allowable range of priorities by invoking the **`run(1)`** command from the shell and not specifying any options or arguments (see the corresponding system manual page for an explanation of this command). Higher numerical values correspond to more favorable scheduling priorities.

For complete information on scheduling policies and priorities, refer to the "Process Scheduling and Management" chapter of the *PowerMAX OS Programming Guide*.

param — refers to a variable that contains an integer value to be passed to a process that is scheduled on a frequency–based scheduler. This value can be retrieved by the FBS–scheduled process through a call to **`RT_Param`** (see page 6-49 for an explanation of this subprogram).

period — refers to a variable that contains an integer value indicating the frequency with which the specified program is to be wakened in each major frame. A period of one indicates that the specified program is to be wakened every minor cycle; a period of two indicates that it is to be wakened once every two minor cycles, a period of three once every three minor cycles, and so on. This value can range from one to the number of minor cycles that compose a frame on the specified scheduler as defined in a call to **`FBS_Configure`** (see page 6-7).

cycle — refers to a variable that contains an integer value indicating the first minor cycle in which the specified program is scheduled to be wakened in each frame. This value can range from zero to the total number of minor cycles per frame minus one. (The total number of minor cycles per frame is specified in a call to **`FBS_Configure`**. See page 6-7 for an explanation of this subprogram).

abrt — refers to a flag that contains an integer value indicating whether or not the scheduler should be stopped in the event that the specified program causes a frame overrun. A non-zero value indicates that the scheduler will be stopped.

CPU — refers to a mask that identifies the processors on which the specified program can be scheduled to run. Acceptable values and corresponding results are presented in Table 6-12.

Table 6-12. CPU Options: PGM_Schedule

Value	Result
0	The program specified by *name* can be scheduled on the processor from which the call is made
-1	The program specified by *name* can be scheduled on any processor
Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) the program specified by *name* can be scheduled on CPU *i*

slot — refers to a variable to which **PGM_Schedule** will return an integer value that is the unique frequency–based scheduler process identifier for the scheduled process.

istat — refers to a variable to which **PGM_Schedule** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

- **– 1,– 7** Scheduler is not configured.
- **– 2** Scheduler does not exist.
- **– 3** *CPU*, *period* or *cycle* value is out of range.
- **– 5** Process specified by *name* does not exist.
- **– 20** Operation permission is denied to the calling process (see **intro(2)**).
- **– 24** Path name specified by *name* is too long.
- **– 28** The **/idle** or **/spare** process is already scheduled.
- **– 29** There is no space left to perform the scheduling.
- **– 32** Process was killed or stopped by a signal.
- **– 33** The **sched_setscheduler(3C)** call failed for the scheduled process when attempting to set the scheduling class or priority.
- **– 34** The **fork(2)** of the scheduled process failed.

PGM_Stat – Query State of FBS–Scheduled Process

This subprogram is invoked to obtain information about the state of a particular process that has been scheduled on a frequency–based scheduler. The state of the process indicates whether it is in the **`FBS_Wait`** sleep state or is in another state.

You can identify the process by using one of the following methods:

- Specify the name of the process and the CPU(s) on which it is scheduled.
- Specify the process's frequency–based scheduler process identifier (slot number).
- Specify the name of the process, the CPU on which it is scheduled, and its frequency–based scheduler process identifier.

NOTE

> The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency–based scheduler process identifier.

Information that is returned includes the following:

- The process's path name
- A mask of the CPU(s) on which the process can run
- The frequency–based scheduler process identifier
- The current state of the process

The Ada specification and corresponding parameters are presented in the following sections. The dynamic string data type `unbounded_string` is defined in the package `Ada.Strings.Unbounded`, which is located in the **`/usr/ada/default/predefined`** environment.

Ada Specification

```
procedure PGM_Stat      ( scheduler     : in integer;
                          name          : in out unbounded_string;
                          CPU           : in out integer;
                          slot          : in out integer;
                          state         : out integer;
                          istat         : out integer );
```

Parameters

Parameters are described as follows.

scheduler — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process for which you wish to obtain state information has been scheduled. You can obtain this value by making a call to **`FBS_Configure`** (see page

6-7 for an explanation of this subprogram). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

name refers to a dynamically allocated string that contains a standard UNIX path name identifying the process for which state information is to be returned. A full or relative path name of up to 1024 characters can be specified. If this variable contains blanks, you must provide the frequency–based scheduler process identifier in the *slot* parameter. **PGM_Stat** will return to this variable the path name of the specified FBS–scheduled process.

CPU refers to a variable that contains an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the program for which state information is to be returned. Acceptable values and corresponding results are presented in Table 6-13.

Table 6-13. CPU Options: PGM_Stat

Value	Result
0	The first process named by *name* that is currently running on the processor from which the call is made is specified
-1	The first process named by *name* that is currently running on any processor is specified

Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is running on CPU *i* is specified
	
	If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is specified

PGM_Stat will return to this variable the mask of the CPUs on which the specified process can run.

slot refers to a variable that contains an integer value providing the unique frequency–based scheduler process identifier for the process for which the state is to be returned. This value is obtained when you make a call to **PGM_Schedule** (see page 6-42 for an explanation of this subprogram). This value must be – **1** if you wish to identify the program to be queried only by specifying *name* and *cpu*. **PGM_Stat** will return to this variable the frequency–based scheduler process identifier for the specified process.

state — refers to a variable to which **PGM_Stat** will return an integer value indicating the current state of the specified process as defined in **fbslib.h**.

istat — refers to a variable to which **PGM_Stat** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

– **1**, – **7** Scheduler is not configured.

– **2** Scheduler does not exist.

– **3** Process is not scheduled on this scheduler.

– **4** **CPU** value is out of range.

– **20** Operation permission is denied to the calling process (see **intro(2)**).

– **24** Path name specified by *name* is too long.

PGM_Trigger – Trigger Process Waiting on FBS

This subprogram enables a process to wake a process that is in the **FBS_Wait** sleep state. It is important to note that the calling process does not have to be scheduled on a frequency–based scheduler; the target process must be.

The Ada specification and corresponding parameters are presented in the following sections.

Ada Specification

```
type trigger_flag is  ( no_context_switch, trigger_context_switch );

procedure PGM_Trigger     ( scheduler   : in integer;
                            slot        : in integer;
                            tgrflag     : in trigger_flag;
                            istat       : out integer );
```

Parameters

Parameters are described as follows.

scheduler — refers to a variable that contains a unique, positive integer value representing the identifier for a frequency–based scheduler on which the sleeping process is scheduled.

slot — refers to a variable that contains an integer value providing the unique frequency–based scheduler process identifier for the sleeping process. This value is obtained when you make a call to **Sched_PGM_Add** (see page 6-52 for an explanation of this subprogram).

tgrflg — refers to a variable that contains an enumeration value indicating whether or not a context switch is to be forced on the processor on which the wakened process is executing. If you wish to force a context switch, specify **trigger_context_switch**; otherwise, specify **no_context_switch**.

istat — refers to a variable to which **PGM_Trigger** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that the process is runnable. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

– **1**, – **7** Scheduler is not configured

– **4** Process is not scheduled.

– **5** Process is already running.

RT_Param – Return Initiation Parameter

This subprogram enables a process that is scheduled on a frequency–based scheduler to obtain the value of a process initiation parameter that has been passed to it via a call to **Sched_PGM_Add** (see page 6-52) or **Sched_PGM_Reschedule** (see page 6-59).

The Ada specification and corresponding parameters are presented in the following sections.

Ada Specification

```
procedure RT_Param      ( param    : out integer );
```

Parameter

RT_Param requires one parameter: *param*. *Param* refers to a variable to which **RT_Param** will return the integer value passed to the process via a call to **Sched_PGM_Add** or **Sched_PGM_Reschedule**. If the call is not successful, a value of zero will be returned.

Sched_FBS_Query

This subprogram is invoked to obtain information about processes that have been scheduled on a frequency-based scheduler. Information is returned for all processes scheduled on the user-specified processor(s). Information provided for each process includes the following:

- A mask of the CPU(s) on which the process can execute
- The frequency–based scheduler process identifier
- The scheduling policy under which the process has been scheduled
- The scheduling priority

- The period (the number of minor cycles indicating the frequency with which the process is wakened in each major frame)
- The starting base cycle (the first minor cycle in which the process is scheduled to be wakened in each major frame)
- The value of the “halt on overrun” flag

The Ada specification and corresponding parameters are presented in the following sections.

Ada Specification

```
type s_fbs_query_buffer_element is record
        name         : unbounded_string;
        CPU          : integer;
        slot         : integer;
        policy       : integer;
        priority     : integer;
        period:        integer;
        cycle        : integer;
        abort_flag   : integer;
end record;

type s_fbs_query_buffer_type is array ( integer range <> )
        of s_fbs_query_buffer_element;

procedure Sched_FBS_Query(scheduler: in integer;
                          CPU  :     in integer;
                          buffer:    in out s_fbs_query_buffer_type;
                          istat  :   out integer  );
```

Parameters

Parameters are described as follows.

scheduler
: refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process for which you wish to obtain scheduling information. You can obtain this value by making a call to **`FBS_Configure`** (see page 6-7 for an explanation of this subprogram). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of **`–1`**

CPU
: refers to a variable that contains an integer value indicating the processor(s) for which scheduling information is to be obtained. Acceptable values and corresponding results are presented in Table 6-14.

Table 6-14. CPU Options: Sched_FBS_Query

Value	Result
0	Scheduling information for processes executing on the processor from which the call is made is returned
-1	Scheduling information for all processes on the scheduler is returned
Bit mask	If (*cpu* & (1<<***i***)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU), scheduling information for processes executing on CPU ***i*** is returned

buffer — refers to an array of records to which **`Sched_FBS_Query`** will return a dynamically allocated string containing the FBS–scheduled process's name and other scheduling information. **`Buffer`** contains scheduling information for all processes scheduled on the specified CPUs bound by the declared size of the Ada buffer array. The type of information returned in each record component for a single process is presented in Table 6-15.

Table 6-15. Contents of buffer Record Components for a Process

Component for Process p	Contents
buffer(p).name	Pointer to a variable length string that contains the path name of process **p**
buffer(p).CPU	A bit mask indicating the processor(s) on which the process can execute (see Table 6-14 for a description of the bit mask)
buffer(p).slot	The process's frequency–based scheduler process identifier
buffer(p).policy	The process's scheduling policy
buffer(p).priority	The process's scheduling priority
buffer(p).period	The number of minor cycles indicating the frequency with which the process is to be wakened in each major frame (period)
buffer(p).cycle	The first minor cycle in which the process is scheduled to be wakened in each major frame (starting base cycle)
buffer(p).abort_flag	The value of the "halt on overrun" flag. A nonzero value indicates that the flag is set. A value of zero indicates that the flag is not set.

istat — refers to a variable to which **Sched_FBS_Query** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

– **1**, – **7** Scheduler is not configured.

– **2** Scheduler does not exist.

– **3** More processes can be queried than can fit in **buffer**, but **buffer** has been filled to its capacity.

– **4** *CPU* value is invalid or out of range.

– **20** Operation permission is denied to the calling process (see **intro(2)**).

– **27** Service could not allocate enough buffers to perform the query.

Sched_PGM_Add

The **Sched_PGM_Add** subprogram is invoked to create a new process and schedule it on a frequency–based scheduler. If you execute this command and you wish to (1) change a process's scheduling policy to the **SCHED_FIFO** or the **SCHED_RR** policy or (2) change the priority of a process scheduled under the **SCHED_FIFO** or the **SCHED_RR** policy, the following conditions must be met:

- The calling process must have the P_RTIME privilege.
- The effective user ID of the calling process must match the effective user ID of the target process (the process for which the scheduling policy and priority are being set), or the calling process must have the P_OWNER privilege.

If the Enhanced Security Utilities are installed and running, the following additional conditions must be met:

- The Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

If you wish to raise the priority of a process scheduled under the **SCHED_OTHER** policy above a per-process or LWP limit, the following conditions must be met:

- The calling process must have the P_TSHAR privilege.
- The effective user ID of the calling process must match the effective user ID of the target process (the process for which the scheduling priority is being set), or the calling process must have the P_OWNER privilege.

If the Enhanced Security Utilities are installed and running, the following additional conditions must be met:

- The Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

If you wish to modify the process's CPU bias when you invoke this command, the following conditions must be met:

- The real or effective user ID of the calling process must match the real or saved user ID of the process for which the CPU assignment is being changed, or the calling process must have the P_OWNER privilege.
- To add a CPU to a process's CPU bias, the calling process must have the P_CPUBIAS privilege.

If the Enhanced Security Utilities are installed and running, the following additional conditions must be met:

- The Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

For additional information on privileges, refer to the *PowerMAX OS Programming Guide* and the **intro(2)** system manual page.

The Ada specification and corresponding parameters are presented in the following sections.

Ada Specification

```
type pgm_sched_policy is (SCHED_INVALID,
                          SCHED_OTHER)
                          SCHED_FIFO,
                          SCHED_RR);
procedure Sched_PGM_Add  (scheduler: in  integer;
                          name      : in  string;
                          cid       : in  pgm_sched_policy;
                          priority  : in  integer;
                          param     : in  integer;
                          period    : in  integer;
                          cycle     : in  integer;
                          abrt      : in  integer;
                          CPU       : in   integer;
                          slot      : in out integer;
                          istat     : out  integer  );

procedure Sched_PGM_Add(scheduler: in  integer;
                          name      : in unbounded_string;
                          cid       : in  pgm_sched_policy;
                          priority  : in  integer;
                          param     : in  integer;
                          period    : in  integer;
                          cycle     : in  integer;
                          abrt      : in  integer;
                          CPU       : in  integer;
```

```
slot      : in out integer;
istat     : in  integer  );
```

Parameters

Parameters are described as follows.

scheduler
: refers to a variable that contains a unique, positive integer value representing the identifier for a frequency–based scheduler. You can obtain this value by making a call to **`FBS_Configure`** (see 6-7 for an explanation of this subprogram). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing the identifier, you can specify the value **`1.`**

name
: refers to a string or dynamically allocated string that contains a standard UNIX path name identifying the program to be scheduled on the scheduler. A full or relative path name of up to 1024 characters can be specified.

cid
: refers to a variable that contains an enumeration value indicating the POSIX scheduling policy under which the specified program is to be scheduled. Acceptable values are presented as follows (**SCHED_INVALID** may not be supplied here):

SCHED_OTHER
time-sharing scheduling policy

SCHED_FIFO
first–in–first–out (FIFO) scheduling policy

SCHED_RR
round–robin (RR) scheduling policy. Note that a process cannot be scheduled under this policy on a CPU on which servicing of the 60 Hz clock interrupt has been disabled. In such cases, the process will behave as though it were scheduled under the **SCHED_FIFO** policy.

priority
: refers to a variable that contains an integer value indicating the scheduling priority of the specified program. The range of acceptable priority values is governed by the scheduling policy specified.

You can determine the allowable range of priorities associated with each policy (**SCHED_FIFO**, **SCHED_RR**, or **SCHED_OTHER**) by invoking the **`run(1)`** command from the shell and not specifying any options or arguments (see the corresponding system manual page for an explanation of this command). Higher numerical values correspond to more favorable priorities.

For complete information on scheduling policies and priorities, refer to the “Process Scheduling and Management” chapter of the *PowerMAX OS Programming Guide*.

param — refers to a variable that contains an integer value to be passed to a process that is scheduled on a frequency–based scheduler. This value can be retrieved by the FBS–scheduled process through a call to **`RT_Param`** (see page 6-49 for an explanation of this subprogram).

period — refers to a variable that contains an integer value indicating the frequency with which the specified program is to be wakened in each major frame. A period of one indicates that the specified program is to be wakened every minor cycle; a period of two indicates that it is to be wakened once every two minor cycles, a period of three once every three minor cycles, and so on. This value can range from one to the number of minor cycles that compose a frame on the specified scheduler as defined in a call to **`FBS_Configure`** (see page 6-7).

cycle — refers to a variable that contains an integer value indicating the first minor cycle in which the specified program is scheduled to be wakened in each frame. This value can range from zero to the total number of minor cycles per frame minus one. (The total number of minor cycles per frame is specified in a call to **`FBS_Configure`**. See page 6-7 for an explanation of this subprogram).

abrt — refers to a flag that contains an integer value indicating whether or not the scheduler should be stopped in the event that the specified program causes a frame overrun. A non-zero value indicates that the scheduler will be stopped.

CPU — refers to a mask that identifies the processors on which the specified program can be scheduled to run. Acceptable values and corresponding results are presented in Table 6-16.

Table 6-16. CPU Options: Sched_PGM_Add

Value	Result
0	The program specified by *name* can be scheduled on the processor from which the call is made
-1	The program specified by *name* can be scheduled on any processor
Bit mask	If (*cpu* & (1<<***`i`***)) is set (where ***`i`*** is an integer ranging from zero to 15 and representing a CPU) the program specified by *name* can be scheduled on CPU ***`i`***

slot — refers to a variable to which **Sched_PGM_Add** will return an integer value that is the unique frequency–based scheduler process identifier for the scheduled process.

istat — refers to a variable to which **Sched_PGM_Add** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

– **1**, – **7** Scheduler is not configured.

– **2** Scheduler does not exist.

– **3** *CPU***,** *period* or *cycle* value is out of range.

– **5** Process specified by *name* does not exist.

– **20** Operation permission is denied to the calling process (see **intro(2)**).

– **24** Path name specified by *name* is too long.

– **28** The **/idle** or **/spare** process is already scheduled.

– **29** There is no space left to perform the scheduling.

– **32** Process was killed or stopped by a signal.

– **33** The **sched_setscheduler(3C)** call failed for the scheduled process when attempting to set the scheduling class or priority.

– **34** The **fork(2)** of the scheduled process failed.

Sched_PGM_Query

The **Sched_PGM_Query** subprogram is invoked to obtain information for a particular process that has been scheduled on a frequency–based scheduler. You can identify the process by using one of the following methods:

- Specify the name of the process and the CPU(s) on which it is scheduled.
- Specify the process's frequency–based scheduler process identifier (slot number).
- Specify the name of the process, the CPU on which it is scheduled, and its frequency–based scheduler process identifier.

NOTE

> The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency–based scheduler process identifier.

Information that is returned includes the following:

- The process's path name
- The CPU on which the process can execute
- The frequency–based scheduler process identifier
- The scheduling policy
- The scheduling priority
- The period (the number of minor cycles indicating the frequency with which the process is wakened in each major frame)
- The starting base cycle (the first minor cycle in which the process is scheduled to be wakened in each major frame)
- The value of the "halt on overrun" flag

The Ada specification and corresponding parameters are presented in the following sections.

Ada Specification

```
type pgm_sched_policy is (SCHED_INVALID,
                          SCHED_OTHER)
                          SCHED_FIFO,
                          SCHED_RR);

procedure Sched_PGM_Query (scheduler:  in integer;
                          name       :  in out unbounded_string;
                          CPU        :  in out integer;
                          slot       :  in out  integer;
                          cid        :  out  pgm_sched_policy;
                          priority   :  out integer;
                          period     :  out integer;
                          cycle      :  out integer;
                          abort_flag :  out integer;
                          istat      :  out integer  );
```

Parameters

Parameters are described as follows.

scheduler — refers to a variable that contains a unique, positive integer value representing the identifier for a frequency–based scheduler. You can obtain this value by making a call to **`FBS_Configure`** (see page 6-7 for an explanation of this subprogram). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing the identifier, you can specify the value **1**.

name refers to a string or dynamically allocated string that contains a standard UNIX path name identifying the program for which information is to be returned. A full or relative path name of up to 1024 characters can be specified.

CPU refers to a variable that contains an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the program for which information is to be returned. Acceptable values and corresponding results are presented in Table 6-17.

Table 6-17. CPU Options: Sched_PGM_Query

Value	Result
0	The first process named by *name* that is currently running on the processor from which the call is made is specified
-1	The first process named by *name* that is currently running on any processor is specified

Bit mask	If (*cpu* & (1<<***i***)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is running on CPU ***i*** is specified
	
	If (*cpu* & (1<<***i***)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is specified

slot refers to a variable that contains an integer value providing the unique frequency–based scheduler process identifier for the process for which information is to be returned. This value is obtained when you make a call to **Sched_PGM_Add** (see page 6-52 for an explanation of this subprogram). This value must be – **1** if you wish to identify the program to be queried only by specifying *name* and *cpu*.

cid refers to a variable to which **Sched_PGM_Query** will return an integer value indicating the scheduling policy under which the specified process has been scheduled

priority refers to a variable to which **Sched_PGM_Query** will return an integer value indicating the specified process's scheduling priority

period refers to a variable to which **Sched_PGM_Query** will return an integer value indicating the frequency with which the specified program is to be wakened in each major frame.

A period of one indicates that the specified program is to be wakened every minor cycle; a period of two indicates that it is to be wakened once every two minor cycles, a period of three once every three minor cycles, and so on

cycle
: refers to a variable to which **`Sched_PGM_Query`** will return an integer value indicating the first minor cycle in which the specified process is scheduled to be wakened in each frame

abort_flag
: refers to a variable to which **`Sched_PGM_Query`** will return an integer value indicating the value of the "halt on overrun" flag. A nonzero value indicates that the flag is set. A value of zero indicates that the flag is not set

istat
: refers to a variable to which **`Sched_PGM_Query`** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

– **`1`**, – **`7`** Scheduler is not configured.

– **`2`** Scheduler does not exist.

– **`3`** Process is not scheduled on this scheduler.

– **`4`** *CPU* value is out of range.

– **`20`** Operation permission is denied to the calling process (see **`intro(2)`**).

– **`24`** Path name specified by *name* is too long.

Sched_PGM_Reschedule

The **`Sched_PGM_Reschedule`** subprogram is invoked to change the scheduling parameters for a process that is scheduled on a frequency–based scheduler. You may wish, for example, to change a program's scheduling policy or priority or the frequency with which it is scheduled to run. You cannot, however, change the CPU on which it has been scheduled.

If you wish to (1) change a process's scheduling policy to the **SCHED_FIFO** or the **SCHED_RR** policy or (2) change the priority of a process scheduled under the **SCHED_FIFO** or the **SCHED_RR** policy, the following conditions must be met:

- The calling process must have the P_RTIME privilege.
- The effective user ID of the calling process must match the effective user ID of the target process (the process for which the scheduling policy and priority are being set), or the calling process must have the P_OWNER privilege.

If the Enhanced Security Utilities are installed and running, the following additional conditions must be met:

- The Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

If you wish to raise the priority of a process scheduled under the **SCHED_OTHER** policy above a per-process or LWP limit, the following conditions must be met:

- The calling process must have the P_TSHAR privilege.
- The effective user ID of the calling process must match the effective user ID of the target process (the process for which the scheduling priority is being set), or the calling process must have the P_OWNER privilege.

If the Enhanced Security Utilities are installed and running, the following additional conditions must be met:

- The Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

For additional information on privileges, refer to the *PowerMAX OS Programming Guide* and the **`intro(2)`** system manual page.

You can identify the process that you wish to reschedule by using one of the following methods:

- Specify the name of the process and the CPU on which it is scheduled.
- Specify the process's frequency-based scheduler process identifier (slot number).
- Specify the name of the process, the CPU on which it is scheduled, and its frequency–based scheduler process identifier.

NOTE

> The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency–based scheduler process identifier.

The Ada specification and corresponding parameters are presented in the following sections.

Ada Specification

```
type pgm_sched_policy is (SCHED_INVALID,
                          SCHED_OTHER)
                          SCHED_FIFO,
                          SCHED_RR);

procedure Sched_PGM_Reschedule(scheduler: in   integer;
                               name     : in   string;
                               CPU      : in   integer;
                               slot     : in out  integer;
                               cid      : in  pgm_sched_policy;
                               priority : in  integer;
                               param    : in  integer;
                               period   : in  integer;
                               cycle    : in  integer;
                               abrt     : in  integer;
                               istat    : out  integer  );


procedure Sched_PGM_Reschedule(scheduler: in  integer;
                               name     : in unbounded_string;
                               CPU      : in  integer;
                               slot     : in out integer;
                               cid      : in  pgm_sched_policy;
                               priority : in  integer;
                               param    : in  integer;
                               period   : in  integer;
                               cycle    : in  integer;
                               abrt     : in  integer;
                               istat    : in  integer  );
```

Parameters

Parameters are described as follows.

scheduler — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process is scheduled. You can obtain this value by making a call to **`FBS_Configure`** (see page 6-7 for an explanation of this subprogram). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing the identifier, you can specify the value – **`1`**.

name — refers to a string or dynamically allocated string that contains a standard UNIX path name identifying the program to be rescheduled. A full or relative path name of up to 1024 characters can be specified.

CPU — refers to a mask that identifies the processor(s) to be used in conjunction with the value of the name parameter to identify the process to be rescheduled. Acceptable values and corresponding results are presented in Table 6-18.

Table 6-18. CPU Options: Sched_PGM_Reschedule

Value	Result
0	The program specified by *name* can be scheduled on the processor from which the call is made
-1	The program specified by *name* can be scheduled on any processor
Bit mask	If (*cpu* & (1<<***i***)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) the program specified by *name* can be scheduled on CPU ***i***

slot
: refers to a variable which provides the unique frequency–based scheduler process identifier for the process to be rescheduled. This value is obtained when you make a call to **Sched_PGM_Add** (see page 6-55 for an explanation of this subprogram). This value must be – **1** if you wish to identify the program to be rescheduled only by specifying *name* and *cpu*.

cid
: refers to a variable that contains an enumeration value indicating the scheduling policy under which the specified program is to be scheduled. Acceptable values are presented as follows (**SCHED_INVALID** may not be supplied here):

SCHED_OTHER
time-sharing scheduling policy

SCHED_FIFO
first–in–first–out (FIFO) scheduling policy

SCHED_RR
round–robin (RR) scheduling policy. Note that a process cannot be scheduled under this policy on a CPU on which servicing of the 60 Hz clock interrupt has been disabled. In such cases, the process will behave as though it were scheduled under the **SCHED_FIFO** policy.

priority
: refers to a variable that contains an integer value indicating the scheduling priority of the specified program. The range of acceptable priority values is governed by the scheduling policy specified.

You can determine the allowable range of priorities associated with each policy (**SCHED_FIFO**, **SCHED_RR**, or **SCHED_OTHER**) by invoking the **run(1)** command from the shell and not specifying any options or arguments (see the corresponding system manual page for an explanation of this command). Higher numerical values correspond to more favorable priorities.

For complete information on scheduling policies and priorities, refer to the "Process Scheduling and Management" chapter of the *PowerMAX OS Programming Guide*.

param — refers to a variable that contains an integer value to be passed to a process that is scheduled on a frequency–based scheduler. This value can be retrieved by the FBS–scheduled process through a call to **`RT_Param`** (see 6-49 for an explanation of this subprogram).

period — refers to a variable that contains an integer value indicating the frequency with which the specified program is to be wakened in each major frame. A period of one indicates that the specified program is to be wakened every minor cycle; a period of two indicates that it is to be wakened once every two minor cycles, a period of three once every three minor cycles, and so on. This value can range from one to the number of minor cycles that compose a frame on the specified scheduler as defined in a call to **`FBS_Configure`** (see page 6-7).

cycle — refers to a variable that contains an integer value indicating the first minor cycle in which the specified program is scheduled to be wakened in each frame. This value can range from zero to the total number of minor cycles per frame minus one. (The total number of minor cycles per frame is specified in a call to **`FBS_Configure`**. See page 6-7 for an explanation of this subprogram).

abrt — refers to a flag that contains an integer value indicating whether or not the scheduler should be stopped in the event that the specified program causes a frame overrun. A nonzero value indicates that the scheduler will be stopped.

istat — refers to a variable to which **`Sched_PGM_Reschedule`** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

– **`1,`**– **`7`** Scheduler is not configured.

– **`2`** Scheduler does not exist.

– **`3`** Process is not scheduled on the specified scheduler.

– **`4`** *CPU***,** *period* or *cycle* value is out of range.

– **`20`** Operation permission is denied to the calling process (see **`intro(2)`**).

– **`29`** There is no space left to perform the reschedule.

– **33** The **sched_setscheduler(3C)** call failed for the scheduled process when attempting to set the scheduling class or priority.

Name_To_Pid – Obtain Process Identifier

This subprogram is invoked to obtain the process identification number (PID) for a selected process name. You can invoke this subprogram to obtain the PID for a process that is scheduled on a frequency–based scheduler and one that is not.

CAUTION

When the PID is returned, the process with which it is associated may no longer be active.

The Ada specification and corresponding parameters are presented in the following sections.

Ada Specification

```
procedure Name_To_Pid (name     : in   string;
                       key      : in  integer;
                       cpu      : in  integer;
                       pid      : out  integer;
                       istat    : out  integer);
```

Parameters

Parameters are described as follows.

name — refers to a variable that contains a standard UNIX path name identifying the process for which the PID is to be returned. A full or relative path name of up to 1024 characters can be specified.

key — refers to a variable that contains an integer value identifying a frequency–based scheduler; this value must be the same value that was specified for *key* when the scheduler was created by making a call to **FBS_Configure** (see page 6-7 for an explanation of this subprogram). If you wish to obtain the PID for a process that is not scheduled on a frequency–based scheduler, specify the value – **1**.

cpu — refers to a variable that contains an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the process for which the process identifier is to be returned. Acceptable values and corresponding results are presented in Table 6-19.

Table 6-19. CPU Options: Name_To_Pid

Value	Result
0	The first process named by *name* that is currently running on the processor from which the call is made is specified
-1	The first process named by *name* that is currently running on any processor is specified

Bit mask	If (*cpu &* (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is running on CPU *i* is specified
	
	If (*cpu &* (1<<*i*)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is specified

pid
: refers to a variable to which **`Name_To_Pid`** will return the process ID of the process that matches the specifications defined by *name*, *key*, and *cpu*. The first process that meets those specifications is returned.

istat
: refers to a variable to which **`Name_To_Pid`** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

`-3` A process with the specified path name is not active on the system.

`-5` An invalid path name has been specified, or the specified path name does not exist on the system.

`-7` The frequency-based scheduler is not configured in the system (and key is something other than -1).

The Performance Monitor Subprograms

The performance monitor subprograms provide access to the key features of the performance monitor. They enable you to perform such basic operations as the following: (1) clear performance monitor values for a process or processor, (2) start and stop performance monitoring for a process or processor, and (3) obtain performance monitor values for a process or processor.

In the sections that follow, all of the performance monitor subprograms contained in the `RT_Interface` package are presented in alphabetical order. Figure 6-2 illustrates the approximate order in which you might invoke the subprograms from an application program.

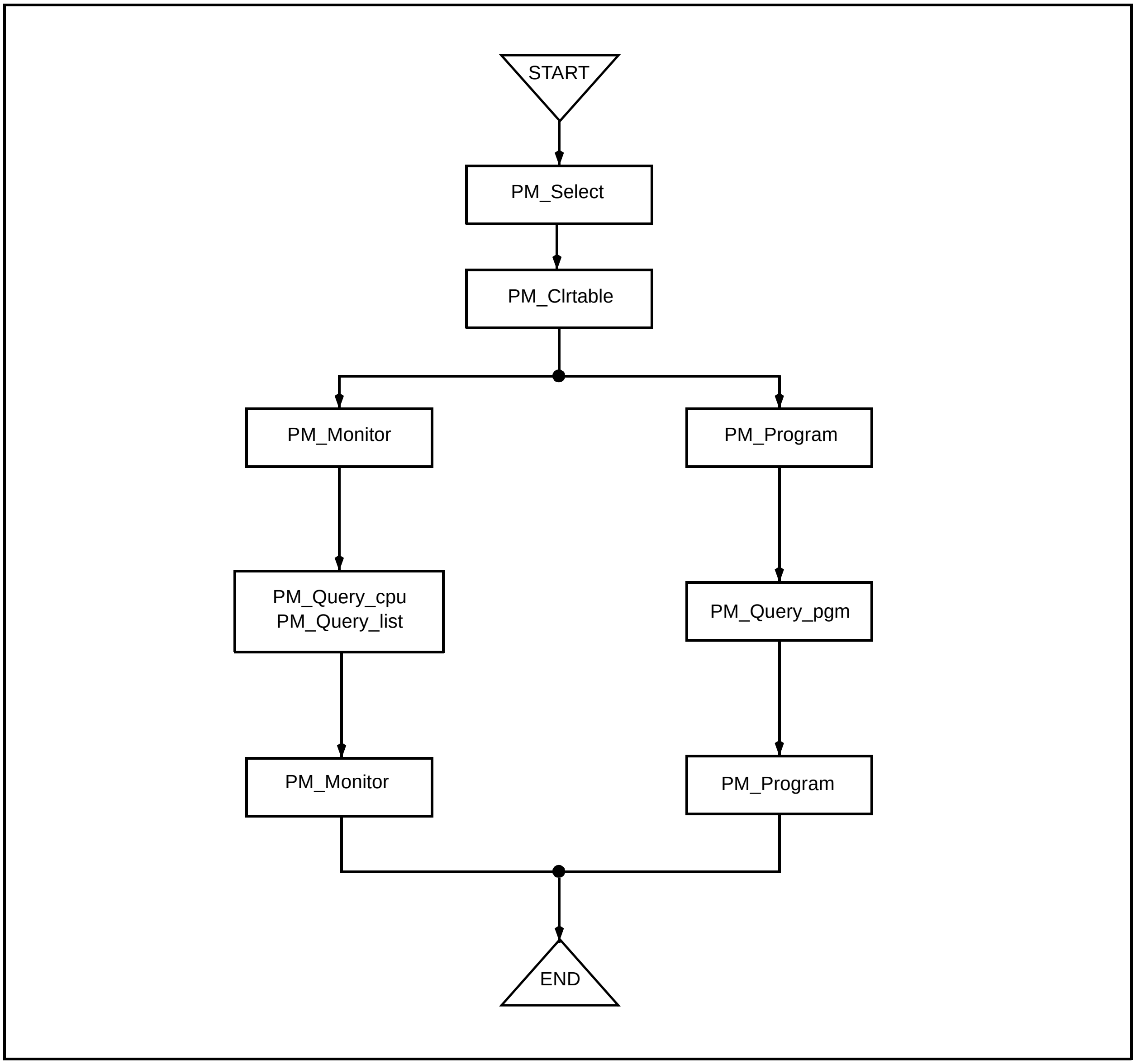

Figure 6-2. Ada Subprogram Call Sequence: Performance Monitor

PM_Clrpgm – Clear Values for a Process

This subprogram is invoked to clear performance monitor values for a particular process that has been scheduled on a frequency–based scheduler. You can identify the process by using one of the following methods:

- Specify the name of the process and the CPU(s) on which it is scheduled.

- Specify the process's frequency–based scheduler process identifier (slot number).
- Specify the name of the process, the CPU on which it is scheduled, and its frequency–based scheduler process identifier.

Ada Specification

```
procedure PM_Clrpgm    ( scheduler   : in integer;
                         name        : in string;
                         CPU         : in integer;
                         slot        : in integer;
                         istat       : out integer );

procedure PM_Clrpgm    ( scheduler   : in integer;
                         name        : in unbounded_string;
                         CPU         : in integer;
                         slot        : in integer;
                         istat       : out integer );
```

Parameters

Parameters are described as follows.

scheduler
: refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process is scheduled. You can obtain this value by making a call to **`FBS_Configure`** (see page 6-7 for an explanation of this subprogram). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of **`–1`**.

name
: refers to a string or dynamically allocated string that contains a standard UNIX path name identifying the process for which values are to be cleared. A full or relative path name of up to 1024 characters can be specified. If this variable is filled with blanks, you must provide the frequency–based scheduler process identifier in the *slot* parameter.

CPU
: refers to a variable that contains an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the process for which values are to be cleared. Acceptable values and corresponding results are presented in Table 6-20.

Table 6-20. CPU Options: PM_Clrpgm

Value	Result
0	The first process named by *name* that is currently running on the processor from which the call is made is specified
-1	The first process named by *name* that is currently running on any processor is specified

Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is currently running on CPU *i* is specified
	
	If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is specified

slot — refers to a variable that contains an integer value providing the unique frequency–based scheduler process identifier for the process for which values are to be cleared. This value is obtained when you make a call to **`Sched_PGM_Add`** (see page 6-52 for an explanation of this subprogram). This value must be – **`1`** if you wish to identify the process only by specifying *name* and *cpu*.

istat — refers to a variable to which **`PM_Clrpgm`** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

– **`1`**,– **`7`** Scheduler is not configured.

– **`2`** Scheduler does not exist.

– **`3`** Process is not scheduled on the specified scheduler.

– **`4`** **CPU** value is out of range.

– **`20`** Operation permission is denied to the calling process (see **`intro(2)`**).

– **`24`** Path name specified by *name* is too long.

PM_Clrtable – Clear Values for Processor(s)

This subprogram is invoked to clear performance monitor values for FBS–scheduled processes on one or more specified processors on a selected scheduler.

The Ada specification and corresponding parameters are presented in the following sections.

Ada Specification

```
type processor_list is array( integer range <> ) of integer;

procedure PM_Clrtable     ( scheduler    : in integer;
                            CPU_list     : in processor_list;
                            istat        : out integer );
```

Parameters

Parameters are described as follows.

scheduler — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the processes are scheduled. You can obtain this value by making a call to **`FBS_Configure`** (see page 6-7 for an explanation of this subprogram). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **`1`**.

CPU_list — refers to an array of integer values that indicate the processor(s) for which performance monitor values are to be cleared. The size of the *CPU_list* array is specified by the user declaration of the actual *CPU_list* parameter. Acceptable values and corresponding results are presented in Table 6-21.

Table 6-21. CPU Options: PM_Clrtable

Value	Result
0	Performance monitor values for FBS–scheduled processes executing on the processor from which the call is made are cleared
-1	Performance monitor values for all processes on the scheduler are cleared
Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU), performance monitor values for processes executing on CPU *i* are cleared

istat — refers to a variable to which **PM_Clrtable** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

- **– 1,– 7** Scheduler is not configured.
- **– 2** Scheduler does not exist.
- **– 3** A processor specified in **CPU_list** is not in this complex.
- **– 20** Operation permission is denied to the calling process (see **intro(2)**).

PM_Monitor – Start/Stop Performance Monitoring on Processor(s)

This subprogram is invoked to start or stop performance monitoring for FBS–scheduled processes on one or more specified processors on a selected scheduler.

The Ada specification and corresponding parameters are presented in the following sections.

Ada Specification

```
type pm_flag_type is  ( turn_off_PM, turn_on_PM );

type processor_list is array( integer range <> ) of integer;

procedure PM_Monitor      ( scheduler    : in integer;
                            pm_flag      : in pm_flag_type;
                            CPU_list     : in processor_list;
                            istat        : out integer );
```

Parameters

Parameters are described as follows.

scheduler — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the processes are scheduled. You can obtain this value by making a call to **FBS_Configure** (see page 6-7 for an explanation of this subprogram). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

pm_flag — refers to a variable that contains an enumeration value indicating whether performance monitoring is to be started or stopped. Specify **turn_on_PM** to indicate that performance monitoring is to be started. Specify **turn_off_PM** to indicate that performance monitoring is to be stopped.

CPU_list — refers to an array of integer values that indicate the processor(s) for which performance monitoring is to be started or stopped. The size of the **CPU_list** array is specified by the user declaration of the actual *CPU_list* parameter. Acceptable values and corresponding results are presented in Table 6-22.

Table 6-22. CPU Options: PM_Monitor

Value	Result
0	Performance monitoring for FBS–scheduled processes executing on the processor from which the call is made is started or stopped
-1	Performance monitoring for all processes on the scheduler is started or stopped
Bit mask	If (*cpu* & (1<<***i***)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU), performance monitoring for processes executing on CPU ***i*** is started or stopped

istat — refers to a variable to which **PM_Monitor** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

– **1**, – **7** Scheduler is not configured.

– **2** Scheduler does not exist.

– **4** A processor specified in **CPU_list** is not in this complex.

– **7** Pmflag is set to start performance monitoring, and the high-resolution timing facility is not configured into the currently executing kernel.

– **20** Operation permission is denied to the calling process (see **intro(2)**).

PM_Program – Start/Stop Performance Monitoring on a Process

This subprogram is invoked to start or stop performance monitoring for a particular process that has been scheduled on a frequency–based scheduler. You can identify the process by using one of the following methods:

- Specify the name of the process and the CPU(s) on which it is scheduled.

- Specify the process's frequency–based scheduler process identifier (slot number).
- Specify the name of the process, the CPU(s) on which it is scheduled, and its frequency–based scheduler process identifier.

The Ada specification and corresponding parameters are presented in the following sections. The dynamic string data type `unbounded_string` is defined in the package `Ada.Strings.Unbounded`, which is located in the **`/usr/ada/default/predefined`** environment.

Ada Specification

```
type pm_flag_type is  ( turn_off_PM, turn_on_PM );

procedure PM_Program     ( scheduler   : in integer;
                           name        : in string;
                           CPU         : in integer;
                           slot        : in integer;
                           pm_flag     : in pm_flag_type;
                           istat       : out integer );

procedure PM_Program     ( scheduler   : in integer;
                           name        : in unbounded_string;
                           CPU         : in integer;
                           slot        : in integer;
                           pm_flag     : in pm_flag_type;
                           istat       : out integer );
```

Parameters

Parameters are described as follows.

scheduler
: refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process is scheduled. You can obtain this value by making a call to **`FBS_Configure`** (see page 6-7 for an explanation of this subprogram). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **`1`**.

name
: refers to a string or dynamically allocated string that contains a standard UNIX path name identifying the process for which performance monitoring is to be started or stopped. A full or relative path name of up to 1024 characters can be specified. If this variable is filled with blanks, you must provide the frequency–based scheduler process identifier in the *slot* parameter.

CPU
: refers to a variable that contains an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the process for which performance monitoring is to be started or stopped. Acceptable

values and corresponding results are presented in Table 6-23.

Table 6-23. CPU Options: PM_Program

Value	Result
0	The first process named by *name* that is currently running on the processor from which the call is made is specified
-1	The first process named by *name* that is currently running on any processor is specified

Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is currently running on CPU *i* is specified
	
	If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is specified

slot — refers to a variable that contains an integer value providing the unique frequency–based scheduler process identifier for the process for which performance monitoring is to be started or stopped. This value is obtained when you make a call to **`Sched_PGM_Add`** (see page 6-52 for an explanation of this subprogram). This value must be – **`1`** if you wish to identify the process only by specifying *name* and *cpu*.

pm_flag — refers to a variable that contains an enumeration value indicating whether performance monitoring is to be started or stopped. Specify **`turn_on_PM`** to indicate that performance monitoring is to be started. Specify **`turn_off_PM`** to indicate that performance monitoring is to be stopped.

istat — refers to a variable to which **`PM_Program`** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

– **`1`**,– **`7`** Scheduler is not configured.

– **`2`** Scheduler does not exist.

– **`3`** Process is not scheduled on the specified scheduler.

– **`4`** **CPU** value is out of range.

– **7** Pmflag is set to start performance monitoring, and the high-resolution timing facility is not configured into the currently executing kernel.

– **20** Operation permission is denied to the calling process (see **intro(2)**).

– **24** Path name specified by *name* is too long.

PM_Query_cpu – Query Values for Selected Processor(s)

This subprogram is invoked to obtain performance monitor values for FBS–scheduled processes on one or more specified processors on a selected scheduler.

The Ada specification and corresponding parameters are presented in the following sections.

Ada Specification

```
type pm_query_buffer_element is record
  last_cycle_time           : integer;
  total_iterations          : integer;
  total_seconds             : integer;
  total_useconds            : integer;
  number_of_overruns        : integer;
  min_cycle_time            : integer;
  min_cycle_cycle           : integer;
  min_cycle_frame           : integer;
  max_cycle_time            : integer;
  max_cycle_cycle           : integer;
  max_cycle_frame           : integer;
  min_frame_time            : integer;
  min_frame_frame           : integer;
  max_frame_time            : integer;
  max_frame_frame           : integer;
end record;


type pm_query_cpu_buffer_element is record
  program_slot              : integer;
  pm_query_buffer           : pm_query_buffer_element;
end record;

 type pm_query_cpu_buffer_type is array( integer range <> )
  of pm_query_cpu_buffer_element;

 procedure PM_Query_cpu    ( scheduler : in integer;
                             CPU       : in integer;
                             buffer    : in out pm_query_cpu_buffer_type;
                             istat     : out integer );
```

Parameters

Parameters are described as follows.

scheduler — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the processes are scheduled. You can obtain this value by making a call to **FBS_Configure** (see page 6-7 for an explanation of this subprogram). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

CPU — refers to a variable that contains an integer value indicating the processor(s) for which performance monitor values are to be obtained. Acceptable values and corresponding results are presented in Table 6-24.

Table 6-24. CPU Options: PM_Query_cpu

Value	Result
0	Performance monitor values for FBS–scheduled processes executing on the processor from which the call is made are returned
-1	Performance monitor values for all processes on the scheduler are returned
Bit mask	If (*cpu* & (1<<***i***)) is set (where ***i*** is an integer ranging from zero to 15 and representing a CPU), performance monitor values for processes executing on CPU ***i*** are returned

buffer — refers to an array of records to which **PM_Query_cpu** will return the performance monitor values for each FBS–scheduled process on the processor(s) specified with the **CPU** parameter. The number of processes for which these values are returned is bound by the size of the Ada **buffer** array. The type of information returned in each record component for a single process is presented in Table 6-25.

Table 6-25. Contents of buffer Record Components: PM_Query_cpu

Component for Process p	Contents
buffer(p).program_slot	The process's frequency–based scheduler process identifier (slot number)
buffer(p).pm_query_buffer.last_cycle_time	The amount of time that the process has spent running from the last time that it has been wakened by the scheduler until it has called **FBS_Wait** (last time)
buffer(p).pm_query_buffer.total_iterations	The number of times that the process has been wakened by the scheduler (total iterations, or cycles)
buffer(p).pm_query_buffer.total_seconds	The total number of seconds that the process has spent running in all cycles (total seconds). The total amount of time that the process has spent running is equal to the value of **total_seconds** plus **total_useconds**.
buffer(p).pm_query_buffer.total_useconds	The additional number of microseconds that the process has spent running in all cycles (total microseconds). The total amount of time that the process has spent running is equal to the value of **total_seconds** plus **total_useconds**.
buffer(p).pm_query_buffer.number_of_overruns	The number of frame overruns caused by the process
buffer(p).pm_query_buffer.min_cycle_time	The least amount of time that the process has spent running in a cycle (minimum cycle time)
buffer(p).pm_query_buffer.min_cycle_cycle	The number of the minor cycle in which the minimum cycle time has occurred (minimum cycle cycle)
buffer(p).pm_query_buffer.min_cycle_frame	The number of the major frame in which the minimum cycle time has occurred (minimum cycle frame)
buffer(p).pm_query_buffer.max_cycle_times	The greatest amount of time that the process has spent running in a cycle (maximum cycle time)
buffer(p).pm_query_buffer.max_cycle_cycle	The number of the minor cycle in which the maximum cycle time has occurred (maximum cycle cycle)
buffer(p).pm_query_buffer.max_cycle_frame	The number of the major frame in which the maximum cycle time has occurred (maximum cycle frame)
buffer(p).pm_query_buffer.min_frame_time	The least amount of time that the process has spent running during a major frame (minimum frame time)
buffer(p).pm_query_buffer.min_frame_frame	The number of the major frame in which the minimum frame time has occurred (minimum frame frame)
buffer(p).pm_query_buffer.max_frame_time	The greatest amount of time that the process has spent running during a major frame (maximum frame time)
buffer(p).pm_query_buffer.max_frame_frame	The number of the major frame in which the maximum frame time has occurred (maximum frame frame)

istat — refers to a variable to which **`PM_Query_cpu`** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

– **1**,– **7** Scheduler is not configured.

– **2** Scheduler does not exist.

– **3** Buffer is too small. Space available will be filled.

– **4** **CPU** value is out of range.

– **20** Operation permission is denied to the calling process (see **`intro(2)`**).

– **27** The service cannot allocate enough memory for the query.

PM_Query_list – Query Values for a List of Processes

This subprogram is invoked to obtain performance monitor values for a list of processes scheduled on a frequency–based scheduler.

The Ada specification and corresponding parameters are presented in the following sections.

Ada Specification

```
type pm_query_buffer_element is record
   last_cycle_time            : integer;
   total_iterations           : integer;
   total_seconds              : integer;
   total_useconds             : integer;
   number_of_overruns         : integer;
   min_cycle_time             : integer;
   min_cycle_cycle            : integer;
   min_cycle_frame            : integer;
   max_cycle_time             : integer;
   max_cycle_cycle            : integer;
   max_cycle_frame            : integer;
   min_frame_time             : integer;
   min_frame_frame            : integer;
   max_frame_time             : integer;
   max_frame_frame            : integer;
end record;
type pm_query_buffer_type is array( integer range <> )
   of pm_query_buffer_element;

 type slot_buffer_type is array( integer range <> ) of integer;

 procedure PM_Query_list    ( scheduler   : in integer;
```

```
slot_list    : in slot_buffer_type;
 buffer      : in out pm_query_buffer_type;
 istat       : out integer );
```

Parameters

Parameters are described as follows.

scheduler — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler for which performance monitor values are requested. You can obtain this value by making a call to **FBS_Configure** (see page 6-7 for an explanation of this subprogram). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

slot_list — refers to an array that consists of the number of elements specified by the size of the Ada **slot_list** array and contains one or more integer values indicating the frequency–based scheduler process identifiers for which performance monitor values are to be returned.

buffer — refers to an array of records to which **PM_Query_list** will return the performance monitor values for each FBS–scheduled process. The number of processes for which these values are returned is bound by the size of the Ada **buffer** array. The type of information returned in each record component for a single process is presented in Table 6-26.

Table 6-26. Contents of buffer Record Components: PM_Query_list

Component for Process p	Contents
buffer(p).last_cycle_time	The amount of time that the process has spent running from the last time that it has been wakened by the scheduler until it has called **FBS_Wait** (last time)
buffer(p).total_iterations	The number of times that the process has been wakened by the scheduler (total iterations, or cycles)
buffer(p).total_seconds	The total number of seconds that the process has spent running in all cycles (total seconds). The total amount of time that the process has spent running is equal to the value of **total_seconds** plus **total_useconds**.
buffer(p).total_useconds	The additional number of microseconds that the process has spent running in all cycles (total microseconds). The total amount of time that the process has spent running is equal to the value of **total_seconds** plus **total_useconds**.
buffer(p).number_of_overruns	The number of frame overruns caused by the process

Table 6-26. Contents of buffer Record Components: PM_Query_list (Cont.)

Component for Process p	Contents
buffer(p).min_cycle_time	The least amount of time that the process has spent running in a cycle (minimum cycle time)
buffer(p).min_cycle_cycle	The number of the minor cycle in which the minimum cycle time has occurred (minimum cycle cycle)
buffer(p).min_cycle_frame	The number of the major frame in which the minimum cycle time has occurred (minimum cycle frame)
buffer(p).max_cycle_time	The greatest amount of time that the process has spent running in a cycle (maximum cycle time)
buffer(p).max_cycle_cycle	The number of the minor cycle in which the maximum cycle time has occurred (maximum cycle cycle)
buffer(p).max_cycle_frame	The number of the major frame in which the maximum cycle time has occurred (maximum cycle frame)
buffer(p).min_frame_time	The least amount of time that the process has spent running during a major frame (minimum frame time)
buffer(p).min_frame_frame	The number of the major frame in which the minimum frame time has occurred (minimum frame frame)
buffer(p).max_frame_time	The greatest amount of time that the process has spent running during a major frame (maximum frame time)
buffer(p).max_frame_frame	The number of the major frame in which the maximum frame time has occurred (maximum frame frame)

istat — refers to a variable to which **PM_Query_list** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

– **1**, – **7** Scheduler is not configured.

– **2** Scheduler does not exist.

– **3** Buffer is too small. Space available will be filled.

– **4** One of the **slot** values is out of range.

– **20** Operation permission is denied to the calling process (see **intro(2)**).

– **27** The service cannot allocate enough memory for the query.

PM_Query_pgm – Query Values for a Selected Process

This subprogram is invoked to obtain performance monitor values for a particular process scheduled on a frequency–based scheduler.

The Ada specification and corresponding parameters are presented in the following sections. The dynamic string data type `unbounded_string` is defined in the package `Ada.Strings.Unbounded`, which is located in the **`/usr/ada/default/predefined`** environment.

Ada Specification

```
type pm_query_buffer_element is record
   last_cycle_time          : integer;
   total_iterations         : integer;
   total_seconds            : integer;
   total_useconds           : integer;
   number_of_overruns       : integer;
   min_cycle_time           : integer;
   min_cycle_cycle          : integer;
   min_cycle_frame          : integer;
   max_cycle_time           : integer;
   max_cycle_cycle          : integer;
   max_cycle_frame          : integer;
   min_frame_time           : integer;
   min_frame_frame          : integer;
   max_frame_time           : integer;
   max_frame_frame          : integer;
end record;

procedure PM_Query_pgm   ( scheduler    : in integer;
                           name         : in out unbounded_string;
                           CPU          : in integer;
                           slot         : in integer;
                           buffer       : out pm_query_buffer_element;
                           istat        : out integer );
```

Parameters

Parameters are described as follows.

scheduler
: refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler for which performance monitor values are requested. You can obtain this value by making a call to **`FBS_Configure`** (see page 6-7 for an explanation of this subprogram). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **`1`**.

name
: refers to a dynamically allocated string that contains a standard UNIX path name identifying the process for which performance monitoring values are to be returned. A full or relative path name of up to 1024 characters can be specified. If this variable is filled with blanks, you must provide the

frequency–based scheduler process identifier in the *slot* parameter.

CPU — refers to a variable that contains an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the process for which performance monitoring values are to be returned. Acceptable values and corresponding results are presented in Table 6-27.

Table 6-27. CPU Options: PM_Query_pgm

Value	Result
0	The first process named by *name* that is currently running on the processor from which the call is made is specified
-1	The first process named by *name* that is currently running on any processor is specified

Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is currently running on CPU *i* is specified
	
	If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is specified

slot — refers to a variable that contains an integer value providing the unique frequency–based scheduler process identifier for the process for which performance monitoring values are to be returned. This value is obtained when you make a call to **`Sched_PGM_Add`** (see page 6-52 for an explanation of this subprogram). This value must be – **`1`** if you wish to identify the process only by specifying *name* and *cpu.*

buffer — refers to a record to which **`PM_Query_pgm`** will return the performance monitor values for the specified process. The information returned in each component of the record is presented in Table 6-28.

Table 6-28. Contents of buffer Record Components: PM_Query_pgm

Component for Process p	Contents
buffer(p).last_cycle_time	The amount of time that the process has spent running from the last time that it has been wakened by the scheduler until it has called **FBS_Wait** (last time)
buffer(p).total_iterations	The number of times that the process has been wakened by the scheduler (total iterations, or cycles)
buffer(p).total_seconds	The total number of seconds that the process has spent running in all cycles (total seconds). The total amount of time that the process has spent running is equal to the value of **total_seconds** plus **total_useconds**.
buffer(p).total_useconds	The additional number of microseconds that the process has spent running in all cycles (total microseconds). The total amount of time that the process has spent running is equal to the value of **total_seconds** plus **total_useconds**.
buffer(p).number_of_overruns	The number of frame overruns caused by the process
buffer(p).min_cycle_time	The least amount of time that the process has spent running in a cycle (minimum cycle time)
buffer(p).min_cycle_cycle	The number of the minor cycle in which the minimum cycle time has occurred (minimum cycle cycle)
buffer(p).min_cycle_frame	The number of the major frame in which the minimum cycle time has occurred (minimum cycle frame)
buffer(p).max_cycle_time	The greatest amount of time that the process has spent running in a cycle (maximum cycle time)
buffer(p).max_cycle_cycle	The number of the minor cycle in which the maximum cycle time has occurred (maximum cycle cycle)
buffer(p).max_cycle_frame	The number of the major frame in which the maximum cycle time has occurred (maximum cycle frame)
buffer(p).min_frame_time	The least amount of time that the process has spent running during a major frame (minimum frame time)
buffer(p).min_frame_frame	The number of the major frame in which the minimum frame time has occurred (minimum frame frame)
buffer(p).max_frame_time	The greatest amount of time that the process has spent running during a major frame (maximum frame time)
buffer(p).max_frame_frame	The number of the major frame in which the maximum frame time has occurred (maximum frame frame)

istat — refers to a variable to which **PM_Query_pgm** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

– **1**, – **7** Scheduler is not configured.

– **2** Scheduler does not exist.

– **3** Specified process is not scheduled on the specified scheduler.

– **4** **CPU** value is out of range.

– **20** Operation permission is denied to the calling process (see **intro(2)**).

– **24** Path name specified by *name* is too long.

PM_Querytimer – Query Performance Monitor Mode

This subprogram is invoked to determine whether performance monitor timing values include or exclude time spent servicing interrupts. The timing mode can be set to include or exclude interrupt time.

The Ada specification and corresponding parameters are presented in the following sections.

Ada Specification

```
type timing_mode_type is ( exclude_interrupt_time, include_interrupt_time );

procedure PM_Querytimer    ( mode    : out timing_mode_type );
```

Parameter

PM_Querytimer requires one parameter: *mode*. *Mode* refers to a variable to which **PM_Querytimer** will return an enumeration value indicating whether performance monitor timing values include or exclude time spent servicing interrupts. The enumeration value of **include_interrupt_time** or **exclude_interrupt_time** will be returned.

PM_Select – Select Performance Monitor Mode

This subprogram is invoked to select the timing mode under which the performance monitor is to run. The timing mode can be set to include or exclude time spent servicing interrupts. Note that to set the timing mode, the calling process must have the P_RTIME privilege (for additional information on privileges, refer to the *PowerMAX OS Programming Guide* and the **intro(2)** system manual page).

CAUTION

> The timing mode for the high–resolution timing facility is set system–wide. It affects all processes running on all CPUs.

The Ada specification and corresponding parameters are presented in the following sections.

Ada Specification

```
type timing_mode_type is (exclude_interrupt_time, include_interrupt_time );

procedure PM_Select      (   mode        : in timing_mode_type;
                             istat       : out integer );
```

Parameters

Parameters are described as follows.

mode
: refers to a variable that contains an enumeration value of **`include_interrupt_time`** or **`exclude_interrupt_time`** to indicate whether interrupt time is included in or excluded from performance monitor timing values.

istat
: refers to a variable to which **`PM_Select`** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. The nonzero values that may be returned are explained as follows:

  – **7** The high–resolution timing facility is not configured in the currently executing kernel.

  – **20** Operation permission is denied to the calling process (see **`intro(2)`**).

Compiling and Linking Procedures

To compile and link an Ada program using the MAXAda product, the command line instructions are as follows:

`/usr/ada/bin/a.mkenv`

`/usr/ada/bin/a.intro` *source_file*

`/usr/ada/bin/a.partition -create active` *main_unit_name*

`/usr/ada/bin/a.build` *main_unit_name*

For additional information on compiling and linking procedures, refer to the *MAXAda Reference Manual* (0890516).

7
The C Library Interface

7
The C Library Interface

The real–time static and dynamic linked libraries for C, **`/usr/lib/librt.a`** and **`/usr/lib/librt.so`**, respectively contains routines that enable you to perform the entire range of functions associated with the frequency–based scheduler and the performance monitor. The frequency–based scheduler routines are presented in "The FBS Routines." The performance monitor routines are presented in "The Performance Monitor Routines." The following information is provided for each routine:

- A description of the routine
- The C specification and call needed to reference the routine in an application program
- Detailed descriptions of each parameter
- The return value

Procedures for compiling and linking user programs are presented in "Compiling and Linking Programs." An example program that illustrates use of the C library interface to the frequency–based scheduler and the performance monitor is provided in Appendix C.

The FBS Routines

The FBS routines provide access to the key features of the scheduler. They enable you to perform such basic operations as:

- Configure a scheduler
- Schedule programs on it
- Set up and connect a timing source to a scheduler
- Start, stop, and resume scheduling on a scheduler
- Get information about scheduled processes
- Reschedule and remove scheduled processes
- Disconnect a timing source
- Register/unregister a Coupled FBS timing device
- Remove a scheduler.

In the sections that follow, all of the FBS routines contained in the **`librt`** library are presented in alphabetical order. Figure 7-1 illustrates the approximate order in which you might call the routines from an application program.

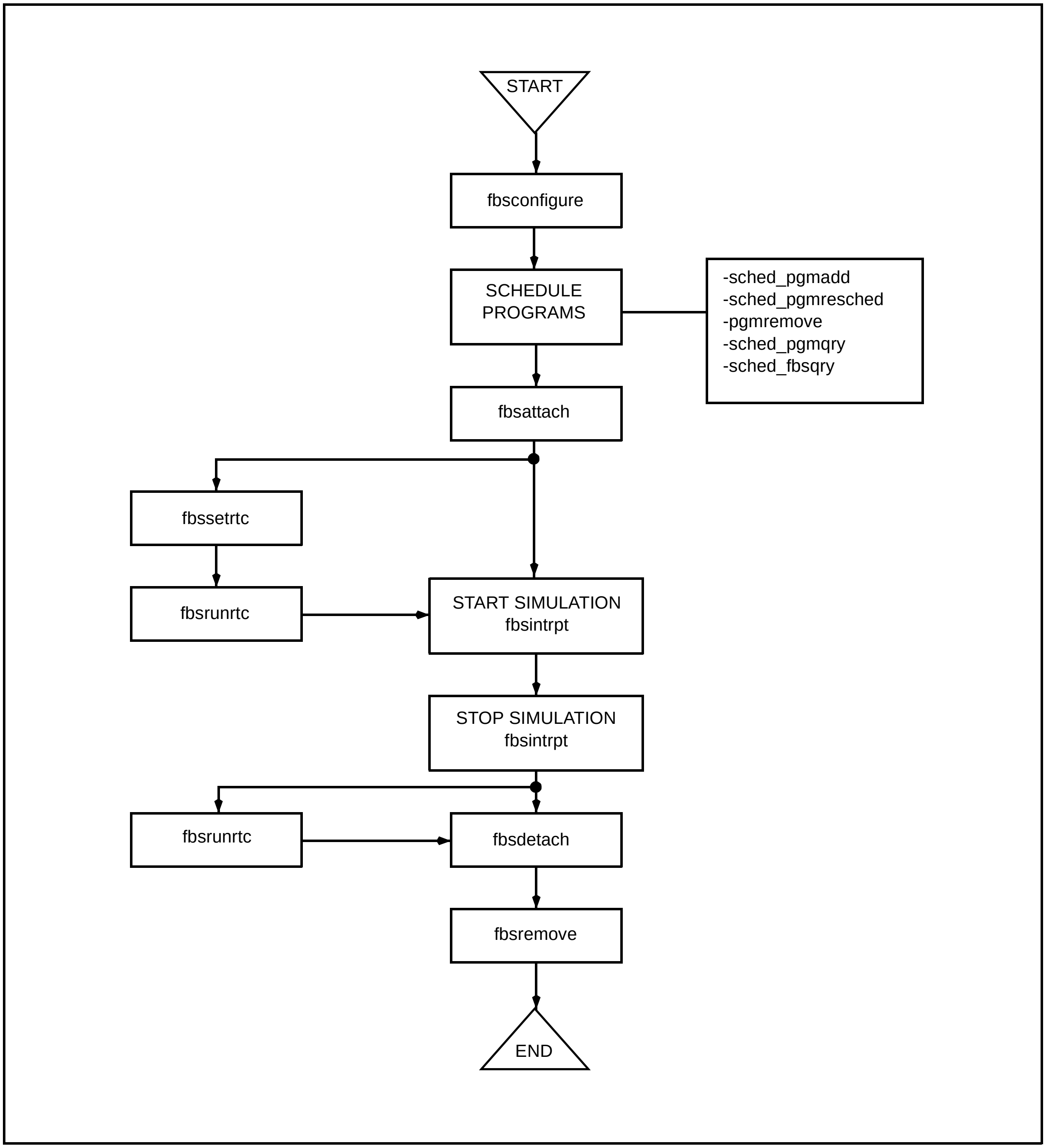

Figure 7-1. C Library Call Sequence: FBS

Fbsaccess – Change Permissions for an FBS

This routine is invoked to change the permissions assigned for a selected frequency–based scheduler. It is important to note that the permissions can be changed only by a process that has the P_OWNER privilege or has an effective user ID that is equal to that of the owner/creator of the frequency–based scheduler.

If the Enhanced Security Utilities are installed and running, the following conditions must also be met:

- The calling process and the frequency-based scheduler must have identical security levels, or the process must have both P_MACREAD and P_MACWRITE privileges.

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int fbsaccess(fbs_id, uid, gid, permissions)
int fbs_id;
int uid;
int gid;
int permissions;
```

Call

```
int istat;
istat = fbsaccess(fbs_id, uid, gid, permissions);
```

Parameters

Parameters are described as follows.

fbs_id	refers to a variable that contains a unique, positive integer value representing the identifier for a frequency–based scheduler. You can obtain this value by making a call to **fbsconfigure** (see page 7-6 for an explanation of this routine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing the identifier, you can specify the value – **1**.
uid	refers to a variable that contains an integer value representing the effective user ID of the specified frequency–based scheduler.
gid	refers to a variable that contains an integer value representing the effective group ID of the specified frequency–based scheduler.
permissions	refers to a variable that contains a bit pattern used to set the permissions associated with the specified frequency–based scheduler. Bit patterns and corresponding permissions are

presented in Table 7-1. Additional information on setting permissions for frequency–based scheduler operations is provided in the system manual page **intro(2)**.

Table 7-1. FBS Permissions

Bit Pattern	Permissions
400	Read by user
200	Alter by user
060	Read, alter by group
006	Read, alter by others

Return Value

A return value of **0** indicates that the call has been successful. A return value of – **1** indicates that an error has occurred; **errno** is set to indicate the error. Refer to the system manual page **fbsaccess(3rt)** for a listing of the types of errors that may occur.

Fbsattach – Attach Timing Source to an FBS

This routine is invoked to attach a timing source to a frequency–based scheduler or to specify end–of–cycle scheduling. The timing source can be a real–time clock, an edge–triggered interrupt device, or a user–supplied real–time device.

NOTE

Routines contained in the C library do not provide the functionality to set up and control operation of an edge–triggered interrupt device or a user–supplied device, as they do for a real–time clock. Procedures for using a real–time clock are described in detail in Chapter 3. Procedures for using an edge–triggered interrupt and a user–supplied real–time device are also explained in that chapter.

To use a real–time clock as the timing source for a frequency–based scheduler on a PowerMAX OS system on which the Enhanced Security Utilities are installed, you must have enough privilege to open the device. Refer to the “Trusted Facility Management” chapter of *System Administration Volume 1* for an explanation of the procedures for using devices when the Enhanced Security Utilities are installed.

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int  fbsattach(fbs_id, devname)
int  fbs_id;
char *devname;
```

Call

```
int istat;
istat = fbsattach(fbs_id, devname);
```

Parameters

Parameters are described as follows.

fbs_id — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler for which the timing source is to be attached or end–of–cycle scheduling specified. You can obtain this value by making a call to **`fbsconfigure`** (see 7-6 for an explanation of this routine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **`1`**.

devname — refers to a variable that contains a null string or the path name of the device that is to be used as the timing source for the specified scheduler. If *devname* contains a null string, end–of–cycle scheduling is specified; that is, execution of the processes in the next minor cycle will occur when the last process scheduled to execute in the current minor cycle finishes its execution for that cycle. If *devname* contains a path name, it may refer to a real–time clock, an edge–triggered interrupt, or a user–supplied device.

If the device is a real–time clock or an edge–triggered interrupt, the path name must be of a certain form. Refer to Chapter 3 for detailed information on the form associated with each type of device.

If the device is a user–supplied device, the path name must be a valid UNIX path name. The device must support the IOCTLVECNUM **`ioctl(2)`** call (see Chapter 3 for additional information).

If the device is a Coupled FBS timing device, the path name must be of a certain form. Refer to Chapter 3 for detailed information on the form associated with a Coupled FBS timing source.

Return Value

A return value of **0** indicates that the call has been successful. A return value of – **1** indicates that an error has occurred; **errno** is set to indicate the error. Refer to the system manual page **fbsattach(3rt)** for a listing of the types of errors that may occur.

Fbsconfigure – Configure an FBS

This routine is invoked to configure a frequency–based scheduler or to obtain configuration details for a frequency–based scheduler that has already been configured. Note that to configure a scheduler, the calling process must have the P_RTIME privilege (for additional information on privileges, refer to the *PowerMAX OS Programming Guide* and the **intro(2)** system manual page).

If you wish to configure a scheduler, you must specify a *key*, which is a user–chosen numeric identifier for a frequency–based scheduler. You must also specify a *configflg*, which is a word that sets the permission and control flag bits to characterize the scheduler.

The permissions are defined in the system manual page **intro(2)**.

The control flags are described in the header file <**sys/ipc.h**>. They include **IPC_CREAT** and **IPC_EXCL**. Setting the **IPC_CREAT** bit without setting the **IPC_EXCL** bit ensures that a new frequency–based scheduler is created if one corresponding to the value of *key* does not exist; it results in the return of the associated frequency–based scheduler identifier if one does exist and if all of the following conditions are met:

- The number of minor cycles specified by the *cycles* parameter matches the number of minor cycles associated with the existing scheduler
- The maximum specified by the *progs* parameter is less than or equal to the maximum number of processes per minor cycle associated with the existing scheduler
- The maximum specified by the *max* parameter is less than or equal to the maximum number of processes allowed on the existing scheduler at one time

Setting both the **IPC_CREAT** and the **IPC_EXCL** bits results in the creation of a new scheduler if one corresponding to the value of **key** does not exist; it ensures that an error is returned if one does exist.

A unique, nonnegative frequency–based scheduler identifier and corresponding data structure will be created for the specified key if the number of frequency–based schedulers already configured is less than the maximum number of schedulers allowed on your system (see Chapter 2 for a description of system tunable parameters) and if one of the following conditions is met:

- The value of *key* is equal to **IPC_PRIVATE** (that is, zero)
- The value of *key* is not associated with a frequency–based scheduler identifier and (*configflg* **& IPC_CREAT**) is “true”

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int fbsconfigure(fbs_buf)
struct fbsconfig_ds {
    int key;
    int cycles;
    int progs;
    int max;
    int reset;
    int configflg;
    int fbs_id;
} *fbs_buf;
```

Call

```
struct fbsconfig_ds  fbs_buf;
int istat;
istat = fbsconfigure(&fbs_buf);
```

Parameters

To create a frequency–based scheduler, you must specify the following parameters as described.

fbs_buf — refers to an **`fbsconfig_ds`** structure that contains the information with which you wish to configure a frequency–based scheduler. The type of information that is specified in each component is presented in Table 7-2.

Table 7-2. Contents of Structure Components: fbsconfigure

Component	Contents
key	An integer value identifying the frequency–based scheduler that is to be created. Note that the number of schedulers that can be configured at one time cannot exceed the value of FBSMNI, which is the maximum number of frequency–based schedulers permitted on your system (see Chapter 2 for a description of system tunable parameters).
cycles	An integer value indicating the number of minor cycles that compose a frame on the specified scheduler.
progs	An integer value indicating the maximum number of programs that can be scheduled to execute during one minor cycle.
max	An integer value indicating the maximum number of programs that can be scheduled on the specified scheduler at one time. This value must be less than or equal to the product that is obtained by multiplying the values specified for the *cycles* and *progs* parameters.

Table 7-2. Contents of Structure Components: fbsconfigure (Cont.)

Component	Contents
reset	An integer value indicating whether or not processes currently scheduled on the specified scheduler are to be killed before the scheduler is reconfigured. Acceptable values and corresponding results are as follows: <0 Kill and remove all processes currently scheduled on the specified scheduler 0 Ignore all processes currently scheduled on the specified scheduler >0 Remove all processes currently scheduled on the specified scheduler
configflg	An integer value indicating the control flags and permissions assigned to the specified scheduler. See the header file **<sys/ipc.h>** to determine the locations of the bits.
fbs_id	a unique, positive integer value that is returned by **fbsconfigure** and represents the identifier for the specified frequency–based scheduler. It is important to note that this identifier is required by most of the library routines for the FBS and the performance monitor

To obtain information for an existing frequency–based scheduler, you must specify the following parameters as described.

fbs_buf — refers to an **fbsconfig_ds** structure to which **fbsconfigure** will return information for an existing frequency - based scheduler. The type of information that is specified or returned in each component is presented in Table 7-3.

Table 7-3. Contents of Structure Components: fbsconfigure

Component	Contents
key	an integer value identifying the frequency–based scheduler for which configuration information is to be returned. If this value is zero, the frequency–based scheduler identifier associated with this scheduler must also be provided by using the **fbs_id** component.
cycles	refers to the component that contains the integer value zero, indicating that current configuration information for the specified scheduler is to be returned. **Fbsconfigure** will return to this component an integer value indicating the number of minor cycles that compose a frame on the specified scheduler.
progs	refers to the component to which **fbsconfigure** will return the maximum number of programs that can be scheduled to run during one minor cycle on the specified scheduler
max	refers to the component to which **fbsconfigure** will return the maximum number of programs that can be scheduled on the specified scheduler at one time
configflg	refers to the component to which **fbsconfigure** will return the permissions assigned to the specified scheduler.
fbs_id	refers to the component to which **fbsconfigure** will return a unique, positive integer value representing the identifier for the specified frequency–based scheduler. If you specify a key of **0**, this component must contain the related frequency–based scheduler identifier.

Return Value

A return value of **0** indicates that the call has been successful. A return value of – **1** indicates that an error has occurred; **errno** is set to indicate the error. Refer to the system manual page **fbsconfigure(3rt)** for a listing of the types of errors that may occur.

Fbscycle – Return Minor Cycle/Major Frame Count

This routine is invoked to obtain the current minor cycle and major frame count values for a frequency–based scheduler. These values enable you to determine the progress of a simulation.

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int fbscycle(fbs_id, cycle_buf)
```

```
int fbs_id;
struct fbscycle_ds {
   int ccycle;
   int cframe;
} *cycle_buf;
```

Call

```
struct fbscycle_ds  cycle_buf;
int istat;
istat = fbscycle(fbs_id, &cycle_buf);
```

Parameters

Parameters are described as follows.

fbs_id — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler for which you wish to obtain the current cycle and frame counts. You can obtain this value by making a call to **fbsconfigure** (see page 7-6 for an explanation of this routine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

cycle_buf — refers to an **fbscycle_ds** structure to which **fbscycle** will return integer values indicating the current minor cycle and major frame for the specified scheduler. The **ccycle** component will contain the number of the cycle. The **cframe** component will contain the number of the frame.

Return Value

A return value of **0** indicates that the call has been successful. A return value of – **1** indicates that an error has occurred; **errno** is set to indicate the error. Refer to the system manual page **fbscycle(3rt)** for a listing of the types of errors that may occur.

Fbsdetach – Detach Timing Source from an FBS

This routine is invoked to detach the currently attached timing source from a frequency–based scheduler or to disable end–of–cycle scheduling. If the timing source is a real-time clock, it is recommended that you stop the clock prior to invoking this routine. You can do so by making a call to **fbsrunrtc** (see page 7-26 for an explanation of this routine).

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int fbsdetach(fbs_id)
int fbs_id;
```

Call

```
int istat;
istat = fbsdetach(fbs_id);
```

Parameters

Parameters are described as follows.

fbs_id — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler from which you wish to detach the currently attached timing source or for which you wish to disable end–of–cycle scheduling. You can obtain this value by making a call to **`fbsconfigure`** (see page 7-6 for an explanation of this routine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of –**`1`**.

Return Value

A return value of **`0`** indicates that the call has been successful. A return value of – **`1`** indicates that an error has occurred; **`errno`** is set to indicate the error. Refer to the system manual page **`fbsdetach(3rt)`** for a listing of the types of errors that may occur.

Fbsgetrtc – Obtain Current Values for Real–Time Clock

This routine is invoked to obtain the current count and resolution values for the real–time clock that is attached to a specified frequency–based scheduler.

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int fbsgetrtc(fbs_id, count, resolution)
int fbs_id;
int *count;
int *resolution;
```

Call

```
int istat;
int count;
int resolution;
istat = fbsgetrtc(fbs_id, &count, &resolution);
```

Parameters

Parameters are described as follows.

fbs_id — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler to which the real–time clock is attached. You can obtain this value by making a call to **fbsconfigure** (see page 7-6 for an explanation of this routine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

count — refers to a variable that contains the address to which **fbsgetrtc** will return an integer value indicating the current number of clock counts per minor cycle. This value can range from one to 65535.

resolution — refers to a variable that contains the address to which **fbsgetrtc** will return an integer value indicating the current duration in microseconds of one clock count. This value will be one of the following: 1, 10, 100, 1000, or 10000.

Return Value

A return value of **0** indicates that the call has been successful. A return value of **–1** indicates that an error has occurred; **errno** is set to indicate the error. Refer to the system manual page **fbsgetrtc(3rt)** for a listing of the types of errors that may occur.

Fbsid – Return the FBS Identifier for a Key

This routine is invoked to obtain the frequency–based scheduler identifier associated with a particular user–specified key. The key must match the key that was specified when the scheduler was created by making a call to **fbsconfigure(3rt)**.

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int fbsid(fbs_key)
int fbs_key;
```

Call

```
int fbs_id;
fbs_id = fbsid(fbs_key);
```

Parameters

Parameters are described as follows.

fbs_key — refers to a variable that contains an integer value identifying a frequency–based scheduler; this value must be the same value that was specified for *key* when the scheduler was created by making a call to **fbsconfigure** (see page 7-6 for an explanation of this subroutine).

Return Value

Upon successful completion, **fbsid** returns an integer value representing the unique frequency–based scheduler identifier associated with the key. A return value of – **1** indicates that an error has occurred; **errno** is set to indicate the error. Refer to the system manual page **fbsid(3rt)** for a listing of the types of errors that may occur.

Fbsinfo – Return Information for an FBS

This routine is invoked to obtain information that is related to a selected frequency–based scheduler but cannot be obtained by invoking other routines (for example, **sched_fbsqry**, **sched_pgmqry**). Such information includes the following:

- The user and group IDs of the owner and the creator of the scheduler
- The permissions assigned for the scheduler
- The key associated with the scheduler's identifier
- The total number of overruns for all processes on the scheduler
- The CPUs that are active in the system
- The CPUs on which performance monitoring has been enabled
- The FBS–enabled flag
- The path name of the device that has been attached to the scheduler

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int fbsinfo(fbs_id, info_buf, devname)
int fbs_id;
struct fbsinfo_ds {
   int uid;
   int gid;
   int cuid;
   int cgid;
   int mode
   int key;
   int flags;
   int devid;
   int overruns;
   int cpuactive;
   int pm_cpuactive;
   int enabled;
```

```
    int filler[29]
} *info_buf;
char *devname;
```

Call

```
struct fbsinfo_ds  info_buf;
int istat;
istat = fbsinfo(fbs_id, &info_buf, devname);
```

Parameters

Parameters are described as follows.

fbs_id
: refers to a variable that contains a unique, positive integer value representing the identifier for a frequency–based scheduler. You can obtain this value by making a call to **fbsconfigure** (see page 7-6 for an explanation of this routine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

info_buf
: refers to an **fbsinfo_ds** structure to which **fbsinfo** will return information about the specified scheduler. The information returned in each component of the structure is presented in Table 7-4.

Table 7-4. Contents of Structure Components: fbsinfo

Component	Contents
uid	owner's user ID
gid	owner's group ID
cuid	creator's user ID
cgid	creator's group ID
mode	access modes
key	key
flags	flags word
devid	reserved for future use
overruns	total number of hard overruns for all processes on the scheduler
cpuactive	mask of CPUs active in the system
pm_cpuactive	mask of CPUs on which performance monitoring has been enabled
enabled	FBS–enabled flag
filler	reserved for future use

devname — refers to a variable to which **fbsinfo** will return the path name of the device that is being used as the timing source for the specified frequency–based scheduler. If end–of–cycle scheduling has been specified, *devname* will contain a null string.

Return Value

A return value of **0** indicates that the call has been successful. A return value of – **1** indicates that an error has occurred; **errno** is set to indicate the error. Refer to the system manual page **fbsinfo(3rt)** for a listing of the types of errors that may occur.

Fbsinfo_rdev - Return rdevfs timing device information

This routine may be used to obtain information about the rdevfs Coupled FBS timing device specified by the rdevfs device file path name.

The information returned from this routine includes the following:

- The type of Coupled FBS timing device; either RCIM Coupled or Closely-Coupled. See Chapter 3 for more information about these two types of timing devices.
- The hostname of the host where the timing device actually resides (where the device interrupt originates).
- A list of all the hostnames of the hosts where this device is registered.
- A list of all the hostnames of the hosts that currently have schedulers attached to this device.
- The path name of the actual device on the host where the device resides.

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>
int fbsinfo_rdev(rdevfs_name, info_buf)
char *rdevfs_name;
struct fbsinfo_rdev_ds {
        u_int attr_flags;
        char device_name[MAXPATHLEN];
        int num_hosts;
struct fbsinfo_rdev_host_ds *host_array;
} *info_buf;
```

Call

```
struct fbsinfo_rdev_ds info_buf;
int status;
char *rdevfs_name;
status = fbsinfo_rdev(rdevfs_name, &info_buf);
```

Parameters

Parameters are described as follows.

rdevfs_name — Refers to a caller-supplied character string pointer that points to the **/dev/rdev/<hostname>/device<*n*>** path name of the Coupled FBS timing device that the caller wishes to obtain information about.

info_buf — Refers to the fbsinfo_rdev_ds structure pointer that points to the structure where the Coupled FBS timing device information will be returned. The caller is also required to setup certain fields within this structure before calling this function. Use of this structure is detailed in Table 7-5 below.

Table 7-5. Contents of Structure Components: fbsinfo_rdev_ds

Component	Contents
attr_flag	The type of timing device is returned in this field; FBS_CC_TYPE for a Closely-Coupled timing device, or FBS_RC_TYPE for a RCIM Coupled timing device. All other flag bits in this field are reserved for future use.
device_name	The path name of the actual device on the host where the device resides is returned in this location.
num_hosts	The caller supplies the size of the host_array in this field before calling fbsinfo_rdev. When this call is successful or when -1 is returned and errno is set to EFBIG, fbsinfo_rdev updates this field with the actual number of registered hosts for this timing device. When this call is successful, then the value returned in this field represents the number of valid entries in the host array that may be examined by the caller upon return from this function call. When -1 is returned and errno is set to EFBIG, then fbsinfo_rdev fills in all the available host_array entry locations with per-host information. The per-host information for all hosts is not returned in this case. The caller may either simply examine all the entries in the host_array, or they may allocate a larger host_array and call this function again in order to obtain the per-host information for all the registered hosts, instead of just a subset of the registered hosts.
host_array	The caller should set this field up before calling the fbsinfo_rdev function. This field should point to an array of fbsinfo_rdev_host_ds structures where this function returns information about each registered host. The size of this array should be equal to the value that the caller specified in the num_hosts field. The per-host information about each host where the specified Coupled FBS timing device has been registered is returned in the user's fbsinfo_rdev_host_ds structure array, where each entry in this array is detailed in Table 7-6 below.

Table 7-6. Contents of Structure Components: fbsinfo_rdev_host_ds

Component	Contents
hostname	This location contains the hostname of the host where this timing device is registered.
host_flag	This field will contain additional information pertaining to the host returned in the hostname field. The valid flags are FBSINFO_SCHED_ATTACHED, which indicates that this host currently has a scheduler attached to this timing device, and FBSINFO_INTR_SOURCE, which indicates that the timing device resides on this host.

Return Value

A return value of 0 indicates that the call has been successful. A return value of -1 indicates that an error has occurred, and errno is set to indicate the error. Refer to the system manual page **`fbsinfo_rdev(3rt)`** for a listing of the types of errors that may occur.

NOTE

The **fbsinfo_rdev** function call is not compatible for use with timing devices that were registered with a **fbs_register_cluster_device** function call. In this case, the user should use the **fbsinfo_cluster** function call to obtain additional information about the Closely-Coupled timing device. However, the **fbs_register_cluster_device** and **fbs_unregister_cluster_device** function calls are obsolete and users are encouraged to make use of the **fbs_register_rdev**, **fbs_unregister_rdev** and **fbsinfo_rdev** function calls.

Fbsinfo_cluster - Return cluster information for an FBS

This routine is invoked to obtain information about the Closely-Coupled timing device that a selected frequency-based scheduler is currently attached to. The information returned from this routine includes the following:

- The SBC board ID where the Closely-Coupled timing device actually resides
- The path name of the actual device on the SBC board where the device resides
- A bit mask of SBC board IDs of the SBCs that currently have schedulers attached to this device

Note that the selected frequency-based scheduler must be currently attached to a Closely-Coupled timing device in order for this routine call to be successful.

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>
int fbsinfo_cluster(fbsid, info_buf, devname)
int fbs_id;
struct fbsinfo_cluster_ds {
        int     sbc_id_location;
        uint_t  sbc_id_attached_mask;
        int     filler[6];
} *info_buf;
char *devname;
```

Call

```
struct fbsinfo_cluster_ds info_buf;
int istat;
istat = fbsinfo_cluster(fbs_id, &info_buf, devname);
```

Parameters

Parameters are described as follows.

fbs_id — refers to a variable that contains a unique, positive integer value representing the identifier for a frequency-based scheduler. You can obtain this value by making a call to **fbsconfigure** (see page 7-6 for an explanation of this routine). If you wish to reference the frequency-based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of **-1**.

info_buf — refers to a fbsinfo_cluster_ds structure to which fbsinfo_cluster will return information about the specified scheduler. The information returned in each component of the structure is presented in Table 7-7.

Table 7-7. Contents of Structure Components: fbsinfo_cluster

Component	Contents
sbc_id_location	SBC board ID where device actually resides
sbc_id_attached_mask	SBC board ID mask of those SBCs that contain schedulers that are currently attached to this timing device
filler	reserved for future use

devname — refers to a variable to which fbsinfo_cluster will return the path name of the actual device that is being used as the timing source on the SBC board where the device resides

Return Value

A return value of **0** indicates that the call has been successful. A return value of **-1** indicates that an error has occurred: **`errno`** is set to indicate the error. Refer to the system manual page **`fbsinfo_cluster(3rt)`** for a listing of the type of errors that may occur.

Fbsintrpt – Start/Stop/Resume Scheduling on an FBS

This routine is invoked to start, stop, or resume scheduling on a frequency–based scheduler. If you invoke this routine to start scheduling, the minor cycle, major frame, and overrun count values are reset. If you invoke it to resume scheduling, these values are not reset.

Prior to invoking **`fbsintrpt`**, you must have invoked **`fbsattach`** to specify end–of–cycle scheduling or attach a timing source to the frequency–based scheduler on which you are starting scheduling (see page 7-4 for an explanation of **`fbsattach`**). If you have specified a real–time clock as the timing source, scheduling will not begin until you have set and started the clock (see pages 7-29 and 7-26 for explanations of **`fbssetrtc`** and **`fbsrunrtc`**, respectively). If you have specified an edge–triggered interrupt device or a user–supplied device as the timing source, it must already be generating interrupts in order for scheduling to start.

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int fbsintrpt(fbs_id, intrflag)
int fbs_id;
int intrflag;
```

Call

```
int istat;
istat = fbsintrpt(fbs_id, intrflag);
```

Parameters

Parameters are described as follows.

fbs_id — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which you wish to start, stop, or resume scheduling of processes. You can obtain this value by making a call to **`fbsconfigure`** (see page 7-6 for an explanation of this routine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **`1`**.

intrflag — refers to a variable that contains an integer value indicating whether scheduling of processes on the specified scheduler is to be started, stopped, or resumed. Acceptable values and corresponding results are presented in Table 7-8.

Table 7-8. Intrflag Options

Value	Result
<0	Start scheduling of processes with the initial frame, cycle, and overrun count values set to zero
0	Stop scheduling of processes, and save the count values for the current frame and cycle
>0	Resume scheduling of processes with the frame, cycle, and overrun count values set to the values that were saved when the scheduler was last stopped

Return Value

A return value of `0` indicates that the call has been successful. A return value of – `1` indicates that an error has occurred; **`errno`** is set to indicate the error. Refer to the system manual page **`fbsintrpt(3rt)`** for a listing of the types of errors that may occur.

Fbsquery – Query Processes on an FBS

CAUTION

This interface is obsolete. It is maintained for compatibility with CX/UX, but it returns processes' scheduling priorities without any indication of the scheduling policies with which they are associated. If you have an existing application that uses this interface, it is recommended that you change your application to use **`sched_fbsqry(3rt)`** (see page 7-49). For details on obsolete interfaces, refer to Chapter 2, "Overview of the FBS."

This routine is invoked to obtain information about processes that have been scheduled on a frequency–based scheduler. Information is returned for all processes scheduled on the user–specified processor(s). Information provided for each process includes the following:

- A mask of the CPU(s) on which the process can execute
- The frequency–based scheduler process identifier
- The scheduling priority
- The period (the number of minor cycles indicating the frequency with which the process is wakened in each major frame)

- The starting base cycle (the first minor cycle in which the process is scheduled to be wakened in each major frame)
- The value of the “halt on overrun” flag
- The current state of the process

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int fbsquery(fbs_id, cpu, fbs_buf, buf_cnt)
int fbs_id;
int cpu;
struct pgm_ds {
   char *name_ptr;
   int  cpu;
   int  fpid;
   int  prior;
   int  param;
   int  period;
   int  cycle;
   int  halt;
   int  status;
} *fbs_buf;
int buf_cnt;
```

Call

```
struct pgm_ds  fbs_buf[buf_cnt];
int istat;
istat = fbsquery(fbs_id, cpu, fbs_buf, buf_cnt);
```

Parameters

Parameters are described as follows.

fbs_id	refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler for which you wish to obtain scheduling information. You can obtain this value by making a call to **`fbsconfigure`** (see page 7-6 for an explanation of this routine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **`1`**.
cpu	refers to a variable that contains an integer value indicating the processor(s) for which scheduling information is to be obtained. Acceptable values and corresponding results are presented in Table 7-9.

Table 7-9. CPU Options: fbsquery

Value	Result
0	Scheduling information for processes executing on the processor from which the call is made is returned
–1	Scheduling information for all processes on the scheduler is returned
Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU), scheduling information for processes executing on CPU *i* is returned

fbs_buf — refers to an array of **`pgm_ds`** structures to which **`fbsquery`** will return scheduling information for each process on the processor(s) specified with the *cpu* parameter. The type of information returned in each component of the structure for a single process is presented in Table 7-10.

Table 7-10. Contents of Structure Components: fbsquery

Component	Contents
name_ptr	A pointer to a variable that contains a standard UNIX path name identifying the process for which information is returned.
cpu	A bit mask indicating the processor(s) on which the process can execute
fpid	The process's frequency–based scheduler process identifier
prior	The process's scheduling priority
param	The process's initiation parameter
period	The number of minor cycles indicating the frequency with which the process is to be wakened in each major frame (period)
cycle	The first minor cycle in which the process is scheduled to be wakened in each major frame (starting base cycle)
halt	The value of the "halt on overrun" flag. A nonzero value indicates that the flag is set. A value of zero indicates that the flag is not set.
status	The current state of the process as defined in **`<fbslib.h>`**.

buf_cnt — refers to a variable that contains an integer value indicating the number of structures in the array to which **`fbs_buf`** points.

Return Value

A return value of **`0`** indicates that the call has been successful. A return value of – **`1`** indicates that an error has occurred; **`errno`** is set to indicate the error. Refer to the system manual page **`fbsquery(3rt)`** for a listing of the types of errors that may occur.

Fbsremove – Remove an FBS

This routine is invoked to remove a frequency–based scheduler and to free the data structure associated with it. It is important to note that prior to invoking **`fbsremove`**, you must ensure that the timing source is detached from the scheduler or that end–of–cycle scheduling is disabled (see page 7-10 for information on the use of **`fbsdetach`**). It is important to note that **`fbsremove`** will remove all processes scheduled on the specified scheduler. It is recommended, however, that you remove all scheduled processes prior to invoking **`fbsremove`**. You can do so by making a call to **`pgmremove`** (see page 7-39 for information on the use of this routine).

Note that to remove a frequency-based scheduler, the calling process must have the P_OWNER privilege or an effective user ID that is equal to that of the owner/creator of the scheduler.

If the Enhanced Security Utilities are installed and running, the following conditions must also be met:

- The calling process and the frequency-based scheduler must have identical security levels, or the process must have both P_MACREAD and P_MACWRITE privileges.

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int fbsremove(fbs_id, ab)
int fbs_id;
int ab;
```

Call

```
int istat;
istat = fbsremove(fbs_id, ab);
```

Parameters

Parameters are described as follows.

fbs_id — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler that you wish to remove. You can obtain this value by making a call to **`fbsconfigure`** (see page 7-6 for an explanation of this routine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

ab refers to a variable that contains an integer value indicating the manner in which processes scheduled on the scheduler are to be handled. Acceptable values and corresponding results are presented in Table 7-11.

Table 7-11. Ab Options

Value	Result
<0	Kill and remove all processes currently scheduled on the specified scheduler
≥0	Remove all processes currently scheduled on the specified scheduler

Return Value

A return value of **0** indicates that the call has been successful. A return value of – **1** indicates that an error has occurred; **errno** is set to indicate the error. Refer to the system manual page **fbsremove(3rt)** for a listing of the types of errors that may occur.

Fbsresume – Resume Scheduling on an FBS

The **fbsresume** library routine is invoked to resume scheduling of processes on a frequency-based scheduler at the specified minor cycle, major frame, and overrun count.

Note that to resume scheduling of processes on a frequency-based scheduler, the calling process must have alter permission for the scheduler. If the Enhanced Security Utilities are installed and running, the following conditions must also be met:

- The calling process and the frequency-based scheduler must have identical security levels, or the process must have the P_MACWRITE privilege.

If you wish to resume scheduling of processes on a frequency-based scheduler without altering the scheduler's current frame, cycle, and overrun values, it is recommended that you use the **fbsintrpt(3rt)** routine (see page 7-19 for an explanation of this routine).

CAUTION

The **fbsresume** routine clears performance monitor values for all processes scheduled on the specified scheduler. Changing the frame and cycle count for the scheduler causes the values that are being maintained by the performance monitor to be inaccurate.

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int fbsresume(fbs_id, frame, cycle, overruns)
int fbs_id;
int frame;
int cycle;
int overruns;
```

Call

```
int istat;
istat = fbsresume(fbs_id, frame, cycle, overruns);
```

Parameters

Parameters are described as follows.

fbs_id
: refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which you wish to resume scheduling of processes. You can obtain this value by making a call to **`fbsconfigure`** or **`fbsid`** (see page 7-6 and page 7-12, respectively, for explanations of these routines). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing the identifier, you can specify the value –1.

frame
: an integer value indicating the major frame in which you wish scheduling of processes to be resumed on the specified scheduler

cycle
: an integer value indicating the minor cycle in which you wish scheduling of processes to be resumed on the specified scheduler. This value can range from zero to the total number of minor cycles per frame minus one. The total number of minor cycles per frame was specified when the scheduler was created by making a call to **`fbsconfigure`** (see page 7-6 for an explanation of this routine).

overruns
: an integer value indicating the value to which you wish the overrun count to be set when scheduling resumes on the specified scheduler

  If you do not wish to change the overrun count, you can specify the value –1.

Return Value

A return value of **`0`** indicates that the call has been successful. A return value of –1 indicates that an error has occurred; **`errno`** is set to indicate the error. Refer to the system manual page **`fbsresume(3rt)`** for a listing of the types of errors that may occur.

Fbsrunrtc – Start/Stop Real–Time Clock

This routine is invoked to start or stop the counting of a real–time clock that has been attached to a frequency–based scheduler.

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int fbsrunrtc(fbs_id, runflag)
int fbs_id;
int runflag;
```

Call

```
int istat;
istat = fbsrunrtc(fbs_id, runflag);
```

Parameters

Parameters are described as follows.

fbs_id — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler for which you wish to start or stop the attached real–time clock. You can obtain this value by making a call to **fbsconfigure** (see page 7-6 for an explanation of this routine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

runflag — refers to a variable that contains an integer value indicating whether the real–time clock is to be started or stopped. A nonzero value indicates that the clock is to be started. A zero value indicates that the clock is to be stopped.

Return Value

A return value of **0** indicates that the call has been successful. A return value of – **1** indicates that an error has occurred; **errno** is set to indicate the error. Refer to the system manual page **fbsrunrtc(3rt)** for a listing of the types of errors that may occur.

Fbsschedself – Schedule an LWP on an FBS

The **fbsschedself** library routine is invoked to schedule the calling lightweight process (LWP) on a frequency-based scheduler.

This routine is designed to be used by a single-threaded or a multithreaded application; however, if it is to be used in a multithreaded application, it can be used only by bound threads.

It is important to note that **fbsschedself** does not allow an LWP to set its scheduling policy and priority or its CPU bias. These tasks must be performed prior to invoking **fbsschedself**.

A single-threaded process can set its scheduling policy and priority by using the **sched_setscheduler(3C)** library routine; it can set its CPU bias by using the **cpu_bias(2)** system call or the **mpadvise(3C**) library routine. Procedures for using these functions are explained in the "Process Scheduling and Management" and "Process Management" chapters of the *PowerMAX OS Programming Guide*.

A bound thread can set its scheduling policy and priority by using the **thr_setscheduler(3thread)** library routine; it can set its CPU bias by using the **cpu_bias** system call or the **mpadvise** library routine. Complete information on bound thread scheduling and use of the **thr_setscheduler** routine are provided in the "Thread Scheduling" section of the "Programming with the Threads Library" chapter of the *PowerMAX OS Programming Guide*.

Note that you cannot use this routine to add **/idle** or **/spare** to a frequency-based scheduler.

To schedule the calling LWP on a frequency-based scheduler, the calling LWP must have alter permission for the scheduler. If the Enhanced Security Utilities are installed and running, the following conditions must also be met:

- The calling LWP and the frequency-based scheduler must have identical security levels, or the LWP must have the P_MACWRITE privilege.

You must <u>not</u> change the scheduling policy or priority of an LWP while it is scheduled on a scheduler by using **sched_setscheduler**, **thr_setscheduler**, or other program interfaces that allow you to change scheduling policy and priority. The frequency-based scheduler is not aware of changes in scheduling policy and priority that are made by using these interfaces.

If you need to change the scheduling policy or priority of a single-threaded FBS-scheduled process, you may do so by using **sched_pgmresched** to reschedule it (see page 7-60 for an explanation of this routine).

If you need to change the scheduling policy or priority of a bound thread, you must first remove it from the scheduler on which it is has been scheduled by using **pgmremove** (see page 7-39 for an explanation of this routine). You can then use **thr_setscheduler** to change its policy or priority and **fbsschedself** to schedule it on a scheduler.

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int fbsschedself(fbs_id, name, sched_buf)
int fbs_id;
char *name;
```

```
struct fbssched_buf {
    int  version;
    int  param;
    int  period;
    int cycle;
    int  ab;
    int  fpid;
} *sched_buf;
```

Call

```
struct fbssched_buf  sched_buf;
int istat;
istat = fbsschedself(fbs_id, name, &sched_buf);
```

Parameters

Parameters are described as follows.

fbs_id
: refers to a variable that contains a unique, positive integer value representing the identifier for a frequency–based scheduler. You can obtain this value by making a call to **fbsconfigure** or **fbsid** (see page 7-6 and page 7-12, respectively, for explanations of these routines). If you wish to reference the frequency–based scheduler on which the calling LWP is scheduled without knowing the identifier, you can specify the value **–1**.

name
: a pointer to a variable that contains a standard UNIX path name or arbitrary content identifying the program associated with the calling LWP. A full or relative path name of up to 1023 characters can be specified.

sched_buf
: refers to a *sched_buf* structure that contains the scheduling parameters with which you wish to schedule the LWP. The type of information that is specified in each component is presented in Table 7-12.

Table 7-12. Contents of Structure Components: fbsschedself

Component	Contents
version	an integer value indicating the version of *sched_buf* that is being passed to **fbsschedself**. Specify the symbolic constant **FBSSCHED_BUF_V1**, which is defined in **<fbslib.h>** for this purpose.
param	an integer value to be passed to a process that is scheduled on a frequency-based scheduler. This value can be retrieved by the FBS–scheduled process through a call to **sched_pgmqry** (see page 7-57 for an explanation of this routine).

Table 7-12. Contents of Structure Components: fbsschedself (Cont.)

Component	Contents
period	an integer value indicating the frequency with which the calling LWP is to be wakened in each major frame. A period of one indicates that the calling LWP is to be wakened every minor cycle; a period of two indicates that it is to be wakened once every two minor cycles, a period of three once every three minor cycles, and so on. This value can range from one to the number of minor cycles that compose a frame on the specified scheduler as defined in a call to **fbsconfigure** (see page 7-6 for an explanation of this routine).
cycle	an integer value indicating the first minor cycle in which the calling LWP is scheduled to be wakened in each frame. This value can range from zero to the total number of minor cycles per frame minus one. The total number of minor cycles per frame is specified in a call to **fbsconfigure** (see page 7-6 for an explanation of this routine).
ab	an integer value indicating whether or not the scheduler should be stopped in the event that the calling LWP causes a frame overrun. A nonzero value indicates that the scheduler will be stopped.
fpid	an integer value that is returned by **fbsschedself** and is the unique frequency–based scheduler process identifier for the scheduled LWP

Return Value

A return value of **0** indicates that the call has been successful. A return value of −1 indicates that an error has occurred; **errno** is set to indicate the error. Refer to the system manual page **fbsschedself(3rt)** for a listing of the types of errors that may occur.

Fbssetrtc – Set Real–Time Clock

This routine is invoked to establish the duration of a minor cycle by setting the count and the resolution values for a real–time clock.

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int fbssetrtc(fbs_id, count, resolution)
int fbs_id;
int count;
int resolution;
```

Call

int istat;
istat = fbssetrtc(fbs_id, count, resolution);

Parameters

Parameters are described as follows.

fbs_id	refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler to which a real–time clock has been attached. You can obtain this value by making a call to **fbsconfigure** (see page 7-6 for an explanation of this routine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.
count	refers to a variable that contains an integer value indicating the number of clock counts per minor cycle. This value can range from one to 65535.
resolution	refers to a variable that contains an integer value indicating the duration in microseconds of one clock count. This value must be one of the following: 1, 10, 100, 1000, or 10000.

Return Value

A return value of **0** indicates that the call has been successful. A return value of – **1** indicates that an error has occurred; **errno** is set to indicate the error. Refer to the system manual page **fbssetrtc(3rt)** for a listing of the types of errors that may occur.

Fbswait – Wait on an FBS

NOTE

There is no C interface routine for **fbswait**; the **fbswait(2)** system call should be used instead.

Fbswait enables a process that is scheduled on a frequency–based scheduler to sleep until its next scheduled minor cycle.

The C specification, call, and return value are presented in the following sections.

Specification

int fbswait()

Call

int istat;
istat = fbswait();

Return Value

A return value of **0** indicates that the process has been wakened by the frequency–based scheduler. A return value of **1** indicates that the process has been wakened by **fbstrig(2)**. A return value of –**1** indicates that an error has occurred; **errno** is set to indicate the error. A return value of **2** indicates that the process did not sleep because the kernel detected a soft overrun and is allowing the process to attempt to recover from it. Refer to the system manual page **fbswait(2)** for a listing of the types of errors that may occur.

Fbs_register_rdev - Register Coupled FBS Timing Device

This routine may be used to register a local device as a remote timing device (**rdevfs(4)**) which may be subsequently used as a Coupled FBS timing device. A Coupled timing device may be used to couple together FBS schedulers that are located on more than one computer system. All schedulers that are attached to the same Coupled FBS timing device will start, stop and resume their executions together on the same frame and cycle, using the Coupled FBS timing device as the interrupt source.

To register a timing device, the calling process must have the P_RTIME privilege as well as enough privilege to open the device file.

Successfully registering a device as a Coupled FBS timing source creates a placeholder, or virtual FBS identifier to reserve the device's interrupt vector. There is one virtual FBS for each device registered and a virtual FBS provides the means for a process on another host to communicate with the real device. Because the virtual FBS is allocated exactly the same way as user FBS identifiers, each device registered reduces by one the number of user schedulers that can be configured. Therefore, depending upon system requirements, it may be necessary to increase the value of the system tunable parameter **FBSMNI**. Virtual FBS descriptors are not directly accessible to user programs.

Registering a device as a Coupled FBS timing device also creates a device file entry in the **/dev/rdev** file system on each host where the device is registered. This **/dev/rdev/<hostname/device<*n*>** path name may be specified on subsequent calls to **fbsattach**. A device may not be registered as a Coupled FBS timing device if a FBS scheduler is already directly attached to that device. The C specification, call, corresponding parameters and return value are presented in the following sections.

Specification

```
#include <fbslib.h>
int fbs_register_rdev(device_name, rdevfs_name,
type, num_hosts, hostname_array)
char *device_name;
char *rdevfs_name;
int type;
int num_hosts;
char **hostname_array;
```

Call

int status, types, num_hosts;
char *device_name, *rdevfs_name;
char **hostname_array;
status = fbs_register_rdev(device_name, rdevfs_name,
type, num_hosts, hostname_array);

Parameters

The parameters are described as follows.

device_name
: Refers to a variable that contains the user-specified path name of the device that is to be registered as a Coupled FBS timing device.

  If the device is a real-time clock or edge triggered interrupt, then the path name must be of a certain form. See Chapter 3 for detailed information on these types of path names.

  If the device is user-supplied device, the path name must be a valid UNIX path name, and the device must support the IOCTLVECNUM **ioctl(2)** call. See Chapter 3 for additional information.

rdevfs_name
: Refers to the location where the corresponding **/dev/rdev/<hostname>/device<*n*> rdevfs** file system device file entry will be returned. This path name should be used on subsequent fbsattach calls for attaching FBS schedulers to this Coupled FBS timing device.

type
: In this field, the caller specifies the type of timing device that is to be registered. When type is set to FBS_RC_TYPE, then a RCIM Coupled timing device will be registered. When type is set to FBS_CC _TYPE, then a Closely-Coupled timing device is to be registered. See Chapter 3 or the **fbs_register_rdev(3rt)** system manual page for details about these two types of timing devices.

num_hosts
: The caller specifies in this field the number of hosts where this device is to be registered. This value should match the number of hostnames in the **hostname_array**.

hostname_array
: This field contains a pointer to an array of character pointers, where each entry is a hostname string of a host where the device is to be registered. The local host's hostname must be specified in this list.

  Only those remote hosts that intend to attach a scheduler to this Coupled FBS timing device need to be in the **hostname_array**.

Return Value

A return value of **0** indicates that the call has been successful. A return value of **-1** indicates that an error has occurred. In this case, errno is set to a value that indicates the type of error. Refer to the system manual page **fbs_register_rdev(3rt)** for a listing of the types of errors that may occur, as well as other detailed usage information.

Fbs_unregister_rdev - Unregister a Coupled FBS Timing Device

This routine may be called to unregister a local device that was previously registered as a Coupled FBS timing device. To unregister a device, the calling process must have the **P_RTIME** privilege as well as enough privilege to open the device file.

Unregistering a device from being a Coupled FBS timing device results in the removal of the virtual FBS identifier that was created when the device was initially registered. The unregistration also removes the corresponding **/dev/rdev/<hostname>/device<n> rdevfs** file system device file entry on each host where the device was previously registered.

Once a device is unregistered, it may once again be directly attached to an FBS scheduler on the local system as a normal, non-Coupled FBS timing device, or, it may be re-registered as a Coupled FBS timing device with the **fbs_register_rdev(3rt)** function.

The C specification, call, corresponding parameters and return value are presented in the following sections.

Specification

```
#include <fbslib.h>
int fbs_unregister_rdev(device_name)
char *device_name;
```

Call

```
int status;
status = fbs_unregister_rdev(device_name)
```

Parameters

Parameters are described as follows.

device_name	Refers to a variable that contains the path name of the device that is to be unregistered from being a Coupled FBS timing device. The path name should be the same path name that was specified on a previous **fbs_register_rdev** function call.

Return Value

A return value of **0** indicates that the call has been successful. A return value of **-1** indicates that an error has occurred. In this case, errno is set to an appropriate value to indicate the type of error. Refer to the system manual page **fbs_unregister_rdev(3rt)** for a listing of the types of errors that may occur.

Fbs_register_cluster_device - Register Cluster Timing Source

This routine is invoked to register a local device as a Closely-Coupled timing device in a Closely-Coupled system. To register a device, the calling process must have the P_RTIME privilege as well as enough privilege to open the device file.

Registering a Closely-Coupled timing device creates a placeholder, or virtual, FBS identifier to reserve the device's interrupt vector. There is one virtual FBS for each device registered and a virtual FBS provides the means for a process on another SBC to communicate with the real device. Because the virtual FBS is allocated exactly the same way as user FBS identifiers, each device registered reduces by one the number of user schedulers that can be configured. Therefore, some thought should be given to increasing the value of the system tunable parameter **FBSMNI**. Virtual FBS descriptors are not directly accessible to user programs.

Registering a device as a Closely-Coupled timing source also creates entries in the **`/dev/rdev`** directories on all SBCs in the VME cluster. These entries can be specified on a subsequent call to **`fbsattach`**.

A device can either be registered as a Closely-Coupled timing device or be attached to an FBS, but not both at the same time.

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int fbs_register_cluster_device(device_name, rdevfs_name)
char *device_name;
char *rdevfs_name;
```

Call

```
int istat;
istat = fbs_register_cluster_device(device_name, rdevfs_name);
```

Parameters

Parameters are described as follows.

device_name
: Refers to a variable that contains the path name of the device that is to be registered as a Closely-Coupled timing device. device_name may refer to a real-time clock, edge-triggered interrupt or to a user-suppled device.

  If the device is a real-time clock or edge-triggered interrupt, then refer to Chapter 3 for detailed information about these pathnames and their associated attributes.

  If the device is a user-supplied device, the path name must be a valid UNIX path name and the device must support the **`IOCTLVECNUM ioctl(2)`** call. See Chapter 3 for additional information.

rdevfs_name
: Refers to the location where the corresponding **`rdvfs_dev`** device file entry path name will be returned. This returned path name should be used on subsequent fbsattach calls to attach schedulers to this Closely-Coupled timing device

Return Value

A return value of **0** indicates that the call has been successful. A return value of **-1** indicates that an error has occurred; **`errno`** is set to indicate the error. Refer to the system manual page **`fbs_register_cluster_device(3rt)`** for a listing of the types of errors that may occur.

NOTE

> The **fbs_register_cluster_device** function call is obsolete. It is being supported only for providing backward compatibility with previous PowerMAX OS releases. Users are highly encouraged to make use of the newer **fbs_register_rdev** function call. Note that **fbs_register_cluster_device** only supports the registration of Closely-Coupled timing devices, while the **fbs_register_rdev** function supports both Closely-Coupled and RCIM Coupled timing device registrations.

Fbs_unregister_cluster_device - Unregister Cluster Timing Source

This routine is invoked to unregister a local device as a Closely-Coupled timing device in a Closely-Coupled system. To unregister a device, the calling process must have the **`P_RTIME`** privilege as well as enough privilege to open the device file.

Unregistering a device as a Closely-Coupled timing device removes the virtual FBS identifier created when the device was registered and also removes the **`/dev/rdev`** entries on all SBCs in the VME cluster. Once a device is unregistered, it is once again available to be attached to an FBS on the local SBC.

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int fbs_unregister_cluster_device(device_name)
char *device_name;
```

Call

```
int istat;
istat = fbs_unregister_cluster_device(device_name);
```

Parameters

Parameters are described as follows.

device_name	Refers to a variable that contains the path name of the device that is to be unregistered as a Closely-Coupled timing device.

The device name should be the same path name that was previously specified on the corresponding fbs_register_cluster_device function call.

If the device is a real-time clock, the path name must be of a certain form. Refer to Chapter 3 for detailed information on the form associated with the real-time clock.

If the device is a user-suppled device, the path name must be a valid UNIX path name. The device must support the **IOCTLVECNUM ioctl(2)** call. See Chapter 3 for additional information.

Return Value

A return value of **0** indicates that the call has been successful. A return value of **-1** indicates that an error has occurred; **errno** is set to indicate the error. Refer to the system manual page **fbs_unregister_cluster_device(3rt)** for a listing of the types of errors that may occur.

NOTE

The **fbs_unregister_cluster_device** function call is obsolete. It is being supported only for providing backward compatibility with previous PowerMAX OS releases. Users are highly encouraged to make use of the newer **fbs_unregister_rdev** function call.

Pgmquery – Query a Process on an FBS

CAUTION

This interface is obsolete. It is maintained for compatibility with CX/UX, but it returns a process's scheduling priority without any indication of the scheduling policy with which that priority is associated. If you have an existing application that uses this interface, it is recommended that you change your application to use **sched_pgmqry(3rt)** (see page 7-57). For details on obsolete interfaces, refer to Chapter 2, "Overview of the FBS."

This routine is invoked to obtain information for a particular process that has been scheduled on a frequency–based scheduler. You can identify the process by using one of the following methods:

- Specify the name of the process and the CPU(s) on which it is scheduled.
- Specify the process's frequency–based scheduler process identifier (slot number).
- Specify the name of the process, the CPU on which it is scheduled, and its frequency–based scheduler process identifier.

NOTE

The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency–based scheduler process identifier.

Information that is returned includes the following:

- The process's path name
- The CPU on which the process can execute
- The frequency–based scheduler process identifier
- The scheduling priority
- The period (the number of minor cycles indicating the frequency with which the process is wakened in each major frame)
- The starting base cycle (the first minor cycle in which the process is scheduled to be wakened in each major frame)
- The value of the "halt on overrun" flag

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int pgmquery(fbs_id, qry_buf)
int fbs_id;
struct pgm_ds {
   char *name_ptr;
   int cpu;
   int fpid;
   int prior;
   int param;
   int period;
   int cycle;
   int halt;
   int status;
} *qry_buf;
```

Call

```
struct pgm_ds  qry_buf;
int istat;
istat = pgmquery(fbs_id, &qry_buf);
```

Parameters

Parameters are described as follows.

fbs_id — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process for which you wish to obtain scheduling information has been scheduled. You can obtain this value by making a call to **fbsconfigure** (see page 7-6 for an explanation of this routine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

qry_buf — refers to a **pgm_ds** structure that contains information identifying the process for which information is to be returned. **Pgmquery** will return to this structure the scheduling information for a specified process. The information contained in each component of the structure to which **qry_buf** points is presented in Table 7-13.

Table 7-13. Contents of Structure Components: pgmquery

Component	Contents	
name_ptr	a pointer to a variable that contains a standard UNIX path name identifying the process for which information is to be returned. A full or relative path name of up to 1024 characters can be specified. If the pointer points to a null string, you must provide the frequency–based scheduler process identifier in the **fpid** component.	
cpu	An integer value indicating the processor(s) to be used in conjunction with the value of *name_ptr* to identify the program for which information is to be obtained. Acceptable values and corresponding results follow:	
	0	The first process whose name matches the name pointed to by *name_ptr* that is currently running on the processor from which the call is made is specified
	–1	The first process whose name matches the name pointed to by *name_ptr* that is currently running on any processor is specified

	Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process whose name matches the name pointed to by *name_ptr* that is running on CPU *i* is specified
		
		If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process whose name matches the name pointed to by *name_ptr* that is currently running on any of the selected CPUs is specified

Table 7-13. Contents of Structure Components: pgmquery (Cont.)

Component	Contents
fpid	an integer value providing the unique frequency–based scheduler process identifier for the process for which information is to be returned. This value is obtained when you make a call to **pgmschedule** (see page 7-45 for an explanation of this routine). This value must be –1 if you wish to identify the program to be queried only by specifying *name_ptr* and *cpu*.
prior	an integer value indicating the specified process's scheduling priority
param	an integer value indicating the value passed to the process via a call to **pgmschedule** or **pgmreschedule**
period	an integer value indicating the frequency with which the specified program is to be wakened in each major frame.
cycle	an integer value indicating the first minor cycle in which the specified process is scheduled to be wakened in each frame
halt	an integer value indicating the value of the "halt on overrun" flag. A nonzero value indicates that the flag is set. A value of zero indicates that the flag is not set.
status	an integer value indicating the current state of the specified process as defined in **<fbslib.h>**

Return Value

A return value of **0** indicates that the call has been successful. A return value of – **1** indicates that an error has occurred; **errno** is set to indicate the error. Refer to the system manual page **pgmquery(3rt)** for a listing of the types of errors that may occur.

Pgmremove – Remove a Process from an FBS

This routine is invoked to remove a process from a frequency–based scheduler. You can identify the process that you wish to remove by using one of the following methods:

- Specify the name of the process and the CPU on which it is scheduled.
- Specify the process's frequency–based scheduler process identifier (slot number).
- Specify the name of the process, the CPU on which it is scheduled, and its frequency–based scheduler process identifier.

NOTE

The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency–based scheduler process identifier.

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int pgmremove(fbs_id, name, cpu, fpid, ab)
int  fbs_id;
char *name;
int  cpu;
int  fpid;
int  ab;
```

Call

```
int istat;
istat = pgmremove(fbs_id, name, cpu, fpid, ab);
```

Parameters

Parameters are described as follows.

fbs_id
: refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process is scheduled. You can obtain this value by making a call to **`fbsconfigure`** (see page 7-6 for an explanation of this routine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **`1`**.

name
: refers to a variable that contains a standard UNIX path name identifying the process to be removed from the specified scheduler. A full or relative path name of up to 1024 characters can be specified. If this variable contains the null string, you must provide the frequency–based scheduler process identifier in the *slot* parameter.

cpu
: refers to a variable that contains an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the process to be removed from the specified scheduler. Acceptable values and corresponding results are presented in Table 7-14.

fpid
: refers to a variable that contains an integer value providing the unique frequency–based scheduler process identifier for the process to be removed from the specified scheduler. This value is obtained when you make a call to **`sched_pgmadd`** (see page 7-52 for an explanation of this routine). This value must be – **`1`** if you choose to identify the program to be removed only by specifying *name* and *cpu*.

ab
: refers to a flag that contains an integer value indicating the manner in which the specified process is be removed from

Table 7-14. CPU Options: pgmremove

Value	Result
0	The first process named by *name* that is currently running on the processor from which the call is made is removed
–1	The first process named by *name* that is currently running on any processor is removed

Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is running on CPU *i* is removed
	
	If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is removed

the specified scheduler. A positive value indicates that the process is to be removed from the scheduler but allowed to continue executing. A negative value indicates that the process is to be removed from the scheduler and terminated.

Return Value

A return value of **0** indicates that the call has been successful. A return value of **– 1** indicates that an error has occurred; **errno** is set to indicate the error. Refer to the system manual page **pgmremove(3rt)** for a listing of the types of errors that may occur.

Pgmreschedule – Reschedule a Process

CAUTION

This interface is obsolete. It is maintained for compatibility with CX/UX, but its behavior with respect to specification of a process's scheduling priority has changed. If you have an existing application that uses this interface, it is recommended that you change your application to use **sched_pgmresched(3rt)** (see page 7-60). For details on obsolete interfaces, refer to Chapter 2, "Overview of the FBS."

This routine is invoked to change the scheduling parameters for a process that is scheduled on a frequency–based scheduler. You may wish, for example, to change a program's priority or the frequency with which it is scheduled to run. You cannot, however, change the CPU on which it has been scheduled.

To change a process's priority, the following conditions must be met:

- The calling process must have the P_RTIME privilege.

- The effective user ID of the calling process must match the effective user ID of the target process (the process for which the scheduling policy and priority are being set), or the calling process must have the P_OWNER privilege.

If the Enhanced Security Utilities are installed and running, the following additional conditions must be met:

- The Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

For additional information on privileges, refer to the *PowerMAX OS Programming Guide* and the **intro(2)** system manual page.

You can call **pgmreschedule** to change the parameters without having called **pgmremove** to remove the process from the scheduler (see page 7-39) or **fbsintrpt** to stop the simulation (see page 7-19).

You can identify the process that you wish to reschedule by using one of the following methods:

- Specify the name of the process and the CPU on which it is scheduled.
- Specify the process's frequency–based scheduler process identifier (slot number).
- Specify the name of the process, the CPU on which it is scheduled, and its frequency–based scheduler process identifier.

NOTE

> The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency–based scheduler process identifier.

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int pgmreschedule(fbs_id, rsch_buf)
int fbs_id;
struct pgm_ds {
   char *name_ptr;
   int  cpu;
   int  fpid;
   int  prior;
   int  param;
   int  period;
   int  cycle;
   int  halt;
```

```
   int  status;
} *rsch_buf;
```

Call

```
struct pgm_ds  rsch_buf;
int istat;
istat = pgmreschedule(fbs_id, &rsch_buf);
```

Parameters

Parameters are described as follows.

fbs_id	refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process is scheduled. You can obtain this value by making a call to **fbsconfigure** (see page 7-6 for an explanation of this routine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.
rsch_buf	refers to a **pgm_ds** structure that contains the scheduling parameters with which you wish to reschedule the process. The type of information that is specified in each component is presented in Table 7-15. Note that the **status** component is ignored on this call.

Table 7-15. Contents of Structure Components: pgmreschedule

Component	Contents	
name_ptr	a pointer to a variable that contains a standard UNIX path name identifying the process for which information is to be rescheduled. A full or relative path name of up to 1024 characters can be specified. If the pointer points to a null string, you must provide the frequency–based scheduler process identifier in the **fpid** component.	
cpu	An integer value indicating the processor(s) to be used in conjunction with the value of *name_ptr* to identify the process to be rescheduled. Acceptable values and corresponding results are as follows:	
	0	The first process whose name matches the name pointed to by *name_ptr* that is currently running on the processor from which the call is made is rescheduled
	–1	The first process whose name matches the name pointed to by *name_ptr* that is currently running on any processor is rescheduled

Table 7-15. Contents of Structure Components: pgmreschedule (Cont.)

Component	Contents	

	Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process whose name matches the name pointed to by *name_ptr* that is running on CPU *i* is rescheduled
		
		If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process whose name matches the name pointed to by *name_ptr* that is currently running on any of the selected CPUs is rescheduled

fpid	an integer value providing the unique frequency–based scheduler process identifier for the process to be rescheduled. This value is obtained when you make a call to **pgmschedule** (see page 7-45 for an explanation of this routine). This value must be – **1** if you wish to identify the program to be rescheduled only by specifying *name_ptr* and *cpu*.	
prior	an integer value indicating the specified process's scheduling priority. A process that has been scheduled using **pgmschedule** (see page 7-45 for an explanation of this routine) is scheduled under the POSIX **SCHED_RR** scheduling policy. The value specified must lie in the range of priorities associated with this policy. You can obtain the allowable range of priorities by invoking the **run(1)** command from the shell and not specifying any options or arguments (see the corresponding system manual page for an explanation of this command). Higher numerical values correspond to more favorable scheduling priorities. For complete information on scheduling policies and priorities, refer to the "Process Scheduling and Management" chapter of the *PowerMAX OS Programming Guide*.	
param	an integer value to be passed to a process that is scheduled on a frequency–based scheduler	

Table 7-15. Contents of Structure Components: pgmreschedule (Cont.)

Component	Contents
period	an integer value indicating the frequency with which the specified program is to be wakened in each major frame. A period of one indicates that the specified program is to be wakened every minor cycle; a period of two indicates that it is to be wakened once every two minor cycles, a period of three once every three minor cycles, and so on. This value can range from one to the number of minor cycles that compose a frame on the specified scheduler as defined in a call to **fbsconfigure** (see page 7-6).
cycle	an integer value indicating the first minor cycle in which the specified process is scheduled to be wakened in each frame. This value can range from zero to the total number of minor cycles per frame minus one. The total number of minor cycles per frame is specified in a call to **fbsconfigure** (see page 7-6 for an explanation of this routine).
halt	an integer value indicating whether or not the scheduler should be stopped in the event that the specified process causes a frame overrun. A nonzero value indicates that the scheduler will be stopped.

Return Value

A return value of **0** indicates that the call has been successful. A return value of – **1** indicates that an error has occurred; **errno** is set to indicate the error. Refer to the system manual page **pgmreschedule(3rt)** for a listing of the types of errors that may occur.

Pgmschedule – Schedule a Process on an FBS

CAUTION

This interface is obsolete. It is maintained for compatibility with CX/UX, but its behavior with respect to specification of a process's scheduling priority has changed. If you have an existing application that uses this interface, it is recommended that you change your application to use **sched_pgmadd(3rt)** (see page 7-52). For details on obsolete interfaces, refer to Chapter 2, "Overview of the FBS."

This routine is invoked to create a new process and schedule it on a frequency–based scheduler. When a process is scheduled using this routine, it is scheduled under the POSIX **SCHED_RR** scheduling policy (for complete information on scheduling policies and priorities, refer to the "Process Scheduling and Management" chapter of the *PowerMAX OS Programming Guide)*. Note that a process can not be scheduled under this policy on a CPU on which servicing of the 60 Hz clock interrupt has been disabled. In such cases, the process will behave as though it were scheduled under the **SCHED_FIFO** policy.

If you wish to set the process's scheduling priority, the following conditions must be met:

- The calling process must have the P_RTIME privilege.
- The effective user ID of the calling process must match the effective user ID of the target process (the process for which the scheduling policy and priority are being set), or the calling process must have the P_OWNER privilege.

If the Enhanced Security Utilities are installed and running, the following additional conditions must be met:

- The Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

If you wish to modify the process's CPU bias when you invoke this routine, the following conditions must be met:

- The real or effective user ID of the calling process must match the real or saved user ID of the process for which the CPU assignment is being changed, or the calling process must have the P_OWNER privilege.
- To add a CPU to a process's CPU bias, the calling process must have the P_CPUBIAS privilege.

If the Enhanced Security Utilities are installed and running, the following additional conditions must be met:

- The Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

For additional information on privileges, refer to the *PowerMAX OS Programming Guide* and the **`intro(2)`** system manual page.

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int pgmschedule(fbs_id, sched_buf)
int fbs_id;
struct pgm_ds {
   char *name_ptr;
   int  cpu;
   int  fpid;
   int  prior;
   int  param;
   int  period;
   int  cycle;
   int  halt;
   int  status;
} *sched_buf;
```

Call

```
struct pgm_ds  sched_buf;
int istat;
istat = pgmschedule(fbs_id, &sched_buf);
```

Parameters

Parameters are described as follows.

fbs_id
: refers to a variable that contains a unique, positive integer value representing the identifier for a frequency–based scheduler. You can obtain this value by making a call to **`fbsconfigure`** (see page 7-6 for an explanation of this routine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing the identifier, you can specify the value – **`1`**.

sched_buf
: refers to a *sched_buf* structure that contains the scheduling parameters with which you wish to schedule the process. The type of information that is specified in each component is presented in Table 7-16. Note that the **`status`** component is ignored on this call.

Table 7-16. Contents of Structure Components: pgmschedule

Component	Contents	
name_ptr	a pointer to a variable that contains a standard UNIX path name identifying the program to be scheduled on the scheduler. A full or relative path name of up to 1024 characters can be specified.	
cpu	An integer value indicating the processors on which the specified program can be scheduled to run. Acceptable values and corresponding results are as follows:	
	0	The program pointed to by *name_ptr* can be scheduled on the processor from which the call is made
	-1	The program pointed to by *name_ptr* can be scheduled on any processor
	Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU), the program pointed to by *name_ptr* can be scheduled on CPU *i*
fpid	an integer value that is returned by **`pgmschedule`** and is the unique frequency–based scheduler process identifier for the scheduled process	

Table 7-16. Contents of Structure Components: pgmschedule (Cont.)

Component	Contents
prior	an integer value indicating the specified process's scheduling priority. A process that is scheduled using **pgmschedule** is scheduled under the POSIX **SCHED_RR** scheduling policy. The value specified must lie in the range of priorities associated with this policy. You can obtain the allowable range of priorities by invoking the **run(1)** command from the shell and not specifying any options or arguments (see the corresponding system manual page for an explanation of this command). Higher numerical values correspond to more favorable scheduling priorities. For complete information on scheduling policies and priorities, refer to the "Process Scheduling and Management" chapter of the *PowerMAX OS Programming Guide*.
param	an integer value to be passed to a process that is scheduled on a frequency–based scheduler. This value can be retrieved by the FBS–scheduled process through a call to **pgmquery** (see page 7-36 for an explanation of this routine).
period	an integer value indicating the frequency with which the specified program is to be wakened in each major frame. A period of one indicates that the specified program is to be wakened every minor cycle; a period of two indicates that it is to be wakened once every two minor cycles, a period of three once every three minor cycles, and so on. This value can range from one to the number of minor cycles that compose a frame on the specified scheduler as defined in a call to **fbsconfigure** (see page 7-6).
cycle	an integer value indicating the first minor cycle in which the specified program is scheduled to be wakened in each frame. This value can range from zero to the total number of minor cycles per frame minus one. The total number of minor cycles per frame is specified in a call to **fbsconfigure** (see page 7-6 for an explanation of this routine).
halt	an integer value indicating whether or not the scheduler should be stopped in the event that the specified program causes a frame overrun. A nonzero value indicates that the scheduler will be stopped.

Return Value

A return value of **0** indicates that the call has been successful. A return value of – **1** indicates that an error has occurred; **errno** is set to indicate the error. Refer to the system manual page **pgmschedule(3rt)** for a listing of the types of errors that may occur.

Pgmtrigger – Trigger Process Waiting on FBS

This routine enables a process to wake a process that is in the **fbswait** sleep state. It is important to note that the calling process does not have to be scheduled on a frequency–based scheduler; the target process must be.

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int pgmtrigger(fpid, tgrflg)
int fpid;
int tgrflg;
```

Call

```
int istat;
istat = pgmtrigger(fpid, tgrflg);
```

Parameters

Parameters are described as follows.

fpid	refers to a variable that contains an integer value providing the unique frequency–based scheduler process identifier for the sleeping process. This value is obtained when you make a call to **sched_pgmadd** (see page 7-52 for an explanation of this routine).
tgrflg	refers to a variable that contains an integer value indicating whether or not a context switch is to be forced on the processor on which the wakened process is executing. A non-zero value indicates that a context switch is to be forced.

Return Value

A return value of **0** indicates that the call has been successful. A return value of **–1** indicates that an error has occurred; **errno** is set to indicate the error. Refer to the system manual page **pgmtrigger(3rt)** for a listing of the types of errors that may occur.

Sched_fbsqry – Query Processes on an FBS

The **sched_fbsqry** routine is invoked to obtain information about processes that have been scheduled on a frequency–based scheduler. Information is returned for all processes scheduled on the user–specified processor(s). Information provided for each process includes the following:

- A mask of the CPU(s) on which the process can execute
- The frequency–based scheduler process identifier

- The policy under which the process has been scheduled
- The scheduling priority
- The period (the number of minor cycles indicating the frequency with which the process is wakened in each major frame)
- The starting base cycle (the first minor cycle in which the process is scheduled to be wakened in each major frame)
- The value of the "halt on overrun" flag
- The current state of the process

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int sched_fbsqry(fbs_id, cpu, fbs_buf, buf_cnt)
int fbs_id;
int cpu;
struct pgm2_ds {
   char *name_ptr;
   int  cpu;
   int  fpid;
   int  cid;
   int  prior;
   int  param;
   int  period;
   int  cycle;
   int  halt;
   int  status;
} *fbs_buf;
int buf_cnt;
```

Call

```
struct pgm2_ds  fbs_buf[buf_cnt];
int istat;
istat = sched_fbsqry(fbs_id, cpu, fbs_buf, buf_cnt);
```

Parameters

Parameters are described as follows.

fbs_id — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler for which you wish to obtain scheduling information. You can obtain this value by making a call to **fbsconfigure** (see page 7-6 for an explanation of this routine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

cpu — refers to a variable that contains an integer value indicating the processor(s) for which scheduling information is to be obtained. Acceptable values and corresponding results are presented in Table 7-17.

Table 7-17. CPU Options: sched_fbsqry

Value	Result
0	Scheduling information for processes executing on the processor from which the call is made is returned
-1	Scheduling information for all processes on the scheduler is returned
Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU), scheduling information for processes executing on CPU *i* is returned

fbs_buf — refers to an array of **`pgm2_ds`** structures to which **`sched_fbsqry`** will return scheduling information for each process on the processor(s) specified with the *cpu* parameter. The type of information returned in each component of the structure for a single process is presented in Table 7-18.

Table 7-18. Contents of Structure Components: sched_fbsqry

Component	Contents
name_ptr	A pointer to a variable that contains a standard UNIX path name identifying the process for which information is returned.
cpu	A bit mask indicating the processor(s) on which the process can execute
fpid	The process’s frequency–based scheduler process identifier
cid	The process’s scheduling policy
prior	an integer value indicating the specified process’s scheduling priority
param	The process’s initiation parameter
period	The number of minor cycles indicating the frequency with which the process is to be wakened in each major frame (period)

Table 7-18. Contents of Structure Components: sched_fbsqry (Cont.)

Component	Contents
cycle	The first minor cycle in which the process is scheduled to be wakened in each major frame (starting base cycle)
halt	The value of the "halt on overrun" flag. A nonzero value indicates that the flag is set. A value of zero indicates that the flag is not set.
status	The current state of the process as defined in **<fbslib.h>**.

buf_cnt — refers to a variable that contains an integer value indicating the number of structures in the array to which **fbs_buf** points.

Return Value

A return value of **0** indicates that the call has been successful. A return value of – **1** indicates that an error has occurred; **errno** is set to indicate the error. Refer to the system manual page **sched_fbsqry(3rt)** for a listing of the types of errors that may occur.

Sched_pgmadd – Schedule a Process on an FBS

The **sched_pgmadd** routine is invoked to create a new process and schedule it on a frequency–based scheduler. It is important to note that to use this routine to (1) change a process's scheduling policy to the **SCHED_FIFO** or the **SCHED_RR** policy or (2) change the priority of a process scheduled under the **SCHED_FIFO** or the **SCHED_RR** policy, the following conditions must be met:

- The calling process must have the P_RTIME privilege.
- The effective user ID of the calling process must match the effective user ID of the target process (the process for which the scheduling policy and priority are being set), or the calling process must have the P_OWNER privilege.

If the Enhanced Security Utilities are installed and running, the following additional conditions must be met:

- The Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

If you wish to raise the priority of a process scheduled under the **SCHED_OTHER** policy above a per-process or LWP limit, the following conditions must be met:

- The calling process must have the P_TSHAR privilege.

- The effective user ID of the calling process must match the effective user ID of the target process (the process for which the scheduling priority is being set), or the calling process must have the P_OWNER privilege.

If the Enhanced Security Utilities are installed and running, the following additional conditions must be met:

- The Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

If you wish to modify the process's CPU bias when you invoke this routine, the following conditions must be met:

- The real or effective user ID of the calling process must match the real or saved user ID of the process for which the CPU assignment is being changed, or the calling process must have the P_OWNER privilege.
- To add a CPU to a process's CPU bias, the calling process must have the P_CPUBIAS privilege.

If the Enhanced Security Utilities are installed and running, the following additional conditions must be met:

- The Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

For additional information on privileges, refer to the *PowerMAX OS Programming Guide* and the **intro(2)** system manual page.

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int sched_pgmadd(fbs_id, sched_buf)
int fbs_id;
struct pgm2_ds {
   char *name_ptr;
   int  cpu;
   int  fpid;
   int  cid;
   int  prior;
   int  param;
   int  period;
   int  cycle;
   int  halt;
   int  status;
} *sched_buf;
```

Call

```
struct pgm2_ds  sched_buf;
int istat;
istat = sched_pgmadd(fbs_id, &sched_buf);
```

Parameters

Parameters are described as follows.

fbs_id
: refers to a variable that contains a unique, positive integer value representing the identifier for a frequency–based scheduler. You can obtain this value by making a call to **fbsconfigure** (see page 7-6 for an explanation of this routine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing the identifier, you can specify the value – **1**.

sched_buf
: refers to a *sched_buf* structure that contains the scheduling parameters with which you wish to schedule the process. The type of information that is specified in each component is presented in Table 7-19. Note that the **status** component is ignored on this call.

Table 7-19. Contents of Structure Components: sched_pgmadd

Component	Contents
name_ptr	a pointer to a variable that contains a standard UNIX path name identifying the program to be scheduled on the scheduler. A full or relative path name of up to 1024 characters can be specified.
cpu	An integer value indicating the processors on which the specified program can be scheduled to run. Acceptable values and corresponding results are as follows: 0 — The program pointed to by *name_ptr* can be scheduled on the processor from which the call is made. -1 — The program pointed to by *name_ptr* can be scheduled on any processor Bit mask — If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU), the program pointed to by *name_ptr* can be scheduled on CPU *i*
fpid	an integer value that is returned by **sched_pgmadd** and is the unique frequency–based scheduler process identifier for the scheduled process

Table 7-19. Contents of Structure Components: sched_pgmadd (Cont.)

Component	Contents
cid	an integer value indicating the POSIX scheduling policy under which the specified program is to be scheduled. Scheduling policies are defined in the file <**`sched.h`**>. The value of *cid* must be one of the following:
	SCHED_FIFO — first–in–first out (FIFO) scheduling policy
	SCHED_RR — round–robin (RR) scheduling policy. Note that a process cannot be scheduled under this policy on a CPU on which servicing of the 60 Hz clock interrupt has been disabled. In such cases, the process behaves as though it were scheduled under the **SCHED_FIFO** policy
	SCHED_OTHER — time-sharing scheduling policy
prior	an integer value indicating the scheduling priority of the specified process. The range of acceptable priority values is governed by the scheduling policy specified. You can determine the allowable range of priorities associated with each policy (**SCHED_FIFO**, **SCHED_RR**, or **SCHED_OTHER**) by invoking the **`run(1)`** command from the shell and not specifying any options or arguments (see the corresponding system manual page for an explanation of this command). Higher numerical values correspond to more favorable priorities. For complete information on scheduling policies and priorities, refer to the "Process Scheduling and Management" chapter of the *PowerMAX OS Programming Guide*.
param	an integer value to be passed to a process that is scheduled on a frequency–based scheduler. This value can be retrieved by the FBS–scheduled process through a call to **`sched_pgmqry`** (see page 7-57 for an explanation of this routine).
period	an integer value indicating the frequency with which the specified program is to be wakened in each major frame. A period of one indicates that the specified program is to be wakened every minor cycle; a period of two indicates that it is to be wakened once every two minor cycles, a period of three once every three minor cycles, and so on. This value can range from one to the number of minor cycles that compose a frame on the specified scheduler as defined in a call to **`fbsconfigure`** (see page 7-6).

Table 7-19. Contents of Structure Components: sched_pgmadd (Cont.)

Component	Contents
cycle	an integer value indicating the first minor cycle in which the specified program is scheduled to be wakened in each frame. This value can range from zero to the total number of minor cycles per frame minus one. The total number of minor cycles per frame is specified in a call to **fbsconfigure** (see page 7-6 for an explanation of this routine).
halt	an integer value indicating whether or not the scheduler should be stopped in the event that the specified program causes a frame overrun. A nonzero value indicates that the scheduler will be stopped.

Return Value

A return value of **0** indicates that the call has been successful. A return value of **–1** indicates that an error has occurred; **errno** is set to indicate the error. Refer to the system manual page **sched_pgmadd(3rt)** for a listing of the types of errors that may occur.

Sched_pgm_set_soft_overrun_limit

Sets the consecutive soft overrun limit for a currently scheduled LWP on the frequency-based scheduler. To set the consecutive soft overrun limit, the calling LWP must have alter permission for the scheduler. If the Enhanced Security Utilities are installed and running, the Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

The LWP can be identified in one of the following ways:

• fpid only (if name_ptr is the null string).

• path name and processor id pair only (if fpid is -1).

• fpid, path name, and processor id.

Specification

#include <fbslib.h>

int sched_pgm_set_soft_overrun_limit (fbs_id, soft_overrun_buf)
int fbs_id;
struct soft_overrun_ds *soft_overrun_buf;

Parameters

fbs_id	Obtained from an fbsid(3rt) library routine call or set to -1. -1 enables an FBS-scheduled LWP to reference the frequency-based scheduler on which it is scheduled without knowing the scheduler identifier.
soft_overrun_buf	soft_overrun_ds structure that contains the soft overrun status for the scheduled LWP.

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned and errno is set to one of the error values listed in the system manual pages.

Sched_pgm_soft_overrun_query

Invoked to obtain status of soft overrun processing for a currently scheduled LWP on the frequency-based scheduler.

Specification

#include <fbslib.h>

```
int sched_pgm_soft_overrun_query(fbs_id, soft_overrun_buf)
int fbs_id;
struct soft_overrun_info_ds *soft_overrun_buf;
```

Parameters

fbs_id	Obtained from an fbsid(3rt) library routine call or set to -1. -1 enables an FBS-scheduled LWP to reference the frequency-based scheduler on which it is scheduled without knowing the scheduler identifier.
soft_overrun_buf	soft_overrun_ds structure that contains the soft overrun status for the scheduled LWP.

Return Value

Upon successful completion, a value of 0 is returned. Otherwise, a value of –1 is returned, and errno is set to one of the values listed in the system manual pages.

Sched_pgmqry – Query a Process on an FBS

The **`sched_pgmqry`** routine is invoked to obtain information for a particular process that has been scheduled on a frequency–based scheduler. You can identify the process by using one of the following methods:

- Specify the name of the process and the CPU(s) on which it is scheduled.
- Specify the process's frequency–based scheduler process identifier (slot number).
- Specify the name of the process, the CPU on which it is scheduled, and its frequency–based scheduler process identifier.

NOTE

> The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency–based scheduler process identifier.

Information that is returned includes the following:

- The process's path name
- The CPU on which the process can execute
- The frequency–based scheduler process identifier
- The scheduling policy under which the process has been scheduled
- The scheduling priority
- The period (the number of minor cycles indicating the frequency with which the process is wakened in each major frame)
- The starting base cycle (the first minor cycle in which the process is scheduled to be wakened in each major frame)
- The value of the "halt on overrun" flag

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int sched_pgmqry(fbs_id, qry_buf)
int fbs_id;
struct pgm2_ds {
    char *name_ptr;
    int cpu;
    int fpid;
    int cid;
    int prior;
    int param;
    int period;
    int cycle;
    int halt;
    int status;
} *qry_buf;
```

Call

```
struct pgm2_ds  qry_buf;
int istat;
istat = sched_pgmqry(fbs_id, &qry_buf);
```

Parameters

Parameters are described as follows.

fbs_id
: refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process for which you wish to obtain scheduling information has been scheduled. You can obtain this value by making a call to **fbsconfigure** (see page 7-6 for an explanation of this routine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of –1.

qry_buf
: refers to a **pgm2_ds** structure that contains information identifying the process for which information is to be returned. **Sched_pgmqry** will return to this structure the scheduling information for a specified process. The information contained in each component of the structure to which **qry_buf** points is presented in Table 7-20.

Table 7-20. Contents of Structure Components: sched_pgmqry

Component	Contents	
name_ptr	a pointer to a variable that contains a standard UNIX path name identifying the process for which information is to be returned. A full or relative path name of up to 1024 characters can be specified. If the pointer points to a null string, you must provide the frequency–based scheduler process identifier in the *fpid* component.	
cpu	An integer value indicating the processor(s) to be used in conjunction with the value of *name_ptr* to identify the program for which information is to be obtained. Acceptable values and corresponding results follow:	
	0	The first process whose name matches the name pointed to by *name_ptr* that is currently running on the processor from which the call is made is specified
	–1	The first process whose name matches the name pointed to by *name_ptr* that is currently running on any processor is specified

	Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process whose name matches the name pointed to by *name_ptr* that is running on CPU *i* is specified
		
		If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process whose name matches the name pointed to by *name_ptr* that is currently running on any of the selected CPUs is specified

Table 7-20. Contents of Structure Components: sched_pgmqry (Cont.)

Component	Contents
fpid	an integer value providing the unique frequency–based scheduler process identifier for the process for which information is to be returned. This value is obtained when you make a call to **sched_pgmadd** (see page 7-52 for an explanation of this routine). This value must be –1 if you wish to identify the program to be queried only by specifying *name_ptr* and *cpu*.
cid	an integer value indicating the specified process's scheduling policy
prior	an integer value indicating the specified process's scheduling priority
param	an integer value indicating the value passed to the process via a call to **sched_pgmadd** or **sched_pgmresched**
period	an integer value indicating the frequency with which the specified program is to be wakened in each major frame
cycle	an integer value indicating the first minor cycle in which the specified process is scheduled to be wakened in each frame
halt	an integer value indicating the value of the "halt on overrun" flag. A nonzero value indicates that the flag is set. A value of zero indicates that the flag is not set.
status	an integer value indicating the current state of the specified process as defined in **<fbslib.h>**

Return Value

A return value of **0** indicates that the call has been successful. A return value of – **1** indicates that an error has occurred; **errno** is set to indicate the error. Refer to the system manual page **sched_pgmqry(3rt)** for a listing of the types of errors that may occur.

Sched_pgmresched – Reschedule a Process

The **sched_pgmresched** routine is invoked to change the scheduling parameters for a process that is scheduled on a frequency–based scheduler. You may wish, for example, to change a program's policy or priority or the frequency with which it is scheduled to run. You cannot, however, change the CPU on which it has been scheduled.

If you wish to (1) change a process's scheduling policy to the **SCHED_FIFO** or the **SCHED_RR** policy or (2) change the priority of a process scheduled under the **SCHED_FIFO** or the **SCHED_RR** policy, the following conditions must be met:

- The calling process must have the P_RTIME privilege.
- The effective user ID of the calling process must match the effective user ID of the target process (the process for which the scheduling policy and priority are being set), or the calling process must have the P_OWNER privilege.

If the Enhanced Security Utilities are installed and running, the following additional conditions must be met:

- The Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

If you wish to raise the priority of a process scheduled under the **SCHED_OTHER** policy above a per-process or LWP limit, the following conditions must be met:

- The calling process must have the P_TSHAR privilege.
- The effective user ID of the calling process must match the effective user ID of the target process (the process for which the scheduling priority is being set), or the calling process must have the P_OWNER privilege.

If the Enhanced Security Utilities are installed and running, the following additional conditions must be met:

- The Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

For additional information on privileges, refer to the *PowerMAX OS Programming Guide* and the **`intro(2)`** system manual page.

You can identify the process that you wish to reschedule by using one of the following methods:

- Specify the name of the process and the CPU on which it is scheduled.
- Specify the process's frequency–based scheduler process identifier (slot number).
- Specify the name of the process, the CPU on which it is scheduled, and its frequency–based scheduler process identifier.

NOTE

> The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency–based scheduler process identifier.

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int sched_pgmresched(fbs_id, rsch_buf)
int fbs_id;
struct pgm2_ds {
   char *name_ptr;
```

```
    int  cpu;
    int  fpid;
    int  cid;
    int  prior;
    int  param;
    int  period;
    int  cycle;
    int  halt;
    int  status;
} *rsch_buf;
```

Call

```
struct pgm2_ds  rsch_buf;
int istat;
istat = sched_pgmresched(fbs_id, &rsch_buf);
```

Parameters

Parameters are described as follows.

fbs_id
: refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process is scheduled. You can obtain this value by making a call to **fbsconfigure** (see page 7-6 for an explanation of this routine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

rsch_buf
: refers to a **pgm2_ds** structure that contains the scheduling parameters with which you wish to reschedule the process. The type of information that is specified in each component is presented in Table 7-21. Note that the *status* component is ignored on this call.

Table 7-21. Contents of Structure Components: sched_pgmresched

Component	Contents	
name_ptr	a pointer to a variable that contains a standard UNIX path name identifying the process for which information is to be rescheduled. A full or relative path name of up to 1024 characters can be specified. If the pointer points to a null string, you must provide the frequency–based scheduler process identifier in the *fpid* component.	
cpu	An integer value indicating the processor(s) to be used in conjunction with the value of *name_ptr* to identify the process to be rescheduled. Acceptable values and corresponding results are as follows:	
	0	The first process whose name matches the name pointed to by *name_ptr* that is currently running on the processor from which the call is made is rescheduled

Table 7-21. Contents of Structure Components: sched_pgmresched (Cont.)

Component	Contents		
	–1	The first process whose name matches the name pointed to by *name_ptr* that is currently running on any processor is rescheduled	

	Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process whose name matches the name pointed to by *name_ptr* that is running on CPU *i* is rescheduled
		
		If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process whose name matches the name pointed to by *name_ptr* that is currently running on any of the selected CPUs is rescheduled	

fpid	an integer value providing the unique frequency–based scheduler process identifier for the process to be rescheduled. This value is obtained when you make a call to **`sched_pgmadd`** (see page 7-52 for an explanation of this routine). This value must be – **`1`** if you wish to identify the program to be rescheduled only by specifying *name_ptr* and *cpu*.		
cid	an integer value indicating the scheduling policy under which the specified program is to be scheduled. Scheduling policies are defined in the file <**`sched.h`**>. The value of *cid* must be one of the following:		
	SCHED_FIFO		first–in–first out (FIFO) scheduling policy
	SCHED_RR		round–robin (RR) scheduling policy. Note that a process cannot be scheduled under this policy on a CPU on which servicing of the 60 Hz clock interrupt has been disabled. In such cases, the process behaves as though it were scheduled under the **SCHED_FIFO** policy.
	SCHED_OTHER		time-sharing scheduling policy

Table 7-21. Contents of Structure Components: sched_pgmresched (Cont.)

Component	Contents
prior	an integer value indicating the scheduling priority of the specified process. The range of acceptable priority values is governed by the scheduling policy specified. You can determine the allowable range of priorities associated with each policy (**SCHED_FIFO**, **SCHED_RR**, or **SCHED_OTHER**) by invoking the **`run(1)`** command from the shell and not specifying any options or arguments (see the corresponding system manual page for an explanation of this command). Higher numerical values correspond to more favorable priorities. For complete information on scheduling policies and priorities, refer to the “Process Scheduling and Management” chapter of the *PowerMAX OS Programming Guide*.
param	an integer value to be passed to a process that is scheduled on a frequency–based scheduler
period	an integer value indicating the frequency with which the specified program is to be wakened in each major frame. A period of one indicates that the specified program is to be wakened every minor cycle; a period of two indicates that it is to be wakened once every two minor cycles, a period of three once every three minor cycles, and so on. This value can range from one to the number of minor cycles that compose a frame on the specified scheduler as defined in a call to **`fbsconfigure`** (see page 7-6).
cycle	an integer value indicating the first minor cycle in which the specified process is scheduled to be wakened in each frame. This value can range from zero to the total number of minor cycles per frame minus one. The total number of minor cycles per frame is specified in a call to **`fbsconfigure`** (see page 7-6 for an explanation of this routine).
halt	an integer value indicating whether or not the scheduler should be stopped in the event that the specified process causes a frame overrun. A nonzero value indicates that the scheduler will be stopped.

Return Value

A return value of **`0`** indicates that the call has been successful. A return value of **–1** indicates that an error has occurred; **`errno`** is set to indicate the error. Refer to the system manual page **`sched_pgmresched(3rt)`** for a listing of the types of errors that may occur.

The Performance Monitor Routines

The performance monitor routines provide access to the key features of the performance monitor. They enable you to perform such basic operations as the following: (1) clear performance monitor values for a process or processor, (2) start and stop performance monitoring for a process or processor, and (3) obtain performance monitor values for a process or processor.

In the sections that follow, all of the performance monitor routines contained in the **`librt`** library are presented in alphabetical order. Figure 7-2 illustrates the approximate order in which you might call the routines from an application program.

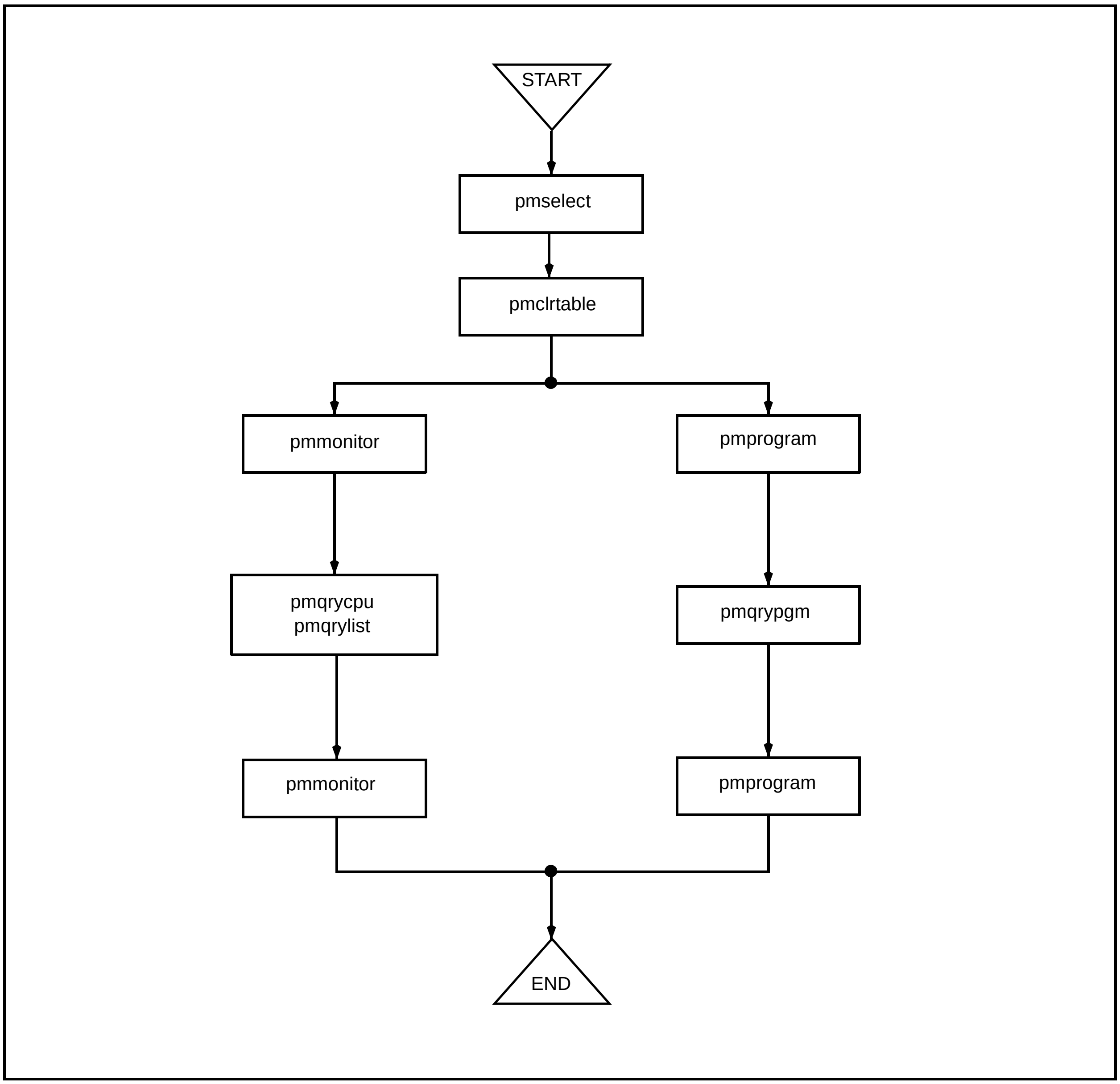

Figure 7-2. C Library Call Sequence: Performance Monitor

Pmclrpgm – Clear Values for a Process

This routine is invoked to clear performance monitor values for a particular process that has been scheduled on a frequency–based scheduler. You can identify the process by using one of the following methods:

- Specify the name of the process and the CPU(s) on which it is scheduled.
- Specify the process's frequency–based scheduler process identifier (slot number).
- Specify the name of the process, the CPU on which it is scheduled, and its frequency–based scheduler process identifier.

NOTE

This routine will clear the process' total soft overrun count.

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>


int  pmclrpgm(fbs_id, name, cpu, fpid)
int  fbs_id;
char *name;
int  cpu;
int  fpid;
```

Call

```
int istat;
istat = pmclrpgm(fbs_id, name, cpu, fpid);
```

Parameters

Parameters are described as follows.

fbs_id
: refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process is scheduled. You can obtain this value by making a call to **`fbsconfigure`** (see page 7-6 for an explanation of this routine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **`1`**.

name
: refers to a variable that contains a standard UNIX path name identifying the process for which values are to be cleared. A full or relative path name of up to 1024 charac-

ters can be specified. If this variable is the null string, you must provide the frequency–based scheduler process identifier in the *fpid* parameter.

cpu — refers to a variable that contains an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the process for which values are to be cleared. Acceptable values and corresponding results are presented in Table 7-22.

Table 7-22. CPU Options: pmclrpgm

Value	Result
0	The first process named by *name* that is currently running on the processor from which the call is made is specified
-1	The first process named by *name* that is currently running on any processor is specified

Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is currently running on CPU *i* is specified
	
	If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is specified

fpid — refers to a variable that contains an integer value providing the unique frequency–based scheduler process identifier for the process for which values are to be cleared. This value is obtained when you make a call to **`sched_pgmadd`** (see page 7-52 for an explanation of this routine). This value must be **–1** if you wish to identify the process only by specifying *name* and *cpu*.

Return Value

A return value of **`0`** indicates that the call has been successful. A return value of – **`1`** indicates that an error has occurred; **`errno`** is set to indicate the error. Refer to the system manual page **`pmclrpgm(3rt)`** for a listing of the types of errors that may occur.

Pmclrtable – Clear Values for Processor(s)

This routine is invoked to clear performance monitor values for FBS–scheduled processes on one or more specified processors on a selected scheduler.

NOTE

> This routine will clear the total soft overrun count for all related processes.

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int  pmclrtable(fbs_id, cpu)
int  fbs_id;
int  cpu;
```

Call

```
int istat;
istat = pmclrtable(fbs_id, cpu);
```

Parameters

Parameters are described as follows.

fbs_id — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the processes are scheduled. You can obtain this value by making a call to **fbsconfigure** (see page 7-6 for an explanation of this routine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

cpu — refers to an integer value indicating the processor or processors for which performance monitor values are to be cleared. Acceptable values and corresponding results are presented in Table 7-23.

Table 7-23. CPU Options: pmclrtable

Value	Result
0	Performance monitor values for FBS–scheduled processes executing on the processor from which the call is made are cleared
-1	Performance monitor values for all processes on the scheduler are cleared
Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU), performance monitor values for processes executing on CPU *i* are cleared

Return Value

A return value of **0** indicates that the call has been successful. A return value of – **1** indicates that an error has occurred; **errno** is set to indicate the error. Refer to the system manual page **pmclrtable(3rt)** for a listing of the types of errors that may occur.

Pmmonitor – Start/Stop Performance Monitoring on Processor(s)

This routine is invoked to start or stop performance monitoring for FBS–scheduled processes on one or more specified processors on a selected scheduler.

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>


int  pmmonitor(fbs_id, pmflag, cpu)
int  fbs_id;
int  pmflag;
int  cpu;
```

Call

```
int istat;
istat = pmmonitor(fbs_id, pmflag, cpu);
```

Parameters

Parameters are described as follows.

fbs_id	refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the processes are scheduled. You can obtain this value by making a call to **fbsconfigure** (see page 7-6 for an explanation of this routine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.
pmflag	refers to a variable that contains an integer value indicating whether performance monitoring is to be started or stopped. A nonzero value indicates that performance monitoring is to be started. A zero value indicates that performance monitoring is to be stopped.
cpu	refers to an integer that indicates the processor or processors for which performance monitoring is to be started or stopped. Acceptable values and corresponding results are presented in Table 7-24.

Table 7-24. CPU Options: pmmonitor

Value	Result
0	Performance monitoring for FBS–scheduled processes executing on the processor from which the call is made is started or stopped
-1	Performance monitoring for all processes on the scheduler is started or stopped
Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU), performance monitoring for processes executing on CPU *i* is started or stopped

Return Value

A return value of **0** indicates that the call has been successful. A return value of – **1** indicates that an error has occurred; **errno** is set to indicate the error. Refer to the system manual page **pmmonitor(3rt)** for a listing of the types of errors that may occur.

Pmprogram – Start/Stop Performance Monitoring on a Process

This routine is invoked to start or stop performance monitoring for a particular process that has been scheduled on a frequency–based scheduler. You can identify the process by using one of the following methods:

- Specify the name of the process and the CPU(s) on which it is scheduled.
- Specify the process's frequency–based scheduler process identifier (slot number).
- Specify the name of the process, the CPU(s) on which it is scheduled, and its frequency–based scheduler process identifier.

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

 int  pmprogram(fbs_id, name, cpu, fpid, pmflag)
int  fbs_id;
char *name;
int  cpu;
int  fpid;
int  pmflag;
```

Call

int istat;
istat = pmprogram(fbs_id, name, cpu, fpid, pmflag);

Parameters

Parameters are described as follows.

fbs_id
: refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process is scheduled. You can obtain this value by making a call to **`fbsconfigure`** (see page 7-6 for an explanation of this routine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **`1`**.

name
: refers to a variable that contains a standard UNIX path name identifying the process for which performance monitoring is to be started or stopped. A full or relative path name of up to 1024 characters can be specified. If this variable is the null string, you must provide the frequency–based scheduler process identifier in the *slot* parameter.

cpu
: refers to a variable that contains an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the process for which performance monitoring is to be started or stopped. Acceptable values and corresponding results are presented in Table 7-25.

Table 7-25. CPU Options: pmprogram

Value	Result
0	The first process named by *name* that is currently running on the processor from which the call is made is specified
-1	The first process named by *name* that is currently running on any processor is specified

Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is currently running on CPU *i* is specified
	
	If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is specified

fpid — refers to a variable that contains an integer value providing the unique frequency–based scheduler process identifier for the process for which performance monitoring is to be started or stopped. This value is obtained when you make a call to **sched_pgmadd** (see page 7-52 for an explanation of this routine). This value must be – **1** if you wish to identify the process only by specifying *name* and *cpu*.

pmflag — refers to a variable that contains an integer value indicating whether performance monitoring is to be started or stopped. A nonzero value indicates that performance monitoring is to be started. A zero value indicates that performance monitoring is to be stopped.

Return Value

A return value of **0** indicates that the call has been successful. A return value of – **1** indicates that an error has occurred; **errno** is set to indicate the error. Refer to the system manual page **pmprogram(3rt)** for a listing of the types of errors that may occur.

Pmqrycpu – Query Values for Selected Processor(s)

This routine is invoked to obtain performance monitor values for FBS–scheduled processes on one or more specified processors on a selected scheduler.

The C specification, call, corresponding parameters, and return value are presented in the following sections. Note that all time related units are in microseconds (usecs).

Specification

```
#include <fbslib.h>

int  pmqrycpu(fbs_id, cpu, pm_buf, buf_cnt)
int  fbs_id;
int  cpu;
struct pmqry_ds {
   int    fpid;              /* FBS process identifier */
   int    lastcyc_tm;        /* time used in last run cycle */
   usecs
   int    tot_cycles;        /* total number of cycles executed */
   int    tot_sec;           /* total number of seconds used */
   int    tot_usec;          /* total microseconds used */
   int    overruns;          /* number of overruns by process */
   int    mincyc_tm;         /* minimum time used in a cycle */
   usecs
   int    mincyc_cycle;      /* cycle number of minimum cycle time */
   int    mincyc_frame;      /* frame number of minimum cycle time */
   int    maxcyc_tm;         /* maximum time used in a cycle */
   usecs
   int    maxcyc_cycle;      /* cycle number of maximum cycle time */
   int    maxcyc_frame;      /* frame number of maximum frame time */
   int    minframe_tm;       /* minimum time used in a frame */
```

```
    usecs
    int    minframe;         /* frame number of minimum frame time */
    int    maxframe_tm;      /* maximum time used in a frame */
    usecs
    int    maxframe;         /* frame number of maximum frame time */
} *pm_buf;
int  buf_cnt;
```

Call

```
struct pmqry_ds  pm_buf[buf_cnt];
int istat;
istat = pmqrycpu(fbs_id, cpu, pm_buf, buf_cnt);
```

Parameters

Parameters are described as follows.

fbs_id — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the processes are scheduled. You can obtain this value by making a call to **fbsconfigure** (see page 7-6 for an explanation of this routine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

cpu — refers to a variable that contains an integer value indicating the processor(s) for which performance monitor values are to be obtained. Acceptable values and corresponding results are presented in Table 7-26.

Table 7-26. CPU Options: pmqrycpu

Value	Result
0	Performance monitor values for FBS–scheduled processes executing on the processor from which the call is made are returned
-1	Performance monitor values for all processes on the scheduler are returned
Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU), performance monitor values for processes executing on CPU *i* are returned

pm_buf — refers to an array of *pmqry_ds* structures to which **pmqrycpu** will return the performance monitor values for each FBS–scheduled process on the processor(s) specified with the *cpu* parameter. The number of processes for which these values are returned is bound by the value of the *buf_cnt* parameter. The type of information returned in each component of the structure for a single process is presented in Table 7-27.

Table 7-27. Contents of Structure Components: pmqrycpu

Component	Contents
fpid	The process's frequency–based scheduler process identifier (slot number)
lastcyc_tm	The amount of time (in usecs) that the process has spent running from the last time that it has been wakened by the scheduler until it has called **fbswait** (last time)
tot_cycles	The number of times that the process has been wakened by the scheduler (total iterations, or cycles)
tot_sec	The number of seconds that the process has spent running in all cycles (total seconds). The total amount of time that the process has spent running is equal to the value of *tot_sec* plus the value of *tot_usec*.
tot_usec	The additional number of microseconds that the process has spent running in all cycles (total microseconds). The total amount of time that the process has spent running is equal to the value of *tot_sec* plus the value of *tot_usec*.
overruns	The number of hard frame overruns caused by the process
mincyc_tm	The least amount of time (in usecs) that the process has spent running in a cycle (minimum cycle time)
mincyc_cycle	The number of the minor cycle in which the minimum cycle time has occurred (minimum cycle cycle)
mincyc_frame	The number of the major frame in which the minimum cycle time has occurred (minimum cycle frame)
maxcyc_tm	The greatest amount of time (in usecs) that the process has spent running in a cycle (maximum cycle time)
maxcyc_cycle	The number of the minor cycle in which the maximum cycle time has occurred (maximum cycle cycle)
maxcyc_frame	The number of the major frame in which the maximum cycle time has occurred (maximum cycle frame)
minframe_tm	The least amount of time (in usecs) that the process has spent running during a major frame (minimum frame time)

Table 7-27. Contents of Structure Components: pmqrycpu (Cont.)

Component	Contents
minframe	The number of the major frame in which the minimum frame time has occurred (minimum frame frame)
maxframe_tm	The greatest amount of time (in usecs) that the process has spent running during a major frame (maximum frame time)
maxframe	The number of the major frame in which the maximum frame time has occurred (maximum frame frame)

buf_cnt refers to a variable that contains an integer value indicating the number of structures in the array to which *pm_buf* points.

Return Value

A return value of **0** indicates that the call has been successful. A return value of – **1** indicates that an error has occurred; **errno** is set to indicate the error. Refer to the system manual page **pmqrycpu(3rt)** for a listing of the types of errors that may occur.

Pmqrylist – Query Values for a List of Processes

This routine is invoked to obtain performance monitor values for a list of processes scheduled on a frequency–based scheduler.

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int  pmqrylist(fbs_id, pm_buf, buf_cnt)
int  fbs_id;
struct pmqry_ds {
   int    fpid;                /* FBS process identifier */
   int    lastcyc_tm;          /* time used in last run cycle */
   usecs
   int    tot_cycles;          /* total number of cycles executed */
   int    tot_sec;             /* total number of seconds used */
   int    tot_usec;            /* total microseconds used */
   int    overruns;            /* number of overruns by process */
   int    mincyc_tm;           /* minimum time used in a cycle */
   usecs
   int    mincyc_cycle;        /* cycle number of minimum cycle time */
   int    mincyc_frame;        /* frame number of minimum cycle time */
   int    maxcyc_tm;           /* maximum time used in a cycle */
   usecs
```

```
    int     maxcyc_cycle;       /* cycle number of maximum cycle time */
    int     maxcyc_frame;       /* frame number of maximum frame time */
    int     minframe_tm;        /* minimum time used in a frame */
    usecs
    int     minframe;           /* frame number of minimum frame time */
    int     maxframe_tm;        /* maximum time used in a frame */
    usecs
    int     maxframe;           /* frame number of maximum frame time */
} *pm_buf;
int  buf_cnt;
```

Call

```
struct pmqry_ds  pm_buf[buf_cnt];
int istat;
istat = pmqrylist(fbs_id, pm_buf, buf_cnt);
```

Parameters

Parameters are described as follows.

fbs_id
: refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler for which performance monitor values are requested. You can obtain this value by making a call to **fbsconfigure** (see page 7-6 for an explanation of this routine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of **–1**.

pm_buf
: refers to an array of *pmqry_ds* structures to which **pmqrylist** will return the performance monitor values for a list of FBS–scheduled processes. The list of processes for which values are returned is created by placing the frequency–based scheduler identifier in the *fpid* component of each structure in the array. The type of information contained in each component of the structure for a single process is presented in Table 7-28.

Table 7-28. Contents of Structure Components: pmqrylist

Component	Contents
fpid	An integer value providing the unique frequency–based scheduler process identifier for which performance monitor values are to be returned
lastcyc_tm	The amount of time (in usecs) that the process has spent running from the last time that it has been wakened by the scheduler until it has called **fbswait** (last time)
tot_cycles	The number of times that the process has been wakened by the scheduler (total iterations, or cycles)

Table 7-28. Contents of Structure Components: pmqrylist (Cont.)

Component	Contents
tot_sec	The number of seconds that the process has spent running in all cycles (total seconds). The total amount of time that the process has spent running is equal to the value of *tot_sec* plus the value of *tot_usec*.
tot_usec	The additional number of microseconds that the process has spent running in all cycles (total microseconds). The total amount of time that the process has spent running is equal to the value of *tot_sec* plus the value of *tot_usec*.
overruns	The number of hard frame overruns caused by the process
mincyc_tm	The least amount of time (in usecs) that the process has spent running in a cycle (minimum cycle time)
mincyc_cycle	The number of the minor cycle in which the minimum cycle time has occurred (minimum cycle cycle)
mincyc_frame	The number of the major frame in which the minimum cycle time has occurred (minimum cycle frame)
maxcyc_tm	The greatest amount of time (in usecs) that the process has spent running in a cycle (maximum cycle time)
maxcyc_cycle	The number of the minor cycle in which the maximum cycle time has occurred (maximum cycle cycle)
maxcyc_frame	The number of the major frame in which the maximum cycle time has occurred (maximum cycle frame)
minframe_tm	The least amount of time (in usecs) that the process has spent running during a major frame (minimum frame time)
minframe	The number of the major frame in which the minimum frame time has occurred (minimum frame frame)
maxframe_tm	The greatest amount of time (in usecs) that the process has spent running during a major frame (maximum frame time)
maxframe	The number of the major frame in which the maximum frame time has occurred (maximum frame frame)

buf_cnt — refers to a variable that contains an integer value indicating the number of structures in the array to which *pm_buf* points.

Return Value

A return value of **0** indicates that the call has been successful. A return value of – **1** indicates that an error has occurred; **errno** is set to indicate the error. Refer to the system manual page **pmqrylist(3rt)** for a listing of the types of errors that may occur.

Pmqrypgm – Query Values for a Selected Process

This routine is invoked to obtain performance monitor values for a particular process scheduled on a frequency–based scheduler.

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int  pmqrypgm(fbs_id, name, cpu, pm_buf)
int  fbs_id;
char *name;
int  cpu;
struct pmqry_ds {
   int     fpid;              /* FBS process identifier */
   int     lastcyc_tm;        /* time used in last run cycle */
   usecs
   int     tot_cycles;        /* total number of cycles executed */
   int     tot_sec;           /* total number of seconds used */
   int     tot_usec;          /* total microseconds used */
   int     overruns;          /* number of overruns by process */
   int     mincyc_tm;         /* minimum time used in a cycle */
   usecs
   int     mincyc_cycle;      /* cycle number of minimum cycle time */
   int     mincyc_frame;      /* frame number of minimum cycle time */
   int     maxcyc_tm;         /* maximum time used in a cycle */
   usecs
   int     maxcyc_cycle;      /* cycle number of maximum cycle time */
   int     maxcyc_frame;      /* frame number of maximum frame time */
   int     minframe_tm;       /* minimum time used in a frame */
   usecs
   int     minframe;          /* frame number of minimum frame time */
   int     maxframe_tm;       /* maximum time used in a frame */
   usecs
   int     maxframe;          /* frame number of maximum frame time */
} *pm_buf;
```

Call

```
struct pmqry_ds  pm_buf;
int istat;
istat = pmqrypgm(fbs_id, name, cpu, &pm_buf);
```

Parameters

Parameters are described as follows.

fbs_id — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler for which performance monitor values are requested. You can obtain this value by making a call to

fbsconfigure (see page 7-6 for an explanation of this routine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

name — refers to a pointer to a variable that contains a standard UNIX path name identifying the process for which performance monitoring values are to be returned. A full or relative path name of up to 1024 characters can be specified. If this variable is the null string, you must provide the frequency–based scheduler process identifier in the *fpid* component of the structure to which *pm_buf* points.

cpu — refers to a variable that contains an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the process for which performance monitoring values are to be returned. Acceptable values and corresponding results are presented in Table 7-29.

Table 7-29. CPU Options: pmqrypgm

Value	Result
0	The first process named by *name* that is currently running on the processor from which the call is made is specified
-1	The first process named by *name* that is currently running on any processor is specified

Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is currently running on CPU *i* is specified
	
	If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is specified

pm_buf — refers to a **pmqry_ds** structure to which **pmqrypgm** will return the performance monitor values for the FBS–scheduled process pointed to by the *name* parameter. The type of information contained in each component of the structure is presented in Table 7-30 below.

Table 7-30. Contents of Structure Components: pmqrypgm

Component	Contents
fpid	An integer value providing the unique frequency–based scheduler process identifier for which performance monitor values are to be returned. This value is obtained when you make a call to **sched_pgmadd** (see page 7-52 for an explanation of this routine). This value must be – **1** if you wish to identify the process only by specifying *name* and *cpu*.
lastcyc_tm	The amount of time (in usecs) that the process has spent running from the last time that it has been wakened by the scheduler until it has called **fbswait** (last time)
tot_cycles	The number of times that the process has been wakened by the scheduler since the last time that performance monitor values have been cleared and performance monitoring has been enabled (total iterations, or cycles)
tot_sec	The number of seconds that the process has spent running in all cycles (total seconds). The total amount of time that the process has spent running is equal to the value of *tot_sec* plus the value of *tot_usec*.
tot_usec	The additional number of microseconds that the process has spent running in all cycles (total microseconds). The total amount of time that the process has spent running is equal to the value of *tot_sec* plus the value of *tot_usec*.
overruns	The number of hard frame overruns caused by the process
mincyc_tm	The least amount of time (in usecs) that the process has spent running in a cycle (minimum cycle time)
mincyc_cycle	The number of the minor cycle in which the minimum cycle time has occurred (minimum cycle cycle)
mincyc_frame	The number of the major frame in which the minimum cycle time has occurred (minimum cycle frame)
maxcyc_tm	The greatest amount of time (in usecs) that the process has spent running in a cycle (maximum cycle time)
maxcyc_cycle	The number of the minor cycle in which the maximum cycle time has occurred (maximum cycle cycle)
maxcyc_frame	The number of the major frame in which the maximum cycle time has occurred (maximum cycle frame)

Table 7-30. Contents of Structure Components: pmqrypgm (Cont.)

Component	Contents
minframe_tm	The least amount of time (in usecs) that the process has spent running during a major frame (minimum frame time)
minframe	The number of the major frame in which the minimum frame time has occurred (minimum frame frame)
maxframe_tm	The greatest amount of time (in usecs) that the process has spent running during a major frame (maximum frame time)
maxframe	The number of the major frame in which the maximum frame time has occurred (maximum frame frame)

Return Value

A return value of `0` indicates that the call has been successful. A return value of – `1` indicates that an error has occurred; **`errno`** is set to indicate the error. Refer to the system manual page **`pmqrypgm(3rt)`** for a listing of the types of errors that may occur.

Pmqrytimer – Query Performance Monitor Mode

This routine is invoked to determine whether performance monitor timing values include or exclude time spent servicing interrupts.

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int pmqrytimer()
int mode;
```

Call

```
int mode;
mode = pmqrytimer;
```

Return Value

The value of *mode* is set to zero to indicate that interrupt time is excluded from performance monitor timing values; it is set to one to indicate that interrupt time is included in timing values; it is set to – **`1`** if an error occurs, and **`errno`** is set to indicate the error. Refer to the system manual page **`pmqrytimer(3rt)`** for a listing of the types of errors that may occur.

Pmselect – Select Performance Monitor Mode

This routine is invoked to select the timing mode under which the performance monitor is to run. The timing mode can be set to include or exclude time spent servicing interrupts. Note that to set the timing mode, the calling process must have the P_RTIME privilege (for additional information on privileges, refer to the *PowerMAX OS Programming Guide* and the **intro(2)** system manual page).

CAUTION

The timing mode for the high–resolution timing facility is set system–wide. It affects all processes running on all CPUs.

The C specification, call, corresponding parameters, and return value are presented in the following sections.

Specification

```
#include <fbslib.h>

int pmselect(mode)
int mode;
```

Call

```
int istat;
istat = pmselect(mode);
```

Parameters

Parameters are described as follows.

mode	refers to a variable that contains an integer value indicating whether time spent servicing interrupts is to be included in or excluded from performance monitor timing values. A nonzero value indicates that interrupt time is to be included. A value of zero indicates that interrupt time is to be excluded.

Return Value

A return value of **0** indicates that the call has been successful. A return value of – **1** indicates that an error has occurred; **errno** is set to indicate the error. Refer to the system manual page **pmselect(3rt)** for a listing of the types of errors that may occur.

Compiling and Linking Programs

When statically linking a C program, the following library is required.

/usr/lib/librt.a

When dynamically linking a program the following library is used:

/usr/lib/librt.so

To compile and link a C program, the command line instruction is as follows:

hc *source_file***.c -lrt**

For additional information on compiling and linking procedures, refer to the system manual pages **ld(1)** and **cc(1)**.

8
The FORTRAN Library Interface

8
The FORTRAN Library Interface

The real–time library for FORTRAN, **`/usr/lib/libF77rt.a`**, contains subroutines that enable you to perform the entire range of functions associated with the frequency–based scheduler and the performance monitor. The frequency–based scheduler subroutines are presented in “The FBS Subroutines.” The performance monitor subroutines are presented in “The Performance Monitor Subroutines.” For each subroutine, the following information is provided:

- A description of the subroutine
- The FORTRAN variable declarations and CALL statement needed to reference the subroutine in an application program
- Detailed descriptions of each parameter.

Procedures for compiling and linking user programs are presented in “Compiling and Linking Procedures.”

The FBS Subroutines

The FBS subroutines provide access to the key features of the scheduler. They enable you to perform such basic operations as the following: (1) configure a scheduler; (2) schedule programs on it; (3) set up and connect a timing device to a scheduler; (4) start, stop, and resume scheduling on a scheduler; (5) obtain information about scheduled processes; (6) reschedule and remove scheduled processes; (7) disconnect a timing device; and (8) remove a scheduler.

In the sections that follow, all of the FBS subroutines contained in the **`libF77rt`** library are presented in alphabetical order. Figure 8-1 illustrates the approximate order in which you might invoke the subroutines from an application program.

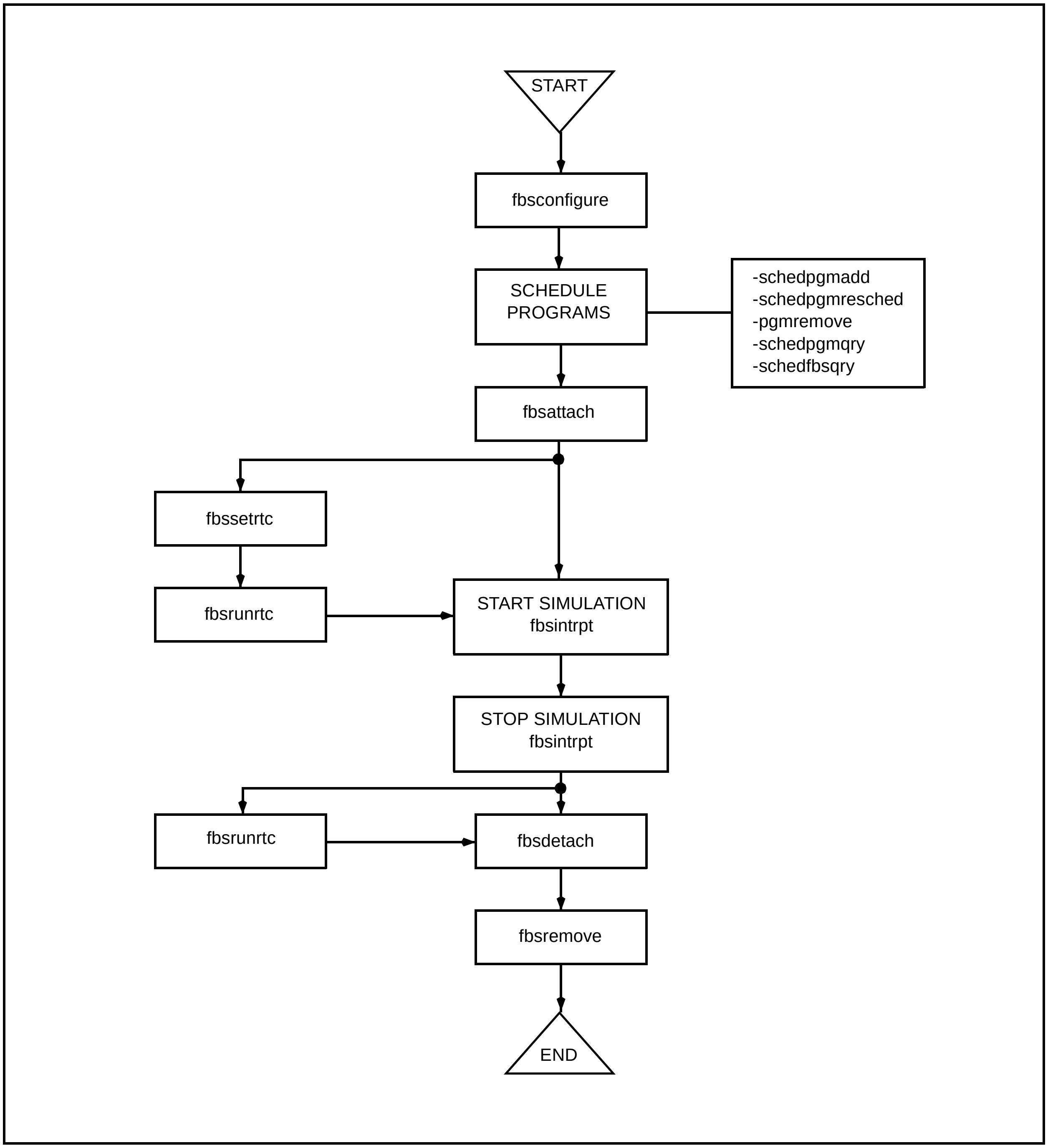

Figure 8-1. FORTRAN Library Call Sequence: FBS

Fbsaccess – Change Permissions for an FBS

This subroutine is invoked to change the permissions assigned for a selected frequency–based scheduler. It is important to note that the permissions can be changed only by a process that has the P_OWNER privilege or has an effective user ID that is equal to that of the owner/creator of the frequency–based scheduler.

If the Enhanced Security Utilities are installed and running, the following conditions must also be met:

- The calling process and the frequency-based scheduler must have identical security levels, or the process must have both P_MACREAD and P_MACWRITE privileges.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER schdle
INTEGER uid
INTEGER gid
INTEGER permissions
INTEGER istat

CALL Statement

CALL fbsaccess (schdle, uid, gid, permissions, istat)

Parameters

Parameters are described as follows.

schdle — refers to a variable that contains a unique, positive integer value representing the identifier for a frequency–based scheduler. You can obtain this value by making a call to **`fbsconfigure`** (see page 8-6 for an explanation of this subroutine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing the identifier, you can specify the value –**`1`**.

uid — refers to a variable that contains an integer value representing the effective user ID of the specified frequency–based scheduler.

gid — refers to a variable that contains an integer value representing the effective group ID of the specified frequency–based scheduler.

permissions — refers to a variable that contains a bit pattern used to set the permissions associated with the specified frequency–based scheduler. Bit patterns and corresponding permissions are presented in Table 8-1. Additional information on setting permissions for frequency–based scheduler operations is provided in the system manual page **`intro(2)`**.

Table 8-1. FBS Permissions

Bit Pattern	Permissions
400	Read by user
200	Alter by user
060	Read, alter by group
006	Read, alter by others

istat — refers to a variable to which **fbsaccess** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page **fbsaccess(3F77rt)** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Fbsattach – Attach Timing Source to an FBS

This subroutine is invoked to attach a timing source to a frequency–based scheduler or to specify end–of–cycle scheduling. The timing source can be a real–time clock, an edge–triggered interrupt device, or a user–supplied real–time device.

NOTE

Subroutines contained in the FORTRAN library do not provide the functionality to set up and control operation of an edge–triggered interrupt device or a user–supplied device, as they do for a real–time clock. Procedures for using a real–time clock are described in detail in Chapter 3. Procedures for using an edge–triggered interrupt and a user–supplied real–time device are also explained in that chapter.

To use a real–time clock as the timing source for a frequency–based scheduler on a PowerMAX OS system on which the Enhanced Security Utilities are installed, you must have enough privilege to open the device. Refer to the “Trusted Facility Management” chapter of *System Administration Volume 1* for an explanation of the procedures for using devices when the Enhanced Security Utilities are installed.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER schdle
INTEGER cpu
CHARACTER* (*) devname
INTEGER istat

CALL Statement

CALL fbsattach(schdle, cpu, devname, istat)

Parameters

Parameters are described as follows.

schdle — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler for which the timing source is to be attached or end–of–cycle scheduling specified. You can obtain this value by making a call to **`fbsconfigure`** (see page 8-6 for an explanation of this subroutine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **`1`**.

cpu — refers to a variable that must contain the value **`0`**.

devname — refers to a variable that contains a null string or the path name of the device that is to be used as the timing source for the specified scheduler. If *devname* contains a null string, end–of–cycle scheduling is specified; that is, execution of the processes in the next minor cycle will occur when the last process scheduled to execute in the current minor cycle finishes its execution for that cycle. If *devname* contains a path name, it may refer to a real–time clock, an edge–triggered interrupt, or a user–supplied device.

If the device is a real–time clock or an edge–triggered interrupt, the path name must be of a certain form. Refer to Chapter 3 for detailed information on the form associated with each type of device.

If the device is a user–supplied device, the path name must be a valid UNIX path name. The device must support the IOCTLVECNUM **`ioctl(2)`** call (see Chapter 3 for additional information).

If the device is a Coupled timing device, the path name must be of a certain form. Refer to Chapter 3 for detailed information on the form associated with a cluster timing source.

istat — refers to a variable to which **`fbsattach`** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page **fbsattach(3F77rt)** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Fbsconfigure – Configure an FBS

This subroutine is invoked to configure a frequency–based scheduler or to obtain configuration details for a frequency–based scheduler that has already been configured. Note that to configure a scheduler, the calling process must have the P_RTIME privilege (for additional information on privileges, refer to the *PowerMAX OS Programming Guide* and the **intro(2)** system manual page).

If you wish to configure a scheduler, you must specify a *key*, which is a user–chosen numeric identifier for a frequency–based scheduler. You must also specify a *configflg*, which is a word that sets the permission and control flag bits to characterize the scheduler.

The permissions are defined in the system manual page **intro(2)**.

The control flags are described in the header file <**sys/ipc.h**>. They include **IPC_CREAT** and **IPC_EXCL**. Setting the **IPC_CREAT** bit without setting the **IPC_EXCL** bit ensures that a new frequency–based scheduler is created if one corresponding to the value of *key* does not exist; it results in the return of the associated frequency–based scheduler identifier if one does exist and if <u>all</u> of the following conditions are met:

- The number of minor cycles specified by the *cycles* parameter matches the number of minor cycles associated with the existing scheduler
- The maximum specified by the *progs* parameter is less than or equal to the maximum number of processes per minor cycle associated with the existing scheduler
- The maximum specified by the *max* parameter is less than or equal to the maximum number of processes allowed on the existing scheduler at one time

Setting both the **IPC_CREAT** and the **IPC_EXCL** bits results in the creation of a new scheduler if one corresponding to the value of *key* does not exist; it ensures that an error is returned if one does exist.

A unique, nonnegative frequency–based scheduler identifier and corresponding data structure will be created for the specified key if the number of frequency–based schedulers already configured is less than the maximum number of schedulers allowed on your system (see Chapter 2 for a description of system tunable parameters) and if <u>one</u> of the following conditions is met:

- The value of *key* is equal to **IPC_PRIVATE** (that is, zero)
- The value of *key* is not associated with a frequency–based scheduler identifier and (*configflg* & **IPC_CREAT**) is "true"

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER key
INTEGER cycles
INTEGER progs
INTEGER max
INTEGER reset
INTEGER configflg
INTEGER schdle
INTEGER istat

CALL Statement

CALL fbsconfigure(key, cycles, progs, max, reset, configflg, schdle, istat)

Parameters

To create a frequency–based scheduler, you must specify the following parameters as described.

key — refers to a variable that contains an integer value identifying the frequency–based scheduler that is to be created. Note that the number of schedulers that can be configured at one time cannot exceed the value of FBSMNI, which is the maximum number of frequency–based schedulers permitted on your system (see Chapter 2 for a description of system tunable parameters).

cycles — refers to a variable that contains an integer value indicating the number of minor cycles that compose a frame on the specified scheduler.

progs — refers to a variable that contains an integer value indicating the maximum number of programs that can be scheduled to execute during one minor cycle.

max — refers to a variable that contains an integer value indicating the maximum number of programs that can be scheduled on the specified scheduler at one time. This value must be less than or equal to the product that is obtained by multiplying the values specified for the *cycles* and *progs* parameters.

reset — refers to a variable that contains an integer value indicating whether or not processes currently scheduled on the specified scheduler are to be killed before the scheduler is reconfigured. Acceptable values and corresponding results are presented in Table 8-2.

configflg — refers to a variable that contains an integer value indicating the control flags and permissions assigned to the specified scheduler. See the header file **<sys/ipc.h>** to determine the locations of the bits.

schdle — refers to a variable to which **fbsconfigure** will return a unique, positive integer value representing the identifier for the specified frequency–based scheduler. It is important to

Table 8-2. Reset Options

Value	Result
<0	Kill and remove all processes currently scheduled on the specified scheduler
0	Ignore all processes currently scheduled on the specified scheduler
>0	Remove all processes currently scheduled on the specified scheduler

note that this identifier is required by most of the library subroutines for the FBS and the performance monitor.

istat — refers to a variable to which **fbsconfigure** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page **fbsconfigure(3F77rt)** for a listing of the nonzero values that may be returned and the types of errors that they represent.

To obtain information for an existing frequency–based scheduler, you must specify the following parameters as described.

key — refers to a variable that contains an integer value identifying the frequency–based scheduler for which configuration information is to be returned. If this value is zero, the frequency–based scheduler identifier associated with this scheduler must also be provided by using the *schdle* parameter.

cycles — refers to a variable that contains the integer value zero, indicating that current configuration information for the specified scheduler is to be returned. **Fbsconfigure** will return to this variable an integer value indicating the number of minor cycles that compose a frame on the specified scheduler.

progs — refers to a variable to which **fbsconfigure** will return the maximum number of programs that can be scheduled to run during one minor cycle on the specified scheduler.

max — refers to a variable to which **fbsconfigure** will return the maximum number of programs that can be scheduled on the specified scheduler at one time.

configflg — refers to a variable to which **fbsconfigure** will return the permissions assigned to the specified scheduler.

schdle — refers to a variable to which **fbsconfigure** will return a unique, positive integer value representing the identifier for the specified frequency–based scheduler. If you specify a key of **0**, this variable must contain the related frequency–based scheduler identifier.

Fbscycle – Return Minor Cycle/Major Frame Count

This subroutine is invoked to obtain the current minor cycle and major frame count values for a frequency–based scheduler. These values enable you to determine the progress of a simulation.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER schdle
INTEGER count(2)
INTEGER istat

CALL Statement

CALL fbscycle(schdle, count, istat)

Parameters

Parameters are described as follows.

schdle
: refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler for which you wish to obtain the current cycle and frame counts. You can obtain this value by making a call to **fbsconfigure** (see page 8-6 for an explanation of this subroutine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of –1.

count
: refers to an array to which **fbscycle** will return integer values indicating the current minor cycle and major frame for the specified scheduler. *Count(1)* will contain the number of the cycle. *Count(2)* will contain the number of the frame.

istat
: refers to a variable to which **fbscycle** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page for **fbscycle(3F77rt)** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Fbsdetach – Detach Timing Source from an FBS

This subroutine is invoked to detach the currently attached timing source from a frequency–based scheduler or to disable end–of–cycle scheduling. If the timing source is a real-time clock, it is recommended that you stop the clock prior to invoking this subroutine. You can do so by making a call to **fbsrunrtc** (see page 8-26 for an explanation of this subroutine).

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER schdle
INTEGER istat

CALL Statement

CALL fbsdetach(schdle, istat)

Parameters

Parameters are described as follows.

schdle — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler from which you wish to detach the currently attached timing source or for which you wish to disable end–of–cycle scheduling. You can obtain this value by making a call to **fbsconfigure** (see page 8-6 for an explanation of this subroutine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

istat — refers to a variable to which **fbsdetach** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page **fbsdetach(3F77rt)** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Fbsgetrtc – Obtain Current Values for Real–Time Clock

This subroutine is invoked to obtain the current count and resolution values for the real–time clock that is attached to a specified frequency–based scheduler.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER schdle
INTEGER count
INTEGER resolution
INTEGER istat1
INTEGER istat2

CALL Statement

CALL fbsgetrtc(schdle, count, resolution, istat1, istat2)

Parameters

Parameters must be specified in the order indicated. They are described as follows.

schdle
: refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler to which the real–time clock is attached. You can obtain this value by making a call to **fbsconfigure** (see page 8-6 for an explanation of this subroutine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

count
: refers to a variable to which **fbsgetrtc** will return an integer value indicating the current number of clock counts per minor cycle. This value can range from one to 65535.

resolution
: refers to a variable to which **fbsgetrtc** will return an integer value indicating the current duration in microseconds of one clock count. This value will be one of the following: 1, 10, 100, 1000, or 10000.

istat1
: refers to a variable to which **fbsgetrtc** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page **fbsgetrtc(3F77rt)** for a listing of the nonzero values that may be returned and the types of errors that they represent. If *istat1* contains a value indicating that an error has occurred on an **open** or **ioctl** call, the error status of that call is returned in *istat2*.

istat2
: refers to a variable to which **fbsgetrtc** will return the error status of an **open** or **ioctl** call. See the include file **<errno.h>** for a description of the errors.

Fbsid – Return the FBS Identifier for a Key

This subroutine is invoked to obtain the frequency–based scheduler identifier associated with a particular user–specified key. The key must match the key that was specified when the scheduler was created by making a call to **`fbsconfigure(3C)`**.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

```
INTEGER key
INTEGER schdle
INTEGER istat
```

CALL Statement

```
CALL fbsid(key, schdle, istat)
```

Parameters

Parameters must be specified in the order indicated. They are described as follows.

key
: refers to a variable that contains an integer value identifying a frequency–based scheduler; this value must be the same value that was specified for *key* when the scheduler was created by making a call to **`fbsconfigure`** (see page 8-6 for an explanation of this subroutine).

schdle
: refers to a variable to which **`fbsid`** will return an integer value representing the unique frequency–based scheduler identifier associated with the key.

istat
: refers to a variable to which **`fbsid`** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A non-zero value indicates that an error of a specific type has occurred. Refer to the system manual page **`fbsid(3F77rt)`** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Fbsinfo – Return Information for an FBS

This subroutine is invoked to obtain information that is related to a selected frequency–based scheduler but cannot be obtained by invoking other subroutines (for example, **`schedfbsqry`**, **`schedpgmqry`**). Such information includes the following:

- The user and group IDs of the owner and the creator of the scheduler
- The permissions assigned for the scheduler
- The key associated with the scheduler's identifier
- The total number of overruns for all processes on the scheduler
- The CPUs that are active in the system
- The CPUs on which performance monitoring has been enabled
- The FBS–enabled flag
- The path name of the device that has been attached to the scheduler

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER schdle
INTEGER buf(41)
CHARACTER* (*) devname
INTEGER istat

CALL Statement

CALL fbsinfo(schdle, buf, devname, istat)

Parameters

Parameters are described as follows.

schdle — refers to a variable that contains a unique, positive integer value representing the identifier for a frequency–based scheduler. You can obtain this value by making a call to **`fbsconfigure`** (see page 8-6 for an explanation of this subroutine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of **`–1`**.

buf — refers to an array to which **`fbsinfo`** will return information about the specified scheduler. The information returned in each element of the array is presented in Table 8-3.

Table 8-3. Contents of Array Elements

Element	Contents
buf(1)	owner's user ID
buf(2)	owner's group ID
buf(3)	creator's user ID
buf(4)	creator's group ID
buf(5)	access modes
buf(6)	key
buf(7)	flags word
buf(8)	reserved for future use
buf(9)	total number of hard overruns for all processes on the scheduler
buf(10)	mask of CPUs active in the system
buf(11)	mask of CPUs on which performance monitoring has been enabled
buf(12)	FBS–enabled flag
buf(13)–(41)	reserved for future use

devname — refers to a variable to which **fbsinfo** will return the path name of the device that is being used as the timing source for the specified frequency–based scheduler. If end–of–cycle scheduling has been specified, **devname** will contain a null string.

istat — refers to a variable to which **fbsinfo** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page **fbsinfo(3F77rt)** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Fbsinfo_rdev - Return Coupled FBS timing device information

This routine may be called to obtain information about a Coupled FBS timing device. The information returned from this routine includes the following:

- The type of Coupled FBS timing device; either RCIM Coupled or Closely-Coupled. See Chapter 3 for more information about these two types of timing devices.
- The host name of the host where the timing device actually resides (where the device interrupt originates).
- A list of all the host names of the hosts where this device is registered.
- A list of all the host names of the hosts where there are schedulers attached to this device.
- The path name of the actual device on the host where the device resides.

The FORTRAN variable declarations, CALL statement and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER attr_flags, num_hosts, host_flags(num_hosts), istat
CHARACTER *(*) rdevfs_name
CHARACTER *(*) device_name
CHARACTER *(*) hostnames(num_hosts)

Call Statement

CALL fbsinfo_rdev(rdevfs_name, device_name, attr_flags, num_hosts, host_flags, hostnames, istat)

Parameters

rdevfs_name
: Refers to a string where the caller specifies the full path name of the registered **`rdevfs(4)`** device: **`/dev/rdev/<hostname>/device<`*`n`*`>`**.

device_name
: Refers to a variable where the actual device name of the timing device on the host where the device resides will be returned.

attr_flags
: Refers to a variable where **`fbsinfo_rdev`** will return the type of timing device, where the type values may be:

1 - for a Closely-Coupled timing device, or
2 - for a RCIM Coupled timing device.

See Chapter 3 for a description of these two types of timing devices.

num_hosts
: Refers to a variable where the caller specifies how many entries are in the **`host_flags`** and hostnames arrays.

When **0** or **-3** is returned in the istat parameter upon return from **fbsinfo_rdev**, the *num_hosts* parameter will be modified so that it contains the number of actual registered hosts for this device.

When **-3** is returned, then the caller did not make the **`host_flags`** and **`hostnames`** arrays large enough to hold all of the registered per-host information. In this case, the value returned in **`num_hosts`** reflects the total number of registered hosts, where this value is larger than the number of entries in the **`host_flags`** and **`hostnames`** array. When **-3** is returned, the caller may wish to allocate more space for these arrays. However, **`fbsinfo_rdev`** has filled in all available entries in the host_flags and hostname arrays with valid data, and thus, the caller may still examine the per-host information that was returned.

host_flags — Refers to an integer array of host attribute flags, with one word for each registered host. When successful, the returned **`num_hosts`** value contains the number of valid entries in this array, starting from the front entry in the array. Each relative entry in the **`num_hosts`** array corresponds to the same relative entry in the hostnames character array. The flag values that may be returned in each host_flags entry are:

1 - this host has a scheduler attached to this device.

2 - this host is where the device interrupt originates.

3 - this host has a scheduler attached to this device and this host is also where the device interrupt originates.

hostnames — Refers to a character array of space where the hostname strings of the registered hosts for this timing device are returned. When successful, the **`num_hosts`** parameter contains the number of valid entries in this array, starting from the front of the array.

istat — Refers to a variable in which **`fbsinfo_rdev`** will return an integer value indicating whether or not an error has occurred. A non-zero value indicates that an error of a specific type has occurred. Refer to the system manual page **`fbsinfo_rdev(3F77rt)`** for a listing of the non-zero values that may be returned and the types of errors that they represent.

NOTE

The **fbsinfo_rdev** function call is not compatible for use with timing devices that were registered with a **fbs_register_cluster_device** function call. In this case, the user should use the **fbsinfo_cluster** function call to obtain additional information about the Closely-Coupled timing device. However, the **fbs_register_cluster_device** and **fbs_unregister_cluster_device** function calls are obsolete and users are encouraged to make use of the **fbs_register_rdev**, **fbs_unregister_rdev** and **fbsinfo_rdev** function calls.

Fbsinfo_cluster - Return cluster information for an FBS

This routine is invoked to obtain information about the Closely-Coupled timing device that a selected frequency-based scheduler is currently attached to. The information returned from this routine includes the following:

- The SBC board ID where the Closely-Coupled timing device actually resides
- The path name of the actual device on the SBC board where the device resides
- A bit mask of SBC board IDs of the SBCs that currently have schedulers attached to this device

Note that the selected frequency-based scheduler must be currently attached to a Closely-Coupled timing device in order for this routine call to be successful.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER schdle
INTEGER buf(8)
CHARACTER *(*) devname
INTEGER istat

CALL Statement

CALL fbsinfo_cluster(schdle, buf, devname, istat)

Parameters

schdle	refers to a variable that contains a unique, positive integer value representing the identifier for a frequency-based scheduler. You can obtain this value by making a call to **fbsconfigure** (see page 8-6 for an explanation of this subroutine). If you wish to reference the frequency-based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of -1.
buf	refers to an array to which fbsinfo_cluster will return information about the specified scheduler. The information returned in each element of the array is presented in Table 8-4.
devname	refers to a variable to which **fbsinfo_cluster** will return the path name of the actual device that is being used as the timing source on the SBC board where the device resides.
istat	refers to a variable which **fbsinfo_cluster** will return an integer value indicating whether or not an error has occurred. A non-zero value indicates that an error of a specific type has occurred. Refer to the system manual page

Table 8-4. Contents of Array Elements

Element	Contents
buf(1)	SBC board ID where device actually resides
buf(2)	SBC board ID mask of those SBCs that contain schedulers that are currently attached to this timing device
buf(3)-(8)	reserved for future use

fbsinfo_cluster(3F77rt) for a listing of the non-zero values that may be returned and the types of errors that they represent.

Fbsintrpt – Start/Stop/Resume Scheduling on an FBS

This subroutine is invoked to start, stop, or resume scheduling on a frequency–based scheduler. If you invoke this subroutine to start scheduling, the minor cycle, major frame, and overrun count values are reset. If you invoke it to resume scheduling, these values are not reset.

Prior to invoking **fbsintrpt**, you must have invoked **fbsattach** to specify end–of–cycle scheduling or attach a timing source to the frequency–based scheduler on which you are starting scheduling (see page 8-4 for an explanation of **fbsattach**). If you have specified a real–time clock as the timing source, scheduling will not begin until you have set and started the clock (see pages 8-30 and 8-26 for explanations of **fbssetrtc** and **fbsrunrtc**, respectively). If you have specified an edge–triggered interrupt device or a user–supplied device as the timing source, it must already be generating interrupts in order for scheduling to start.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER schdle
INTEGER intrflag
INTEGER istat

CALL Statement

CALL fbsintrpt(schdle, intrflag, istat)

Parameters

Parameters are described as follows.

schdle — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which you wish to start, stop, or resume scheduling of processes. You can obtain this value by making a call to **`fbsconfigure`** (see page 8-6 for an explanation of this subroutine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **`1`**.

intrflag — refers to a variable that contains an integer value indicating whether scheduling of processes on the specified scheduler is to be started, stopped, or resumed. Acceptable values and corresponding results are presented in Table 8-5.

Table 8-5. Intrflag Options

Value	Result
<0	Start scheduling of processes with the initial frame, cycle, and overrun count values set to zero
0	Stop scheduling of processes, and save the count values for the current frame and cycle
>0	Resume scheduling of processes with the frame, cycle, and overrun count values set to the values that were saved when the scheduler was last stopped

istat — refers to a variable to which **`fbsintrpt`** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page **`fbsintrpt(3F77rt)`** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Fbsquery – Query Processes on an FBS

CAUTION

This interface is obsolete. It is maintained for compatibility with CX/UX, but it returns processes' scheduling priorities without any indication of the scheduling policies with which they are associated. If you have an existing application that uses this interface, it is recommended that you change your application to use **`schedfbsqry(3F77rt)`** (see p. 8-54). For details on obsolete interfaces, refer to Chapter 2, "Overview of the FBS."

This subroutine is invoked to obtain information about processes that have been scheduled on a frequency–based scheduler. Information is returned for all processes scheduled on the user–specified processor(s). Information provided for each process includes the following:

- A mask of the CPU(s) on which the process can execute
- The frequency–based scheduler process identifier
- The scheduling priority
- The period (the number of minor cycles indicating the frequency with which the process is wakened in each major frame)
- The starting base cycle (the first minor cycle in which the process is scheduled to be wakened in each major frame)
- The value of the “halt on overrun” flag

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

```
INTEGER schdle
INTEGER cpu
INTEGER buf1size
INTEGER buf1(buf1size)
INTEGER maxsize
INTEGER buf2size
CHARACTER* (*) buf2
INTEGER istat
```

CALL Statement

```
CALL fbsquery(schdle, cpu, buf1size, buf1, maxsize, buf2size, buf2, istat)
```

Parameters

Parameters are described as follows.

schdle
: refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler for which you wish to obtain scheduling information. You can obtain this value by making a call to **fbsconfigure** (see page 8-6 for an explanation of this subroutine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of **–1**.

cpu
: refers to a variable that contains an integer value indicating the processor(s) for which scheduling information is to be obtained. Acceptable values and corresponding results are presented in Table 8-6.

buf1size
: refers to a variable that contains an integer value indicating the size in 32–bit words of the array represented by **buf1**. Because 10 words of information are returned for each

Table 8-6. CPU Options: fbsquery

Value	Result
0	Scheduling information for processes executing on the processor from which the call is made is returned
-1	Scheduling information for all processes on the scheduler is returned
Bit mask	If (*cpu* & (1<<***i***)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU), scheduling information for processes executing on CPU ***i*** is returned

process, it is recommended that this value be a multiple of 10.

buf1 — refers to an array to which **`fbsquery`** will return a series of 10 integer values for each process on the processor(s) specified with the *cpu* parameter. The number of processes for which these values are returned is bound by the value of the *buf1size* parameter. If, for example, the value of *buf1size* is 145, values for 14 processes will be returned. These values represent the scheduling information for the process(es). The type of information returned in each array element for a single process is presented in Table 8-7.

Table 8-7. Contents of Array Elements for a Process

Element	Contents
1	Byte offset of the process's path name in *buf2*
2	Length in bytes of the process's path name
3	Zero
4	Zero
5	Mask of the CPU(s) on which the process can execute
6	The process's frequency–based scheduler process identifier
7	The process's scheduling priority

Table 8-7. Contents of Array Elements for a Process (Cont.)

Element	Contents
8	The number of minor cycles indicating the frequency with which the process is to be wakened in each major frame (period)
9	The first minor cycle in which the process is scheduled to be wakened in each major frame (starting base cycle)
10	The value of the “halt on overrun” flag. A nonzero value indicates that the flag is set. A value of zero indicates that the flag is not set.

maxsize — refers to a variable that contains an integer value indicating the maximum length of a path name to be returned in *buf2*

buf2size — refers to a variable that contains an integer value indicating the size in bytes of the character string represented by *buf2*. To ensure that *buf2* is large enough to accommodate the names of all processes that you wish to query, you may find it helpful to compute the number of bytes needed by multiplying the maximum number of processes allowed on the scheduler (see the information on **fbsconfigure** presented on page 8-6) by 32.

buf2 — refers to a variable to which **fbsquery** will return the path names for each process on the processor(s) specified with the *cpu* parameter. Path names are returned as a series of strings. The length of each string is less than or equal to the value of *maxsize*. Where *maxsize* is not large enough to accommodate a full path name, the concluding component names are returned. The number of path names returned is bound by the value of the *buf2size* parameter.

istat — refers to a variable to which **fbsquery** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page **fbsquery(3F77rt)** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Fbsremove – Remove an FBS

This subroutine is invoked to remove a frequency–based scheduler and to free the data structure associated with it. It is important to note that prior to invoking **fbsremove**, you must ensure that the timing source is detached from the scheduler or that end–of–cycle scheduling is disabled (see page 8-10 for information on the use of **fbsdetach**). It is important to note that **fbsremove** will remove all processes scheduled on the specified scheduler. It is recommended, however, that you remove all scheduled processes prior to invoking **fbsremove**. You can do so by making a call to **pgmremove** (see page 8-39 for information on the use of this subroutine).

Note that to remove a frequency-based scheduler, the calling process must have the P_OWNER privilege or an effective user ID that is equal to that of the owner/creator of the scheduler.

If the Enhanced Security Utilities are installed and running, the following conditions must also be met:

- The calling process and the frequency-based scheduler must have identical security levels, or the process must have both P_MACREAD and P_MACWRITE privileges.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER schdle
INTEGER ab
INTEGER istat

CALL Statement

CALL fbsremove(schdle, ab, istat)

Parameters

Parameters are described as follows.

schdle — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler that you wish to remove. You can obtain this value by making a call to **fbsconfigure** (see page 8-6 for an explanation of this subroutine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of **–1**.

ab — refers to a variable that contains an integer value indicating the manner in which processes scheduled on the scheduler are to be handled. Acceptable values and corresponding results are presented in Table 8-8.

istat — refers to a variable to which **fbsremove** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page for **fbsremove(3F77rt)** for a listing of the nonzero

Table 8-8. Ab Options

Value	Result
<0	Kill and remove all processes currently scheduled on the specified scheduler
≥0	Remove all processes currently scheduled on the specified scheduler

values that may be returned and the types of errors that they represent.

Fbsresume – Resume Scheduling on an FBS

The **fbsresume** subroutine is invoked to resume scheduling of processes on a frequency-based scheduler at the specified minor cycle, major frame, and overrun count.

Note that to resume scheduling of processes on a frequency-based scheduler, the calling process must have alter permission for the scheduler. If the Enhanced Security Utilities are installed and running, the following conditions must also be met:

- The calling process and the frequency-based scheduler must have identical security levels, or the process must have the P_MACWRITE privilege.

If you wish to resume scheduling of processes on a frequency-based scheduler without altering the scheduler's current frame, cycle, and overrun values, it is recommended that you use the **fbsintrpt(3F77rt)** subroutine (see page 8-18 for an explanation of this subroutine).

CAUTION

The **fbsresume** subroutine clears performance monitor values for all processes scheduled on the specified scheduler. Changing the frame and cycle count for the scheduler causes the values that are being maintained by the performance monitor to be inaccurate.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER schdle
INTEGER frame
INTEGER cycle

INTEGER overruns
INTEGER istat

CALL Statement

CALL fbsresume(schdle, frame, cycle, overruns, istat)

Parameters

Parameters are described as follows.

schdle — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which you wish to resume scheduling of processes. You can obtain this value by making a call to **`fbsconfigure`** or **`fbsid`** (see page 8-6 and page 8-12, respectively, for explanations of these subroutine). If you wish to reference the frequency–based scheduler on which the calling LWP is scheduled without knowing the identifier, you can specify the value –1.

frame — an integer value indicating the major frame in which you wish scheduling of processes to be resumed on the specified scheduler

cycle — an integer value indicating the minor cycle in which you wish scheduling of processes to be resumed on the specified scheduler. This value can range from zero to the total number of minor cycles per frame minus one. The total number of minor cycles per frame was specified when the scheduler was created by making a call to **`fbsconfigure`** (see page 8-6 for an explanation of this subroutine).

overruns — an integer value indicating the value to which you wish the overrun count to be set when scheduling resumes on the specified scheduler. If you do not wish to change the overrun count, you can specify the value –1.

istat — refers to a variable to which **`fbsresume`** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page **`fbsresume(3F77rt)`** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Fbsrunrtc – Start/Stop Real–Time Clock

This subroutine is invoked to start or stop the counting of a real–time clock that has been attached to a frequency–based scheduler.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER schdle
INTEGER runflag
INTEGER istat1
INTEGER istat2

CALL Statement

CALL fbsrunrtc(schdle, runflag, istat1, istat2)

Parameters

Parameters are described as follows.

schdle
: refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler for which you wish to start or stop the attached real–time clock. You can obtain this value by making a call to **fbsconfigure** (see page 8-6 for an explanation of this subroutine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of –1.

runflag
: refers to a variable that contains an integer value indicating whether the real–time clock is to be started or stopped. A nonzero value indicates that the clock is to be started. A zero value indicates that the clock is to be stopped.

istat1
: refers to a variable to which **fbsrunrtc** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page for **fbsrunrtc(3F77rt)** for a listing of the nonzero values that may be returned and the types of errors that they represent. If *istat1* contains a value indicating that an error has occurred on an **open** or **ioctl** call, the error status of that call is returned in *istat2*.

istat2
: refers to a variable to which **fbsrunrtc** will return the error status of an **open** or **ioctl** call. See the include file <**errno.h**> for a description of the error.

Fbsschedself – Schedule an LWP on an FBS

The **fbsschedself** subroutine is invoked to schedule the calling lightweight process (LWP) on a frequency-based scheduler.

This subroutine is designed to be used by a single-threaded or a multithreaded application; however, if it is to be used in a multithreaded application, it can be used only by bound threads.

It is important to note that **fbsschedself** does not allow an LWP to set its scheduling policy and priority or its CPU bias. These tasks must be performed prior to invoking **fbsschedself**.

A single-threaded process can set its scheduling policy and priority by using the **sched_setscheduler(3C)** library routine; it can set its CPU bias by using the **cpu_bias(2)** system call or the **mpadvise(3C**) library routine. Procedures for using these functions are explained in the "Process Scheduling and Management" and "Process Management" chapters of the *PowerMAX OS Programming Guide*.

A bound thread can set its scheduling policy and priority by using the **thr_setscheduler(3thread)** library routine; it can set its CPU bias by using the **cpu_bias** system call or the **mpadvise** library routine. Complete information on bound thread scheduling and use of the **thr_setscheduler** routine are provided in the "Thread Scheduling" section of the "Programming with the Threads Library" chapter of the *PowerMAX OS Programming Guide*.

Note that you cannot use this subroutine to add **/idle** or **/spare** to a frequency-based scheduler.

To schedule the calling LWP on a frequency-based scheduler, the calling LWP must have alter permission for the scheduler. If the Enhanced Security Utilities are installed and running, the following conditions must also be met:

- The calling LWP and the frequency-based scheduler must have identical security levels, or the LWP must have the P_MACWRITE privilege.

You must not change the scheduling policy or priority of an LWP while it is scheduled on a scheduler by using **sched_setscheduler**, **thr_setscheduler**, or other program interfaces that allow you to change scheduling policy and priority. The frequency-based scheduler is not aware of changes in scheduling policy and priority that are made by using these interfaces.

If you need to change the scheduling policy or priority of a single-threaded FBS-scheduled process, you may do so by using **schedpgmresched** to reschedule it (see page 8-63 for an explanation of this routine).

If you need to change the scheduling policy or priority of a bound thread, you must first remove it from the scheduler on which it is has been scheduled by using **pgmremove** (see page 8-39 for an explanation of this subroutine). You can then use **thr_setscheduler** to change its policy or priority and **fbsschedself** to schedule it on a scheduler.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER schdle
CHARACTER* (*) name
INTEGER (*) sched_buf
INTEGER istat

CALL Statement

CALL fbsschedself(schdle, name, sched_buf, istat)

Parameters

Parameters are described as follows.

schdle	refers to a variable that contains a unique, positive integer value representing the identifier for a frequency–based scheduler. You can obtain this value by making a call to **fbsconfigure** or **fbsid** (see page 8-6 and page 8-12, respectively, for explanations of these subroutines). If you wish to reference the frequency–based scheduler on which the calling LWP is scheduled without knowing the identifier, you can specify the value –1.
name	refers to a variable that contains a standard UNIX path name or arbitrary content identifying the program associated with the calling LWP. A full or relative path name of up to 1023 characters can be specified.
sched_buf	refers to an integer array that contains the scheduling parameters with which you wish to schedule the LWP. The information that is specified in this array is presented in Table 8-9.
istat	refers to a variable to which **fbsschedself** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page **fbsschedself(3F77rt)** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Table 8-9. Contents of Array Elements

Element	Contents
sched_buf(1)	an integer value indicating the version of *sched_buf* that is being passed to **fbsschedself**. Specify the symbolic constant **FBSSCHED_BUF_V1**, which is defined in **<fbslib.h>** for this purpose.
sched_buf(2)	an integer value to be passed to a process that is scheduled on a frequency-based scheduler. This value can be retrieved by the FBS–scheduled process through a call to **rtparm** (see page 8-52 for an explanation of this subroutine).
sched_buf(3)	an integer value indicating the frequency with which the calling LWP is to be wakened in each major frame. A period of one indicates that the calling LWP is to be wakened every minor cycle; a period of two indicates that it is to be wakened once every two minor cycles, a period of three once every three minor cycles, and so on. This value can range from one to the number of minor cycles that compose a frame on the specified scheduler as defined in a call to **fbsconfigure** (see page 8-6 for an explanation of this subroutine).
sched_buf(4)	an integer value indicating the first minor cycle in which the calling LWP is scheduled to be wakened in each frame. This value can range from zero to the total number of minor cycles per frame minus one. The total number of minor cycles per frame is specified in a call to **fbsconfigure** (see page 8-6 for an explanation of this routine).
sched_buf(5)	an integer value indicating whether or not the scheduler should be stopped in the event that the calling LWP causes a frame overrun. A nonzero value indicates that the scheduler will be stopped.
sched_buf(6)	an integer value that is returned by **fbsschedself** and is the unique frequency–based scheduler process identifier for the scheduled LWP

Fbssetrtc – Set Real–Time Clock

This subroutine is invoked to establish the duration of a minor cycle by setting the count and the resolution values for a real–time clock.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER schdle
INTEGER count
INTEGER resolution
INTEGER istat1
INTEGER istat2

CALL Statement

CALL fbssetrtc(schdle, count, resolution, istat1, istat2)

Parameters

Parameters must be specified in the order indicated. They are described as follows.

schdle
: refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler to which a real–time clock has been attached. You can obtain this value by making a call to **fbsconfigure** (see page 8-6 for an explanation of this subroutine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of **–1**.

count
: refers to a variable that contains an integer value indicating the number of clock counts per minor cycle. This value can range from one to 65535.

resolution
: refers to a variable that contains an integer value indicating the duration in microseconds of one clock count. This value must be one of the following: 1, 10, 100, 1000, or 10000.

istat1
: refers to a variable to which **fbssetrtc** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page **fbssetrtc(3F77rt)** for a listing of the nonzero values that may be returned and the types of errors that they represent. If *istat1* contains a value indicating that an error has occurred on an **open** or **ioctl** call, the error status of that call is returned in *istat2*.

istat2
: refers to a variable to which **fbssetrtc** will return the error status of an **open** or **ioctl** call. See the include file <**errno.h**> for a description of the error.

Fbswait – Wait on an FBS

This subroutine enables a process that is scheduled on a frequency-based scheduler to sleep until its next scheduled minor cycle.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER istat

CALL Statement

CALL fbswait(istat)

Parameter

`Fbswait` requires one parameter: *istat*. *Istat* refers to a variable to which **`fbswait`** will return an integer value indicating whether or not an error has occurred and whether the process has been wakened by the scheduler or by an **`fbstrig(2)`** call from another process. Values that may be returned are described in Table 8-10.

Table 8-10. Istat Values: fbswait

Value	Description
0	The process has been wakened normally
1	The process has been wakened as the result of an **`fbstrig(2)`** call
2	The process did not sleep because the kernel detected a soft overrun and is allowing the process to attempt to recover from it.
Other nonzero value	An error of a specific type has occurred. Refer to the system manual page **`fbswait(3F77rt)`** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Fbs_register_rdev - Register Coupled FBS Timing Device

This routine may be used to register a local device as a remote timing device (**`rdevfs(4)`**) which may be subsequently used as a Coupled FBS timing device. A Coupled timing device may be used to couple together FBS schedulers that are located on more than one computer system. All schedulers that are attached to the same Coupled FBS timing device will start, stop and resume their executions together on the same frame and cycle, using the Coupled FBS timing device as the interrupt source.

To register a timing device, the calling process must have the P_RTIME privilege as well as enough privilege to open the device file.

Successfully registering a device as a Coupled FBS timing source creates a placeholder, or virtual FBS identifier to reserve the device's interrupt vector. There is one virtual FBS for each device registered, and this virtual FBS provides the means for a process on another host to communicate with the real device. Because the virtual FBS is allocated exactly the same way as user FBS identifiers, each device registered reduces by one the number of user schedulers that can be configured. Therefore, depending upon system requirements, it may be necessary to increase the value of the system tunable parameter FBSMNI. Virtual FBS descriptors are not directly accessible to user programs.

Registering a device as a Coupled FBS timing device also creates a device file entry in the **/dev/rdev** file system on each host where the device is registered. This **/dev/rdev/<hostname/device<*n*>** path name may be specified on subsequent calls to **fbsattach**.

A device may not be registered as a Coupled FBS timing device if a FBS scheduler is already directly attached to that device.

The FORTRAN variable declarations, CALL statement and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER type, num_hosts, istat
CHARACTER *(*) device_name
CHARACTER *(*) rdevfs_name
CHARACTER *(*) hostname_array(num_hosts)

Call Statement

CALL fbs_register_rdev(device_name, rdevfs_name, type,
num_hosts, hostname_array, istat)

Parameters

device_name — Refers to a caller-specified string that contains the path name of the device to be registered.

If the device is a real-time clock or edge-triggered interrupt, then refer to Chapter 3 for detailed information about these path names and their associated attributes.

If the device is a user-supplied device, the path name must be a valid UNIX path name, and the device must support the IOCTLVECNUM **ioctl(2)** call. See Chapter 3 for additional information.

rdevfs_name — Refers to a variable in which **fbs_register_rdev** will return the path name within the **rdevfs(4)** filesystem that corresponds to this Coupled FBS timing device registration, where the **rdevfs** path name will have the form of **/dev/rdev/<hostname>/device<*n*>**. This returned path name should be used on subsequent fbsattach calls to attach FBS schedulers to this Coupled FBS timing device.

type — Refers to a variable where the caller specifies the type of Coupled FBS timing device to be registered. The type variable may be: **1** to indicate a Closely-Coupled timing device, or **2** to indicate a RCIM Coupled timing device. See Chapter 3 or the **fbs_register_rdev(3F77rt)** system manual page for information about these two types of timing devices.

num_hosts — Refers to a variable where the caller specifies the number of hostnames in the **hostname_array**.

hostname_array — Refers to a variable that contains an array of caller-specified hostname strings that denote the hosts where the Coupled FBS timing device is to be registered. A host name that corresponds to the local host **must** be included in this array. Note that while alias hostnames may be specified in this array, each host should only appear once in this array.

istat — Refers to a variable in which **fbs_register_rdev** will return an integer value indicating whether or not an error has occurred. A non-zero value indicates that an error of a specific type has occurred. Refer to the system manual page **fbs_register_rdev(3F77rt)** for a listing of the non-zero values that may be returned and the types of errors that they represent.

Fbs_unregister_rdev - Unregister a Coupled FBS timing device

This routine may be called to unregister a local device that was previously registered as a Coupled FBS timing device. To unregister a device, the calling process must have the P_RTIME privilege as well as enough privilege to open the device file.

Unregistering a device from being a Coupled FBS timing device results in the removal of the virtual FBS identifier that was created when the device was initially registered. The unregistration also removes the corresponding **/dev/rdev/<hostname>/device<*n*> rdevfs** file system device file entry on each host where the device was previously registered.

Once a device is unregistered, it may once again be directly attached to an FBS scheduler on the local system as a normal, non-Coupled FBS timing device, or, it may be re-registered as a Coupled FBS timing device with the **fbs_register_rdev(3rt)** function.

The FORTRAN variable declarations, CALL statement and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER istat
CHARACTER *(*) device_name

Call Statement

CALL fbs_unregister_rdev(device_name, istat)

Parameters

device_name	Refers to a caller-specified string that contains the path name of the device to be unregistered. Device_name should contain the same path name that was specified on the previous corresponding **fbs_register_rdev** call.
istat	Refers to a variable which **fbs_unregister_rdev** will return an integer value indicating whether or not an error has occurred. A non-zero value indicates that an error of a specific type has occurred. Refer to the system manual page **`fbs_unregister_rdev(3F77rt)`** for a listing of the non-zero values that may be returned and the types of errors that they represent.

Fbs_register_cluster_device - Register cluster timing device

This routine is invoked to register a local device as a Closely-Coupled timing device in a Closely-Coupled system. To register a device, the calling process must have the P_RTIME privilege as well as enough privilege to open the device file.

Registering a Closely-Coupled timing device creates a placeholder, or virtual, FBS identifier to reserve the device's interrupt vector. There is one virtual FBS for each device registered and a virtual FBS provides the means for a process on another SBC to communicate with the real device. Because the virtual FBS is allocated exactly the same way as user FBS identifiers, each device registered reduces by one the number of user schedulers that can be configured. Therefore, some thought should be given to increasing the value of the system tunable parameter FBSMNI. Virtual FBS descriptors are not directly accessible to user programs.

Registering a device as a Closely-Coupled timing source also creates entries in the **`/dev/rdev`** directories on all SBCs in the VME cluster. These entries can be specified on a subsequent call to **`fbsattach`**.

A device can either be registered as a Closely-Coupled timing device or be attached to an FBS, but not both at the same time.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

CHARACTER* (*) device_name
CHARACTER* (*) rdevfs_name
INTEGER istat

CALL Statement

CALL fbs_register_cluster_device(device_name, rdevfs_name, istat);

Parameters

Parameters are described as follows.

device_name — refers to a variable that contains the path name of the device that is to be registered as a Closely-Coupled timing source. The device_name may refer to a real-time clock or to a user-supplied device.

If the device is a real-time clock, the path name must be of a certain form. Refer to Chapter 3 for detailed information on the form associated with the real-time clock.

If the device is a user-supplied device, the path name must be a valid UNIX path name. The device Chapter 3 must support the **IOCTLVECNUM ioctl(2)** call. See for additional information.

rdevfs_name — refers to a variable to which fbs_register_cluster_device will return the path name within the **rdev(4)** file system that corresponds to this Closely-Coupled timing device registration. This returned path name should used on subsequent **fbsattach** calls to attach schedulers to this Closely-Coupled timing device.

istat — refers to a variable to which fbs_register_cluster_device will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error has occurred. Refer to the system manual page **fbs_register_cluster_device(3F77rt)** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Note

The **fbs_register_cluster_device** function call is obsolete. It is being supported only for providing backward compatibility with previous PowerMAX OS releases. Users are highly encouraged to make use of the newer **fbs_register_rdev** function call. Note that **fbs_register_cluster_device** only supports the registration of Closely-Coupled timing devices, while the **fbs_register_rdev** function supports both Closely-Coupled and RCIM Coupled timing device registrations.

Fbs_unregister_cluster_device - Unregister cluster timing device

This routine is invoked to unregister a local device as a Closely-Coupled timing device in a closely-coupled system. To unregister a device, the calling process must have the P_RTIME privilege as well as enough privilege to open the device file.

Unregistering a device as a Closely-Coupled timing source removes the virtual FBS identifier created when the device was registered and also removes the **/dev/rdev** entries on all SBCs in the VME cluster. Once a device is unregistered, it is once again available to be attached to an FBS on the local SBC.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

CHARACTER* (*) device_name
INTEGER istat

CALL Statement

CALL fbs_unregister_cluster_device(device_name, istat);

Parameters

Parameters are described as follows.

device_name
: Refers to a variable that contains the path name of the device that is to be unregistered as a Closely-Coupled timing source. device_name may refer to a real-time clock or to a user-supplied device.

  If the device is a real-time clock, the path name must be of a certain form. Refer to Chapter 3 for detailed information on the form associated with the real-time clock.

  If the device is a user-supplied device, the path name must be a valid UNIX path name. The device must support the **`IOCTLVECNUM ioctl(2)`** call. See Chapter 3 for additional information.

ristat
: refers to a variable to which fbs_unregister_cluster_device will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error has occurred. Refer to the system manual page **`fbs_unregister_cluster_device(3F77rt)`** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Note

The **fbs_unregister_cluster_device** function call is obsolete. It is being supported only for providing backward compatibility with previous PowerMAX OS releases. Users are highly encouraged to make use of the newer **fbs_unregister_rdev** function call.

Pgmquery – Query a Process on an FBS

NOTE

The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency–based scheduler process identifier.

Information that is returned includes the following:

- The process’s path name
- The CPU on which the process can execute
- The frequency–based scheduler process identifier

Pgmquery – Query a Process on an FBS

- The scheduling priority
- The period (the number of minor cycles indicating the frequency with which the process is wakened in each major frame)
- The starting base cycle (the first minor cycle in which the process is scheduled to be wakened in each major frame)
- The value of the “halt on overrun” flag

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

```
INTEGER schdle
CHARACTER* (*) name
INTEGER cpu
INTEGER slot
INTEGER prior
INTEGER period
INTEGER cycle
INTEGER ab
INTEGER istat
```

CALL Statement

```
CALL pgmquery(schdle, name, cpu, slot, prior, period, cycle, ab, istat)
```

Parameters

Parameters are described as follows.

schdle refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process for which you wish to obtain scheduling information has been scheduled. You can obtain this value by making a call to **fbsconfigure** (see page 8-6 for an explanation of this subroutine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

name refers to a variable that contains a standard UNIX path name identifying the process for which information is to be returned. A full or relative path name of up to 1024 characters can be specified. If this variable contains blanks, you must provide the frequency–based scheduler process identifier in the *slot* parameter.

cpu refers to a variable that contains an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the program for which information is to be returned. Acceptable values and corresponding results are presented in Table 8-11.

Table 8-11. CPU Options: pgmquery

Value	Result
0	The first process named by *name* that is currently running on the processor from which the call is made is specified
-1	The first process named by *name* that is currently running on any processor is specified

Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is running on CPU *i* is specified
	
	If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is specified

slot refers to a variable that contains an integer value providing the unique frequency–based scheduler process identifier for the process for which information is to be returned. This value is obtained when you make a call to **pgmschedule** (see page 8-45 for an explanation of this subroutine). This value must be – **1** if you wish to identify the program to be queried only by specifying *name* and *cpu*.

prior refers to a variable to which **pgmquery** will return an integer value indicating the specified process's scheduling priority.

period	refers to a variable to which **`pgmquery`** will return an integer value indicating the frequency with which the specified program is to be wakened in each major frame.
cycle	refers to a variable to which **`pgmquery`** will return an integer value indicating the first minor cycle in which the specified process is scheduled to be wakened in each frame
ab	refers to a variable to which **`pgmquery`** will return an integer value indicating the value of the "halt on overrun" flag. A nonzero value indicates that the flag is set. A value of zero indicates that the flag is not set.
istat	refers to a variable to which **`pgmquery`** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page **`pgmquery(3F77rt)`** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Pgmremove – Remove a Process from an FBS

This subroutine is invoked to remove a process from a frequency–based scheduler. You can identify the process that you wish to remove by using one of the following methods:

- Specify the name of the process and the CPU on which it is scheduled.
- Specify the process's frequency–based scheduler process identifier (slot number).
- Specify the name of the process, the CPU on which it is scheduled, and its frequency–based scheduler process identifier.

NOTE

The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency–based scheduler process identifier.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

```
INTEGER schdle
CHARACTER* (*) name
INTEGER cpu
INTEGER slot
```

INTEGER ab
INTEGER istat

CALL Statement

CALL pgmremove(schdle, name, cpu, slot, ab, istat)

Parameters

Parameters are described as follows.

schdle — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process is scheduled. You can obtain this value by making a call to **`fbsconfigure`** (see page 8-6 for an explanation of this subroutine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **`1`**.

name — refers to a variable that contains a standard UNIX path name identifying the process to be removed from the specified scheduler. A full or relative path name of up to 1024 characters can be specified. If this variable contains blanks, you must provide the frequency–based scheduler process identifier in the *slot* parameter.

cpu — refers to a variable that contains an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the process to be removed from the specified scheduler. Acceptable values and corresponding results are presented in Table 8-12.

Table 8-12. CPU Options: pgmremove

Value	Result
0	The first process named by *name* that is currently running on the processor from which the call is made is removed
-1	The first process named by *name* that is currently running on any processor is removed

Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is running on CPU *i* is removed
	
	If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is removed

slot — refers to a variable that contains an integer value providing the unique frequency–based scheduler process identifier for the process to be removed from the specified scheduler. This value is obtained when you make a call to **schedpgmadd** (see page 8-57 for an explanation of this subroutine). This value must be – **1** if you choose to identify the program to be removed only by specifying *name* and *cpu*.

ab — refers to a flag that contains an integer value indicating the manner in which the specified process is be removed from the specified scheduler. A positive value indicates that the process is to be removed from the scheduler but allowed to continue executing. A negative value indicates that the process is to be removed from the scheduler and terminated.

istat — refers to a variable to which **pgmremove** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page **pgmremove(3F77rt)** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Pgmreschedule – Reschedule a Process

CAUTION

This interface is obsolete. It is maintained for compatibility with CX/UX, but its behavior with respect to specification of a process's scheduling priority has changed. If you have an existing application that uses this interface, it is recommended that you change your application to use **schedpgmresched(3F77rt)** (see p. 8-63). For details on obsolete interfaces, refer to Chapter 2, "Overview of the FBS."

This subroutine is invoked to change the scheduling parameters for a process that is scheduled on a frequency–based scheduler. You may wish, for example, to change a program's priority or the frequency with which it is scheduled to run. You cannot, however, change the CPU on which it has been scheduled.

To change a process's priority, the following conditions must be met:

- The calling process must have the P_RTIME privilege.
- The effective user ID of the calling process must match the effective user ID of the target process (the process for which the scheduling policy and priority are being set), or the calling process must have the P_OWNER privilege.

If the Enhanced Security Utilities are installed and running, the following additional conditions must be met:

- The Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

For additional information on privileges, refer to the *PowerMAX OS Programming Guide* and the **intro(2)** system manual page.

You can call **pgmreschedule** to change the parameters without having called **pgmremove** to remove the process from the scheduler (see page 8-39) or **fbsintrpt** to stop the simulation (see page 8-18).

You can identify the process that you wish to reschedule by using one of the following methods:

- Specify the name of the process and the CPU on which it is scheduled.
- Specify the process's frequency–based scheduler process identifier (slot number).
- Specify the name of the process, the CPU on which it is scheduled, and its frequency–based scheduler process identifier.

NOTE

The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency–based scheduler process identifier.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

```
INTEGER schdle
CHARACTER* (*) name
INTEGER cpu
INTEGER slot
INTEGER prior
INTEGER param
INTEGER period
INTEGER cycle
INTEGER ab
INTEGER istat
```

CALL Statement

```
CALL pgmreschedule(schdle, name, cpu, slot, prior, param, period, cycle, ab, istat)
```

Parameters

Parameters are described as follows.

schdle — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process is scheduled. You can obtain this value by making a call to **`fbsconfigure`** (see page 8-6 for an explanation of this subroutine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **`1`**.

name — refers to a variable that contains a standard UNIX path name identifying the process to be rescheduled. A full or relative path name of up to 1024 characters can be specified. If this variable contains blanks, you must provide the frequency–based scheduler process identifier in the *slot* parameter.

cpu — refers to a variable that contains an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the process to be rescheduled. Acceptable values and corresponding results are presented in Table 8-13.

Table 8-13. CPU Options: pgmreschedule

Value	Result
0	The first process named by *name* that is currently running on the processor from which the call is made is rescheduled
-1	The first process named by *name* that is currently running on any processor is rescheduled

Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is running on CPU *i* is rescheduled
	
	If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is rescheduled

slot — refers to a variable that contains an integer value providing the unique frequency–based scheduler process identifier for the process to be rescheduled. This value is obtained when you make a call to **`pgmschedule`** (see page 8-45 for an explanation of this subroutine). This value must be – **`1`** if you wish to identify the program to be rescheduled only by specifying *name* and *cpu*.

prior — an integer value indicating the specified process's scheduling priority. A process that has been scheduled using **`pgm-`**

schedule (see p. 8-45 for an explanation of this subroutine) is scheduled under the POSIX **SCHED_RR** scheduling policy. The value specified must lie in the range of priorities associated with this policy. You can obtain the allowable range of priorities by invoking the **run(1)** command from the shell and not specifying any options or arguments (see the corresponding system manual page for an explanation of this command). Higher numerical values correspond to more favorable scheduling priorities.

For complete information on scheduling policies and priorities, refer to the "Process Scheduling and Management" chapter of the *PowerMAX OS Programming Guide*.

param — refers to a variable that contains an integer value to be passed to a process that is scheduled on a frequency–based scheduler.

period — refers to a variable that contains an integer value indicating the frequency with which the specified program is to be wakened in each major frame. A period of one indicates that the specified program is to be wakened every minor cycle; a period of two indicates that it is to be wakened once every two minor cycles, a period of three once every three minor cycles, and so on. This value can range from one to the number of minor cycles that compose a frame on the specified scheduler as defined in a call to **fbsconfigure** (see page 8-6).

cycle — refers to a variable that contains an integer value indicating the first minor cycle in which the specified process is scheduled to be wakened in each frame. This value can range from zero to the total number of minor cycles per frame minus one. The total number of minor cycles per frame is specified in a call to **fbsconfigure** (see page 8-6 for an explanation of this subroutine).

ab — refers to a flag that contains an integer value indicating whether or not the scheduler should be stopped in the event that the specified process causes a frame overrun. A nonzero value indicates that the scheduler will be stopped.

istat — refers to a variable to which **pgmreschedule** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page **pgmreschedule(3F77rt)** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Pgmschedule – Schedule a Process on an FBS

CAUTION

This interface is obsolete. It is maintained for compatibility with CX/UX, but its behavior with respect to specification of a process's scheduling priority has changed. If you have an existing application that uses this interface, it is recommended that you change your application to use **schedpgmadd(3F77rt)** (see p. 8-57). For details on obsolete interfaces, refer to Chapter 2, "Overview of the FBS."

This subroutine is invoked to create a new process and schedule it on a frequency–based scheduler. When a process is scheduled using this subroutine, it is scheduled under the POSIX **SCHED_RR** scheduling policy (for complete information on scheduling policies and priorities, refer to the "Process Scheduling and Management" chapter of the *PowerMAX OS Programming Guide)*. Note that a process can not be scheduled under this policy on a CPU on which servicing of the 60 Hz clock interrupt has been disabled. In such cases, the process will behave as though it were scheduled under the **SCHED_FIFO** policy.

If you wish to set the process's scheduling priority, the following conditions must be met:

- The calling process must have the P_RTIME privilege.
- The effective user ID of the calling process must match the effective user ID of the target process (the process for which the scheduling policy and priority are being set), or the calling process must have the P_OWNER privilege.

If the Enhanced Security Utilities are installed and running, the following additional conditions must be met:

- The Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

If you wish to modify the process's CPU bias when you invoke this subroutine, the following conditions must be met:

- The real or effective user ID of the calling process must match the real or saved user ID of the process for which the CPU assignment is being changed, or the calling process must have the P_OWNER privilege.
- To add a CPU to a process's CPU bias, the calling process must have the P_CPUBIAS privilege.

If the Enhanced Security Utilities are installed and running, the following additional conditions must be met:

- The Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

For additional information on privileges, refer to the *PowerMAX OS Programming Guide* and the **intro(2)** system manual page.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER schdle
CHARACTER* (*) name
INTEGER prior
INTEGER param
INTEGER period
INTEGER cycle
INTEGER ab
INTEGER cpu
INTEGER slot
INTEGER istat

CALL Statement

CALL pgmschedule(schdle, name, prior, param, period, cycle, ab, cpu, slot, istat)

Parameters

Parameters are described as follows.

schdle
: refers to a variable that contains a unique, positive integer value representing the identifier for a frequency–based scheduler. You can obtain this value by making a call to **fbsconfigure** (see page 8-6 for an explanation of this subroutine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing the identifier, you can specify the value **–1**.

name
: refers to a variable that contains a standard UNIX path name identifying the program to be scheduled on the scheduler. A full or relative path name of up to 1024 characters can be specified.

prior
: an integer value indicating the specified process's scheduling priority. A process that is scheduled using **pgmschedule** is scheduled under the POSIX **SCHED_RR** scheduling policy. The value specified must lie in the range of priorities associated with this policy. You can obtain the allowable range of priorities by invoking the **run(1)** command from the shell and not specifying any options or arguments (see the corresponding system manual page for an explanation of this command). Higher numerical values correspond to more favorable scheduling priorities.

: For complete information on scheduling policies and priorities, refer to the "Process Scheduling and Management" chapter of the *PowerMAX OS Programming Guide*.

param — refers to a variable that contains an integer value to be passed to a process that is scheduled on a frequency–based scheduler. This value can be retrieved by the FBS–scheduled process through a call to **`rtparm`** (see page 8-52 for an explanation of this subroutine).

period — refers to a variable that contains an integer value indicating the frequency with which the specified program is to be wakened in each major frame. A period of one indicates that the specified program is to be wakened every minor cycle; a period of two indicates that it is to be wakened once every two minor cycles, a period of three once every three minor cycles, and so on. This value can range from one to the number of minor cycles that compose a frame on the specified scheduler as defined in a call to **`fbsconfigure`** (see page 8-6).

cycle — refers to a variable that contains an integer value indicating the first minor cycle in which the specified program is scheduled to be wakened in each frame. This value can range from zero to the total number of minor cycles per frame minus one. (The total number of minor cycles per frame is specified in a call to **`fbsconfigure`**. See page 8-6 for an explanation of this subroutine.)

ab — refers to a flag that contains an integer value indicating whether or not the scheduler should be stopped in the event that the specified program causes a frame overrun. A non-zero value indicates that the scheduler will be stopped.

cpu — refers to a mask that identifies the processors on which the specified program can be scheduled to run. Acceptable values and corresponding results are presented in Table 8-14.

Table 8-14. CPU Options: pgmschedule

Value	Result
0	The program specified by *name* can be scheduled on the processor from which the call is made
-1	The program specified by *name* can be scheduled on any processor
Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) the program specified by *name* can be scheduled on CPU *i*

slot — refers to a variable to which **`pgmschedule`** will return an integer value that is the unique frequency–based scheduler process identifier for the scheduled process.

istat — refers to a variable to which **`pgmschedule`** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page **pgmschedule(3F77rt)** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Pgmstat – Query State of FBS–Scheduled Process

This subroutine is invoked to obtain information about the state of a particular process that has been scheduled on a frequency–based scheduler. The state of the process indicates whether it is in the **fbswait** sleep state or is in another state.

You can identify the process by using one of the following methods:

- Specify the name of the process and the CPU(s) on which it is scheduled.
- Specify the process's frequency–based scheduler process identifier (slot number).
- Specify the name of the process, the CPU on which it is scheduled, and its frequency–based scheduler process identifier.

NOTE

The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency–based scheduler process identifier.

Information that is returned includes the following:

- The process's path name
- A mask of the CPU(s) on which the process can run
- The frequency–based scheduler process identifier
- The current state of the process

The FORTRAN specifications and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER schdle
CHARACTER* (*) name
INTEGER cpu
INTEGER slot
INTEGER state
INTEGER istat

CALL Statement

CALL pgmstat(schdle, name, cpu, slot, state, istat)

Parameters

Parameters are described as follows.

schdle	refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process for which you wish to obtain state information has been scheduled. You can obtain this value by making a call to **`fbsconfigure`** (see page 8-6 for an explanation of this subroutine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **`1`**.
name	refers to a variable that contains a standard UNIX path name identifying the process for which state information is to be returned. A full or relative path name of up to 1024 characters can be specified. If this variable contains blanks, you must provide the frequency–based scheduler process identifier in the *slot* parameter. **`Pgmstat`** will return to this variable the path name of the specified FBS–scheduled process.
cpu	refers to a variable that contains an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the program for which state information is to be returned. Acceptable values and corresponding results are presented in Table 8-15.

Table 8-15. CPU Options: pgmstat

Value	Result
0	The first process named by *name* that is currently running on the processor from which the call is made is specified
-1	The first process named by *name* that is currently running on any processor is specified

Bit mask	If (*cpu* & (1<<***`i`***)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is running on CPU *i* is specified
	
	If (*cpu* & (1<<***`i`***)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is specified

Pgmstat will return to this variable the mask of the CPUs on which the specified process can run.

slot
: refers to a variable that contains an integer value providing the unique frequency–based scheduler process identifier for the process for which status information is to be returned. This value is obtained when you make a call to **schedpgmadd** (see page 8-57 for an explanation of this subroutine). This value must be – **1** if you wish to identify the program to be queried only by specifying *name* and *cpu*. **Pgmstat** will return to this variable the frequency–based scheduler process identifier for the specified process.

state
: refers to a variable to which **pgmstat** will return an integer value indicating the current state of the specified process as defined in <**fbslib.h**.>

istat
: refers to a variable to which **pgmstat** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A non-zero value indicates that an error of a specific type has occurred. Refer to the system manual page **pgmstat(3F77rt)** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Pgmtrigger – Trigger Process Waiting on FBS

This subroutine enables a process to wake a process that is in the **`fbswait`** sleep state. It is important to note that the calling process does not have to be scheduled on a frequency–based scheduler; the target process must be.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER schdle
INTEGER slot
INTEGER tgrflg
INTEGER istat

CALL Statement

CALL pgmtrigger(schdle, slot, tgrflg, istat)

Parameters

Parameters are described as follows.

schdle — refers to a variable that contains a unique, positive integer value representing the identifier for a frequency–based scheduler on which the sleeping process is scheduled.

slot — refers to a variable that contains an integer value providing the unique frequency–based scheduler process identifier for the sleeping process. This value is obtained when you make a call to **`schedpgmadd`** (see page 8-57 for an explanation of this subroutine).

tgrflg — refers to a variable that contains an integer value indicating whether or not a context switch is to be forced on the processor on which the wakened process is executing. A nonzero value indicates that a context switch is to be forced.

istat — refers to a variable to which **`pgmtrigger`** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that the process is runnable. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page **`pgmtrigger(3F77rt)`** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Rtparm – Return Initiation Parameter

This subroutine enables a process that is scheduled on a frequency–based scheduler to obtain the value of a process initiation parameter that has been passed to it via a call to **schedpgmadd** (see page 8-57) or **schedpgmresched** (see 8-63).

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER param

CALL Statement

CALL rtparm(param)

Parameter

Rtparm requires one parameter: *param*. *Param* refers to a variable to which **rtparm** will return the integer value passed to the process via a call to **schedpgmadd** or **schedpgmresched**.

Sched_pgm_set_soft_overrun_limit

Sets the consecutive soft overrun limit for a currently scheduled LWP on a frequency-based scheduler.

To set the consecutive soft overrun limit, the calling LWP must have alter permission for the scheduler. If the Enhanced Security Utilities are installed and running, the Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the `P_MACWRITE` privilege.

The LWP can be identified in one of the following ways:

- A slot only (if name is blank).

- A path name and processor id pair only (if slot is -1).

- Both a slot and the path name and processor id pair.

The FORTRAN variable declarations, CALL statement, corresponding parameters and return values are presented in the following sections.

Variable Declarations

INTEGER schdle, cpu, slot, soft_limit, istat
CHARACTER* (*) name

CALL Statement

CALL sched_pgm_set_soft_overrun_limit (schdle, name, cpu, slot, soft_limit, istat)

Parameters

schdle   Obtained from an fbsid(3F77rt) library routine call or set to -1. -1 enables an FBS-scheduled LWP to reference the frequency-based scheduler on which it is scheduled without knowing the scheduler identifier.

name   Path name that identifies the LWP. If the name is all blanks, then the slot field (frequency-based scheduler process identifier) must be given.

cpu   Either a bit mask or set to 0 or -1. If a bit mask is specified, then those processors with (cpu & (1 << i)) set are requested. If cpu is 0, then the processor on which the call is made is requested. If cpu is -1, then all processors are requested. The first LWP named that is currently running on one of the requested processors has its soft overrun limit set.

slot   Frequency-based scheduler process identifier for the LWP. If the slot number equals -1, then a name and processor ID must be given.

soft_limit   Number of consecutive soft overruns allowed to occur before failure. soft_limit must be non-negative and must be less than INT_MAX. By default, this value is zero; i.e., if the LWP never sets a consecutive soft overrun limit, then it is zero.

istat   Variable to which sched_pgm_set_soft_overrun_limit returns an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page fbsid(3F77rt) for a listing of the nonzero values that may be returned and the types of errors that they represent.

Sched_pgm_soft_overrun_query

Queries the status of soft overrun processing for a currently scheduled LWP on a frequency-based scheduler. The LWP can be identified in one of the following ways:

- A slot only (if name is blank).

- A path name and processor id pair only (if slot is -1).

- Both a slot and the path name and processor id pair.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER schdle, cpu, slot, soft_limit, soft_total, istat
CHARACTER* (*) name

CALL Statement

CALL sched_pgm_soft_overrun_query (schdle, name, cpu, slot, soft_limit, soft_total,
    istat)

Parameters

schdle Obtained from an fbsid(3F77rt) library routine call or set to -1. -1 enables an FBS-scheduled LWP to reference the frequency-based scheduler on which it is scheduled without knowing the scheduler identifier.

name Path name that identifies the LWP. If the name is all blanks, then the slot field (frequency-based scheduler process identifier) must be given.

cpu Either a bit mask or set to 0 or -1. If a bit mask is specified, then those processors with (cpu & (1<<i)) set are requested. If cpu is 0, then the processor on which the call is made is requested. If cpu is -1, then all processors are requested. The first LWP named name that is currently running on one of the requested processors is returned.

slot Frequency-based scheduler process identifier for the LWP. If the slot number equals -1, then a name and processor id must be given.

soft_limit Number of consecutive soft overruns set by calling sched_pgm_set_soft_overrun_limit.

soft_total Total number of soft overruns incurred by the LWP.

istat Variable to which sched_pgm_set_soft_overrun_limit returns an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page fbsid(3F77rt) for a listing of the nonzero values that may be returned and the types of errors that they represent.

Schedfbsqry – Query Processes on an FBS

The **schedfbsqry** subroutine is invoked to obtain information about processes that have been scheduled on a frequency–based scheduler. Information is returned for all processes scheduled on the user–specified processor(s). Information provided for each process includes the following:

- A mask of the CPU(s) on which the process can execute
- The frequency–based scheduler process identifier
- The scheduling policy under which the process has been scheduled
- The scheduling priority
- The period (the number of minor cycles indicating the frequency with which the process is wakened in each major frame)
- The starting base cycle (the first minor cycle in which the process is scheduled to be wakened in each major frame)
- The value of the “halt on overrun” flag

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER schdle
INTEGER cpu
INTEGER buf1size
INTEGER buf1(buf1size)
INTEGER maxsize
INTEGER buf2size
CHARACTER* (*) buf2
INTEGER istat

CALL Statement

CALL schedfbsqry(schdle, cpu, buf1size, buf1, maxsize, buf2size, buf2, istat)

Parameters

Parameters are described as follows.

schdle
: refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler for which you wish to obtain scheduling information. You can obtain this value by making a call to **`fbsconfigure`** (see page 8-6 for an explanation of this subroutine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **`1`**.

cpu
: refers to a variable that contains an integer value indicating the processor(s) for which scheduling information is to be obtained. Acceptable values and corresponding results are presented in Table 8-16.

buf1size
: refers to a variable that contains an integer value indicating the size in 32–bit words of the array represented by *buf1*. Because 9 words of information are returned for each process, it is recommended that this value be a multiple of 9.

Table 8-16. CPU Options: schedfbsqry

Value	Result
0	Scheduling information for processes executing on the processor from which the call is made is returned
-1	Scheduling information for all processes on the scheduler is returned
Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU), scheduling information for processes executing on CPU *i* is returned

buf1 refers to an array to which **schedfbsqry** will return a series of 11 integer values for each process on the processor(s) specified with the *cpu* parameter. The number of processes for which these values are returned is bound by the value of the *buf1size* parameter. If, for example, the value of *buf1size* is 145, values for 16 processes will be returned. These values represent the scheduling information for the process(es). The type of information returned in each array element for a single process is presented in Table 8-17.

Table 8-17. Contents of Array Elements for a Process

Element	Contents
1	Byte offset of the process's path name in *buf2*
2	Length in bytes of the process's path name
3	Mask of the CPU(s) on which the process can execute
4	The process's frequency–based scheduler process identifier
5	The process's scheduling policy
6	The process's scheduling priority
7	The number of minor cycles indicating the frequency with which the process is to be wakened in each major frame (period)
8	The first minor cycle in which the process is scheduled to be wakened in each major frame (starting base cycle)
9	The value of the "halt on overrun" flag. A nonzero value indicates that the flag is set. A value of zero indicates that the flag is not set.

maxsize refers to a variable that contains an integer value indicating the maximum length of a path name to be returned in *buf2*

buf2size refers to a variable that contains an integer value indicating the size in bytes of the character string represented by *buf2*. To ensure that *buf2* is large enough to accommodate the names of all processes that you wish to query, you may find it helpful to compute the number of bytes needed by multiplying the maximum number of processes allowed on the scheduler (see the information on **fbsconfigure** presented on page 8-6) by 32.

buf2 refers to a variable to which **schedfbsqry** will return the path names for each process on the processor(s) specified with the *cpu* parameter. Path names are returned as a series of strings. The length of each string is less than or equal to the value of *maxsize*. Where *maxsize* is not large enough to accommodate a full path name, the concluding component names are returned. The number of path names returned is bound by the value of the *buf2size* parameter.

istat — refers to a variable to which **schedfbsqry** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page **schedfbsqry(3F77rt)** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Schedpgmadd – Schedule a Process on an FBS

The **schedpgmadd** subroutine is invoked to create a new process and schedule it on a frequency–based scheduler. It is important to note that to use this subroutine to (1) change a process's scheduling policy to the **SCHED_FIFO** or the **SCHED_RR** policy or (2) change the priority of a process scheduled under the **SCHED_FIFO** or the **SCHED_RR** policy, the following conditions must be met:

- The calling process must have the P_RTIME privilege.
- The effective user ID of the calling process must match the effective user ID of the target process (the process for which the scheduling policy and priority are being set), or the calling process must have the P_OWNER privilege.

If the Enhanced Security Utilities are installed and running, the following additional conditions must be met:

- The Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

If you wish to raise the priority of a process scheduled under the **SCHED_OTHER** policy above a per-process or LWP limit, the following conditions must be met:

- The calling process must have the P_TSHAR privilege.
- The effective user ID of the calling process must match the effective user ID of the target process (the process for which the scheduling priority is being set), or the calling process must have the P_OWNER privilege.

If the Enhanced Security Utilities are installed and running, the following additional conditions must be met:

- The Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

If you wish to modify the process's CPU bias when you invoke this subroutine, the following conditions must be met:

- The real or effective user ID of the calling process must match the real or saved user ID of the process for which the CPU assignment is being changed, or the calling process must have the P_OWNER privilege.
- To add a CPU to a process's CPU bias, the calling process must have the P_CPUBIAS privilege.

If the Enhanced Security Utilities are installed and running, the following additional conditions must be met:

- The Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

For additional information on privileges, refer to the *PowerMAX OS Programming Guide* and the **`intro(2)`** system manual page.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER schdle
CHARACTER* (*) name
INTEGER cid
INTEGER prior
INTEGER param
INTEGER period
INTEGER cycle
INTEGER ab
INTEGER cpu
INTEGER slot
INTEGER istat

CALL Statement

CALL schedpgmadd(schdle, name, cid, prior, param, period, cycle, ab, cpu, slot, istat)

Parameters

Parameters are described as follows.

schdle — refers to a variable that contains a unique, positive integer value representing the identifier for a frequency–based scheduler. You can obtain this value by making a call to **`fbsconfigure`** (see page 8-6 for an explanation of this subroutine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing the identifier, you can specify the value – **`1`**.

name — refers to a variable that contains a standard UNIX path name identifying the program to be scheduled on the scheduler. A full or relative path name of up to 1024 characters can be specified.

cid — refers to a variable that contains an integer value indicating the POSIX scheduling policy under which the specified process is to be scheduled. Scheduling policies are defined in the file <**sched.h**>. The value of *cid* must be one of the following:

SCHED_FIFO
first–in–first–out (FIFO) scheduling policy

SCHED_RR
round–robin (RR) scheduling policy. Note that a process cannot be scheduled under this policy on a CPU on which servicing of the 60 Hz clock interrupt has been disabled. In such cases, the process will behave as though it were scheduled under the **SCHED_FIFO** policy.

SCHED_OTHER
time-sharing scheduling policy

prior — refers to a variable that contains an integer value indicating the scheduling priority of the specified program. The range of acceptable priority values is governed by the scheduling policy specified.

You can determine the allowable range of priorities associated with each policy (**SCHED_FIFO**, **SCHED_RR**, or **SCHED_OTHER**) by invoking the **run(1)** command from the shell and not specifying any options or arguments (see the corresponding system manual page for an explanation of this command). Higher numerical values correspond to more favorable priorities.

For complete information on scheduling policies and priorities, refer to the “Process Scheduling and Management” chapter of the *PowerMAX OS Programming Guide*.

param — refers to a variable that contains an integer value to be passed to a process that is scheduled on a frequency–based scheduler. This value can be retrieved by the FBS–scheduled process through a call to **rtparm** (see page 8-52 for an explanation of this subroutine).

period — refers to a variable that contains an integer value indicating the frequency with which the specified program is to be wakened in each major frame. A period of one indicates that the specified program is to be wakened every minor cycle; a period of two indicates that it is to be wakened once every two minor cycles, a period of three once every three minor cycles, and so on. This value can range from one to the number of minor cycles that compose a frame on the specified scheduler as defined in a call to **fbsconfigure** (see page 8-6).

cycle — refers to a variable that contains an integer value indicating the first minor cycle in which the specified program is scheduled to be wakened in each frame. This value can range from zero to the total number of minor cycles per frame minus one. (The total number of minor cycles per frame is specified in a call to **`fbsconfigure`**. See page 8-6 for an explanation of this subroutine).

ab — refers to a flag that contains an integer value indicating whether or not the scheduler should be stopped in the event that the specified program causes a frame overrun. A non-zero value indicates that the scheduler will be stopped.

cpu — refers to a mask that identifies the processors on which the specified program can be scheduled to run. Acceptable values and corresponding results are presented in Table 8-18.

Table 8-18. CPU Options: schedpgmadd

Value	Result
0	The program specified by *name* can be scheduled on the processor from which the call is made
-1	The program specified by *name* can be scheduled on any processor
Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) the program specified by *name* can be scheduled on CPU *i*

slot — refers to a variable to which **`schedpgmadd`** will return an integer value that is the unique frequency–based scheduler process identifier for the scheduled process.

istat — refers to a variable to which **`schedpgmadd`** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page **`schedpgmadd(3F77rt)`** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Schedpgmqry – Query a Process on an FBS

The **`schedpgmqry`** subroutine is invoked to obtain information for a particular process that has been scheduled on a frequency–based scheduler. You can identify the process by using one of the following methods:

- Specify the name of the process and the CPU(s) on which it is scheduled.

- Specify the process's frequency–based scheduler process identifier (slot number).
- Specify the name of the process, the CPU on which it is scheduled, and its frequency–based scheduler process identifier.

NOTE

The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency–based scheduler process identifier.

Information that is returned includes the following:

- The process's path name
- The CPU on which the process can execute
- The frequency–based scheduler process identifier
- The scheduling policy
- The scheduling priority
- The period (the number of minor cycles indicating the frequency with which the process is wakened in each major frame)
- The starting base cycle (the first minor cycle in which the process is scheduled to be wakened in each major frame)
- The value of the "halt on overrun" flag

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

```
INTEGER schdle
CHARACTER* (*) name
INTEGER cpu
INTEGER slot
INTEGER cid
INTEGER prior
INTEGER period
INTEGER cycle
INTEGER ab
INTEGER istat
```

CALL Statement

```
CALL schedpgmqry(schdle, name, cpu, slot, cid, prior, period, cycle, ab, istat)
```

Parameters

Parameters are described as follows.

schdle — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process for which you wish to obtain scheduling information has been scheduled. You can obtain this value by making a call to **fbsconfigure** (see page 8-6 for an explanation of this subroutine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

name — refers to a variable that contains a standard UNIX path name identifying the process for which information is to be returned. A full or relative path name of up to 1024 characters can be specified. If this variable contains blanks, you must provide the frequency–based scheduler process identifier in the *slot* parameter.

cpu — refers to a variable that contains an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the program for which information is to be returned. Acceptable values and corresponding results are presented in Table 8-19.

Table 8-19. CPU Options: schedpgmqry

Value	Result
0	The first process named by *name* that is currently running on the processor from which the call is made is specified
-1	The first process named by *name* that is currently running on any processor is specified

Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is running on CPU *i* is specified
	
	If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is specified

slot — refers to a variable that contains an integer value providing the unique frequency–based scheduler process identifier for the process for which information is to be returned. This value is obtained when you make a call to **schedpgmadd** (see page 8-57 for an explanation of this subroutine). This value must be – **1** if you wish to identify the program to be queried only by specifying *name* and *cpu*.

cid — refers to a variable to which **schedpgmqry** will return an integer value indicating the scheduling policy under which the specified process has been scheduled prior — refers to a variable to which **schedpgmqry** will return an integer value indicating the specified process's scheduling priority

period — refers to a variable to which **schedpgmqry** will return an integer value indicating the frequency with which the specified program is to be wakened in each major frame.

cycle — refers to a variable to which **schedpgmqry** will return an integer value indicating the first minor cycle in which the specified process is scheduled to be wakened in each frame

ab — refers to a variable to which **schedpgmqry** will return an integer value indicating the value of the "halt on overrun" flag. A nonzero value indicates that the flag is set. A value of zero indicates that the flag is not set.

istat — refers to a variable to which **schedpgmqry** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page **schedpgmqry(3F77rt)** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Schedpgmresched – Reschedule a Process

The **schedpgmresched** subroutine is invoked to change the scheduling parameters for a process that is scheduled on a frequency–based scheduler. You may wish, for example, to change a program's scheduling policy or priority or the frequency with which it is scheduled to run. You cannot, however, change the CPU on which it has been scheduled.

If you wish to (1) change a process's scheduling policy to the **SCHED_FIFO** or the **SCHED_RR** policy or (2) change the priority of a process scheduled under the **SCHED_FIFO** or the **SCHED_RR** policy, the following conditions must be met:

- The calling process must have the P_RTIME privilege.
- The effective user ID of the calling process must match the effective user ID of the target process (the process for which the scheduling policy and priority are being set), or the calling process must have the P_OWNER privilege.

If the Enhanced Security Utilities are installed and running, the following additional conditions must be met:

- The Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

If you wish to raise the priority of a process scheduled under the **SCHED_OTHER** policy above a per-process or LWP limit, the following conditions must be met:

- The calling process must have the P_TSHAR privilege.
- The effective user ID of the calling process must match the effective user ID of the target process (the process for which the scheduling priority is being set), or the calling process must have the P_OWNER privilege.

If the Enhanced Security Utilities are installed and running, the following additional conditions must be met:

- The Mandatory Access Control (MAC) level of the calling process must equal the MAC level of the target process, or the calling process must have the P_MACWRITE privilege.

For additional information on privileges, refer to the *PowerMAX OS Programming Guide* and the **`intro(2)`** system manual page.

You can identify the process that you wish to reschedule by using one of the following methods:

- Specify the name of the process and the CPU on which it is scheduled.
- Specify the process's frequency–based scheduler process identifier (slot number).
- Specify the name of the process, the CPU on which it is scheduled, and its frequency–based scheduler process identifier.

NOTE

The only method that can be used to identify a process that has been scheduled multiple times on the same CPU is to specify its frequency–based scheduler process identifier.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER schdle
CHARACTER* (*) name
INTEGER cpu
INTEGER slot
INTEGER cid
INTEGER prior
INTEGER param
INTEGER period
INTEGER cycle
INTEGER ab
INTEGER istat

CALL Statement

CALL schedpgmresched(schdle, name, cpu, slot, cid, prior, param, period, cycle, ab, istat)

Parameters

Parameters are described as follows.

schdle
: refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process is scheduled. You can obtain this value by making a call to **`fbsconfigure`** (see page 8-6 for an explanation of this subroutine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **`1`**.

name
: refers to a variable that contains a standard UNIX path name identifying the process to be rescheduled. A full or relative path name of up to 1024 characters can be specified. If this variable contains blanks, you must provide the frequency–based scheduler process identifier in the *slot* parameter.

cpu
: refers to a variable that contains an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the process to be rescheduled. Acceptable values and corresponding results are presented in Table 8-20.

Table 8-20. CPU Options: schedpgmresched

Value	Result
0	The first process named by *name* that is currently running on the processor from which the call is made is rescheduled
-1	The first process named by *name* that is currently running on any processor is rescheduled

Bit mask	If (*cpu &* (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is running on CPU *i* is rescheduled
	
	If (*cpu &* (1<<*i*)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is rescheduled

slot
: refers to a variable that contains an integer value providing the unique frequency–based scheduler process identifier for the process to be rescheduled. This value is obtained when you make a call to **`schedpgmadd`** (see page 8-57 for an explanation of this subroutine). This value must be –1 if you wish to identify the program to be rescheduled only by specifying *name* and *cpu*.

cid — refers to a variable that contains an integer value indicating the scheduling policy under which the specified program is to be scheduled. Scheduling policies are defined in the file <**sched.h**>. The value of *cid* must be one of the following:

SCHED_FIFO
first–in–first–out (FIFO) scheduling policy

SCHED_RR
round–robin (RR) scheduling policy. Note that a process cannot be scheduled under this policy on a CPU on which servicing of the 60 Hz clock interrupt has been disabled. In such cases, the process will behave as though it were scheduled under the **SCHED_FIFO** policy.

SCHED_OTHER
time-sharing scheduling policy

prior — refers to a variable that contains an integer value indicating the scheduling priority of the specified program. The range of acceptable priority values is governed by the scheduling policy specified.

You can determine the allowable range of priorities associated with each policy (**SCHED_FIFO**, **SCHED_RR**, or **SCHED_OTHER**) by invoking the **run(1)** command from the shell and not specifying any options or arguments (see the corresponding system manual page for an explanation of this command). Higher numerical values correspond to more favorable priorities.

For complete information on scheduling policies and priorities, refer to the “Process Scheduling and Management” chapter of the *PowerMAX OS Programming Guide*.

param — refers to a variable that contains an integer value to be passed to a process that is scheduled on a frequency–based scheduler.

period — refers to a variable that contains an integer value indicating the frequency with which the specified program is to be wakened in each major frame. A period of one indicates that the specified program is to be wakened every minor cycle; a period of two indicates that it is to be wakened once every two minor cycles, a period of three once every three minor cycles, and so on. This value can range from one to the number of minor cycles that compose a frame on the specified scheduler as defined in a call to **fbsconfigure** (see page 8-6).

cycle — refers to a variable that contains an integer value indicating the first minor cycle in which the specified process is scheduled to be wakened in each frame. This value can range from zero to the total number of minor cycles per frame minus one. The total number of minor cycles per frame is specified in a call to **fbsconfigure** (see page 8-6 for an explanation of this subroutine).

ab — refers to a flag that contains an integer value indicating whether or not the scheduler should be stopped in the event that the specified process causes a frame overrun. A nonzero value indicates that the scheduler will be stopped.

istat — refers to a variable to which **schedpgmresched** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page **schedpgmresched(3F77rt)** for a listing of the nonzero values that may be returned and the types of errors that they represent.

The Performance Monitor Subroutines

The performance monitor subroutines provide access to the key features of the performance monitor. They enable you to perform such basic operations as the following: (1) clear performance monitor values for a process or processor, (2) start and stop performance monitoring for a process or processor, and (3) obtain performance monitor values for a process or processor.

In the sections that follow, all of the performance monitor subroutines contained in the **libF77rt** library are presented in alphabetical order. Figure 8-2 illustrates the approximate order in which you might invoke the subroutines from an application program.

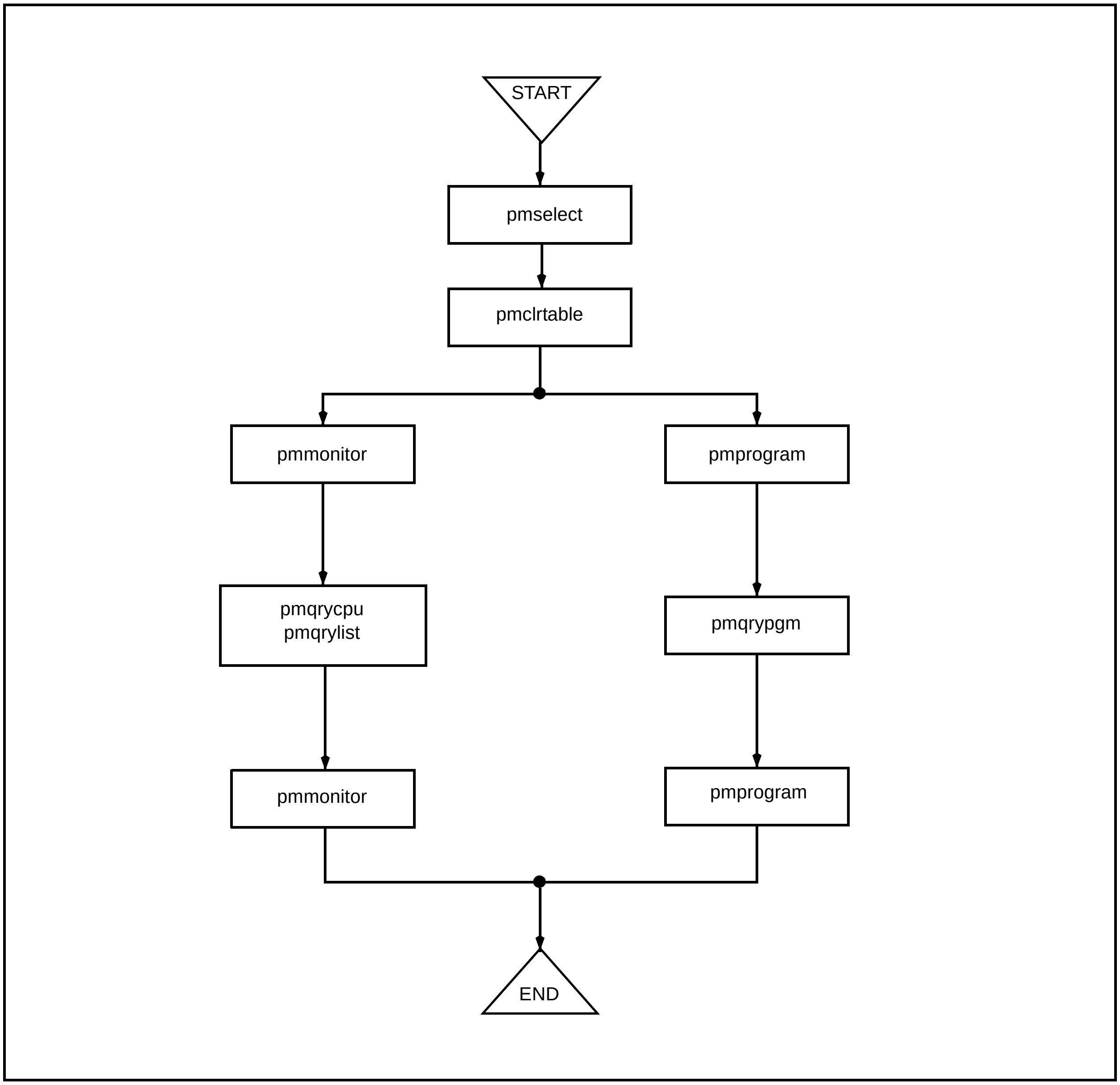

Figure 8-2. FORTRAN Library Call Sequence: Performance Monitor

Pmclrpgm – Clear Values for a Process

This subroutine is invoked to clear performance monitor values for a particular process that has been scheduled on a frequency–based scheduler. You can identify the process by using one of the following methods:

- Specify the name of the process and the CPU(s) on which it is scheduled.
- Specify the process's frequency–based scheduler process identifier (slot number).
- Specify the name of the process, the CPU on which it is scheduled, and its frequency–based scheduler process identifier.

NOTE

This subroutine will clear the process' total soft overrun count.

The FORTRAN variable declarations, call statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER schdle
CHARACTER* (*) name
INTEGER cpu
INTEGER slot
INTEGER istat

CALL Statement

CALL pmclrpgm(schdle, name, cpu, slot, istat)

Parameters

Parameters are described as follows.

schdle — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process is scheduled. You can obtain this value by making a call to **`fbsconfigure`** (see page 8-6 for an explanation of this subroutine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **`1`**.

name — refers to a variable that contains a standard UNIX path name identifying the process for which values are to be cleared. A full or relative path name of up to 1024 characters can be specified. If this variable is filled with blanks, you must provide the frequency–based scheduler process identifier in the *slot* parameter.

cpu — refers to a variable that contains an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the process for which values are to be cleared. Acceptable values and corresponding results are presented in Table 8-21

Table 8-21. CPU Options: pmclrpgm

Value	Result
0	The first process named by *name* that is currently running on the processor from which the call is made is specified
-1	The first process named by *name* that is currently running on any processor is specified

Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is currently running on CPU *i* is specified
	
	If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is specified

slot — refers to a variable that contains an integer value providing the unique frequency–based scheduler process identifier for the process for which values are to be cleared. This value is obtained when you make a call to **`schedpgmadd`** (see page 8-57 for an explanation of this subroutine). This value must be – **`1`** if you wish to identify the process only by specifying *name* and *cpu.*

istat — refers to a variable to which **`pmclrpgm`** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page **`pmclrpgm(3F77rt)`** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Pmclrtable – Clear Values for Processor(s)

This subroutine is invoked to clear performance monitor values for FBS–scheduled processes on one or more specified processors on a selected scheduler.

NOTE

This subroutine will clear the total soft overrun count for all related processes.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER schdle
INTEGER cpucount
INTEGER cpulist(cpucount)
INTEGER istat

CALL Statement

CALL pmclrtable(schdle, cpucount, cpulist, istat)

Parameters

Parameters are described as follows.

schdle — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the processes are scheduled. You can obtain this value by making a call to **`fbsconfigure`** (see page 8-6 for an explanation of this subroutine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **`1`**.

cpucount — refers to a variable that contains an integer value indicating the number of elements contained in the array represented by *cpulist.*

cpulist — refers to an array that consists of the number of elements specified by the *cpucount* parameter and contains one or more integer values indicating the processor or processors for which performance monitor values are to be cleared.

Acceptable values and corresponding results are presented in Table 8-22.

Table 8-22. CPU Options: pmclrtable

Value	Result
0	Performance monitor values for FBS–scheduled processes executing on the processor from which the call is made are cleared
-1	Performance monitor values for all processes on the scheduler
Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU), performance monitor values for processes executing on CPU *i* are cleared

istat refers to a variable to which **pmclrtable** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page **pmclrtable(3F77rt)** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Pmmonitor – Start/Stop Performance Monitoring on Processor(s)

This subroutine is invoked to start or stop performance monitoring for FBS–scheduled processes on one or more specified processors on a selected scheduler.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER schdle
INTEGER pmflag
INTEGER cpucount
INTEGER cpulist(cpucount)
INTEGER istat

CALL Statement

CALL pmmonitor(schdle, pmflag, cpucount, cpulist, istat)

Parameters

Parameters are described as follows.

schdle — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the processes are scheduled. You can obtain this value by making a call to **fbsconfigure** (see page 8-6 for an explanation of this subroutine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

pmflag — refers to a variable that contains an integer value indicating whether performance monitoring is to be started or stopped. A nonzero value indicates that performance monitoring is to be started. A zero value indicates that performance monitoring is to be stopped.

cpucount — refers to a variable that contains an integer value indicating the number of elements in the array represented by *cpulist*.

cpulist — refers to an array that consists of the number of elements specified by the *cpucount* parameter and contains one or more integer values indicating the processor or processors for which performance monitoring is to be started or stopped. Acceptable values and corresponding results are presented in Table 8-23.

Table 8-23. CPU Options: pmmonitor

Value	Result
0	Performance monitoring for FBS–scheduled processes executing on the processor from which the call is made is started or stopped
-1	Performance monitoring for all processes on the scheduler is started or stopped
Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU), performance monitoring for processes executing on CPU *i* is started or stopped

istat — refers to a variable to which **pmmonitor** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page **pmmonitor(3F77rt)** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Pmprogram – Start/Stop Performance Monitoring on a Process

This subroutine is invoked to start or stop performance monitoring for a particular process that has been scheduled on a frequency–based scheduler. You can identify the process by using one of the following methods:

- Specify the name of the process and the CPU(s) on which it is scheduled.
- Specify the process's frequency–based scheduler process identifier (slot number).
- Specify the name of the process, the CPU(s) on which it is scheduled, and its frequency–based scheduler process identifier.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER schdle
CHARACTER* (*) name
INTEGER cpu
INTEGER slot
INTEGER pmflag
INTEGER istat

CALL Statement

CALL pmprogram(schdle, name, cpu, slot, pmflag, istat)

Parameters

Parameters are described as follows.

schdle — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the process is scheduled. You can obtain this value by making a call to **`fbsconfigure`** (see page 8-6 for an explanation of this subroutine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **`1`**.

name — refers to a variable that contains a standard UNIX path name identifying the process for which performance monitoring is to be started or stopped. A full or relative path name of up to 1024 characters can be specified. If this variable is filled with blanks, you must provide the frequency–based scheduler process identifier in the *slot* parameter.

cpu — refers to a variable that contains an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the process for which performance monitoring is to be started or stopped. Acceptable values and corresponding results are presented in Table 8-24.

Table 8-24. CPU Options: pmprogram

Value	Result
0	The first process named by *name* that is currently running on the processor from which the call is made is specified
-1	The first process named by *name* that is currently running on any processor is specified

Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is currently running on CPU *i* is specified
	
	If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is specified

slot — refers to a variable that contains an integer value providing the unique frequency–based scheduler process identifier for the process for which performance monitoring is to be started or stopped. This value is obtained when you make a call to **schedpgmadd** (see page 8-57 for an explanation of this subroutine). This value must be – **1** if you wish to identify the process only by specifying *name* and *cpu*.

pmflag — refers to a variable that contains an integer value indicating whether performance monitoring is to be started or stopped. A nonzero value indicates that performance monitoring is to be started. A zero value indicates that performance monitoring is to be stopped.

istat — refers to a variable to which **pmprogram** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page **pmprogram(3F77rt)** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Pmqrycpu – Query Values for Selected Processor(s)

This subroutine is invoked to obtain performance monitor values for FBS–scheduled processes on one or more specified processors on a selected scheduler.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER schdle
INTEGER cpu
INTEGER bufsiz
INTEGER buf(bufsize)
INTEGER istat

CALL Statement

CALL pmqrycpu(schdle, cpu, bufsiz, buf, istat)

Parameters

Parameters are described as follows.

schdle — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler on which the processes are scheduled. You can obtain this value by making a call to **fbsconfigure** (see page 8-6 for an explanation of this subroutine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

cpu — refers to a variable that contains an integer value indicating the processor(s) for which performance monitor values are to be obtained. Acceptable values and corresponding results are presented in Table 8-25.

Table 8-25. CPU Options: pmqrycpu

Value	Result
0	Performance monitor values for FBS–scheduled processes executing on the processor from which the call is made are returned
-1	Performance monitor values for all processes on the scheduler are returned
Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU), performance monitor values for processes executing on CPU *i* are returned

bufsiz — refers to a variable that contains an integer value indicating the size in 32–bit words of the array represented by *buf*. Because 16 words of information are returned for each process, it is recommended that this value be a multiple of 16.

buf — refers to an array to which **pmqrycpu** will return a series of 16 integer values for each FBS–scheduled process on the processor(s) specified with the *cpu* parameter. The number of processes for which these values are returned is bound by the value of the *bufsiz* parameter. If, for example, the value of *bufsiz* is 165, values for 10 processes will be returned. These values represent the performance monitoring information for the process(es). The type of information returned in each array element for a single process is presented in Table 8-26.

Table 8-26. Contents of Array Elements: pmqrycpu

Element	Contents
1	The process's frequency–based scheduler process identifier (slot number)
2	The amount of time that the process has spent running from the last time that it has been wakened by the scheduler until it has called **fbswait** (last time)
3	The number of times that the process has been wakened by the scheduler (total iterations, or cycles)
4	The number of seconds that the process has spent running in all cycles (total seconds). The total amount of time that the process has spent running is equal to the value of Element 4 plus the value of Element 5.
5	The additional number of microseconds that the process has spent running in all cycles (total microseconds). The total amount of time that the process has spent running is equal to the value of Element 4 plus the value of Element 5.
6	The number of hard frame overruns caused by the process
7	The least amount of time that the process has spent running in a cycle (minimum cycle time)
8	The number of the minor cycle in which the minimum cycle time has occurred (minimum cycle cycle)
9	The number of the major frame in which the minimum cycle time has occurred (minimum cycle frame)
10	The greatest amount of time that the process has spent running in a cycle (maximum cycle time)

Table 8-26. Contents of Array Elements: pmqrycpu (Cont.)

Element	Contents
11	The number of the minor cycle in which the maximum cycle time has occurred (maximum cycle cycle)
12	The number of the major frame in which the maximum cycle time has occurred (maximum cycle frame)
13	The least amount of time that the process has spent running during a major frame (minimum frame time)
14	The number of the major frame in which the minimum frame time has occurred (minimum frame frame)
15	The greatest amount of time that the process has spent running during a major frame (maximum frame time)
16	The number of the major frame in which the maximum frame time has occurred (maximum frame frame)

istat — refers to a variable to which **pmqrycpu** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page **pmqrycpu(3F77rt)** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Pmqrylist – Query Values for a List of Processes

This subroutine is invoked to obtain performance monitor values for a list of processes scheduled on a frequency–based scheduler.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER schdle
INTEGER slotcount
INTEGER slotlist(slotcount)
INTEGER bufsiz
INTEGER buf(bufsize)
INTEGER istat

CALL Statement

CALL pmqrylist(schdle, slotcount, slotlist, bufsiz, buf, istat)

Parameters

Parameters are described as follows.

schdle — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler for which performance monitor values are requested. You can obtain this value by making a call to **`fbsconfigure`** (see page 8-6 for an explanation of this subroutine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **`1`**.

slotcount — refers to a variable that contains an integer value indicating the number of frequency–based scheduler process identifiers contained in the array represented by *slotlist*.

slotlist — refers to an array that consists of the number of elements specified by the *slotcount* parameter and contains one or more integer values indicating the frequency–based scheduler process identifiers for which performance monitor values are to be returned.

bufsiz — refers to a variable that contains an integer value indicating the size in 32–bit words of the array represented by *buf*. Because 15 words of information are returned for each process, it is recommended that this value be a multiple of 15.

buf — refers to an array to which **`pmqrylist`** will return a series of 15 integer values for each FBS–scheduled process. The number of processes for which these values are returned is bound by the value of the *bufsiz* parameter. If, for example, the value of *bufsiz* is 155, values for 10 processes will be returned. These values represent the performance monitoring information for the processes. The type of information returned in each array element for a single process is presented in Table 8-27.

Table 8-27. Contents of Array Elements: pmqrylist

Element	Contents
1	The amount of time that the process has spent running from the last time that it has been wakened by the scheduler until it has called **`fbswait`** (last time)
2	The number of times that the process has been wakened by the scheduler (total iterations, or cycles)
3	The number of seconds that the process has spent running in all cycles (total seconds). The total amount of time that the process has spent running is equal to the value of Element 3 plus the value of Element 4.

Table 8-27. Contents of Array Elements: pmqrylist

Element	Contents
4	The additional number of microseconds that the process has spent running in all cycles (total microseconds). The total amount of time that the process has spent running is equal to the value of Element 3 plus the value of Element 4.
5	The number of hard frame overruns caused by the process
6	The least amount of time that the process has spent running in a cycle (minimum cycle time)
7	The number of the minor cycle in which the minimum cycle time has occurred (minimum cycle cycle)
8	The number of the major frame in which the minimum cycle time has occurred (minimum cycle frame)
9	The greatest amount of time that the process has spent running in a cycle (maximum cycle time)
10	The number of the minor cycle in which the maximum cycle time has occurred (maximum cycle cycle)
11	The number of the major frame in which the maximum cycle time has occurred (maximum cycle frame)
12	The least amount of time that the process has spent running during a major frame (minimum frame time)
13	The number of the major frame in which the minimum frame time has occurred (minimum frame frame)
14	The greatest amount of time that the process has spent running during a major frame (maximum frame time)
15	The number of the major frame in which the maximum frame time has occurred (maximum frame frame)

istat
: refers to a variable to which **pmqrylist** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page **pmqrylist(3F77rt)** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Pmqrypgm – Query Values for a Selected Process

This subroutine is invoked to obtain performance monitor values for a particular process scheduled on a frequency–based scheduler.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER schdle
CHARACTER* (*) name
INTEGER cpu
INTEGER slot
INTEGER last
INTEGER iter
INTEGER totsec
INTEGER totusec
INTEGER over
INTEGER minc
INTEGER minctc
INTEGER minctf
INTEGER maxc
INTEGER maxctc
INTEGER maxctf
INTEGER minf
INTEGER minftf
INTEGER maxf
INTEGER maxftf
INTEGER istat

CALL Statement

CALL pmqrypgm(schdle, name, cpu, slot, last, iter, totsec, totusec, over, minc,
minctc, minctf, maxc, maxctc, maxctf, minf, minftf, maxf, maxftf, istat)

Parameters

Parameters are described as follows.

schdle — refers to a variable that contains a unique, positive integer value representing the identifier for the frequency–based scheduler for which performance monitor values are requested. You can obtain this value by making a call to **fbsconfigure** (see page 8-6 for an explanation of this subroutine). If you wish to reference the frequency–based scheduler on which the calling process is scheduled without knowing its identifier, you can specify a value of – **1**.

name — refers to a variable that contains a standard UNIX path name identifying the process for which performance monitoring values are to be returned. A full or relative path name of up to 1024 characters can be specified. If this variable is filled with blanks, you must provide the frequency–based scheduler process identifier in the *slot* parameter.

cpu
: refers to a variable that contains an integer value indicating the processor(s) to be used in conjunction with the value of the *name* parameter to identify the process for which performance monitoring values are to be returned. Acceptable values and corresponding results are presented in Table 8-28.

Table 8-28. CPU Options: pmqrypgm

Value	Result
0	The first process named by *name* that is currently running on the processor from which the call is made is specified
-1	The first process named by *name* that is currently running on any processor is specified

Bit mask	If (*cpu* & (1<<*i*)) is set (where *i* is an integer ranging from zero to 15 and representing a CPU) and it is the only bit set, the first process named by *name* that is currently running on CPU *i* is specified
	
	If (*cpu* & (1<<*i*)) is set and it is not the only bit set, the first process named by *name* that is currently running on any of the selected CPUs is specified

slot
: refers to a variable that contains an integer value providing the unique frequency–based scheduler process identifier for the process for which performance monitoring values are to be returned. This value is obtained when you make a call to **`schedpgmadd`** (see page 8-57 for an explanation of this subroutine). This value must be − **`1`** if you wish to identify the process only by specifying *name* and *cpu*.

last
: refers to a variable to which **`pmqrypgm`** will return an integer value indicating the amount of time that the process has spent running from the last time that it has been wakened by the scheduler until it has called **`fbswait`** (last time).

iter
: refers to a variable to which **`pmqrypgm`** will return an integer value indicating the number of times that the process has been wakened by the frequency–based scheduler since the last time that performance monitor values have been cleared and performance monitoring has been enabled (total iterations, or cycles).

totsec
: refers to a variable to which **`pmqrypgm`** will return an integer value indicating the number of seconds that the process has spent running in all cycles (total time in seconds). The total amount of time that the process has spent running is equal to the value of *totsec* plus *totusec*.

totusec — refers to a variable to which **pmqrypgm** will return an integer value indicating the additional number of microseconds that the process has spent running in all cycles (total time in microseconds). The total amount of time that the process has spent running is equal to the value of *totsec* plus *totusec*.

over — refers to a variable to which **pmqrypgm** will return an integer value indicating the number of times that the process has caused a hard frame overrun.

minc — refers to a variable to which **pmqrypgm** will return an integer value indicating the least amount of time that the process has spent running in a cycle (minimum cycle time).

minctc — refers to a variable to which **pmqrypgm** will return an integer value indicating the number of the minor cycle in which the minimum cycle time has occurred (minimum cycle cycle).

minctf — refers to a variable to which **pmqrypgm** will return an integer value indicating the number of the major frame in which the minimum cycle time has occurred (minimum cycle frame).

maxc — refers to a variable to which **pmqrypgm** will return an integer value indicating the greatest amount of time that the process has spent running in a cycle (maximum cycle time).

maxctc — refers to a variable to which **pmqrypgm** will return an integer value indicating the number of the minor cycle in which the maximum cycle time has occurred (maximum cycle cycle).

maxctf — refers to a variable to which **pmqrypgm** will return an integer value indicating the number of the major frame in which the maximum cycle time has occurred (maximum cycle frame).

minf — refers to a variable to which **pmqrypgm** will return an integer value indicating least amount of time that the process has spent running in a major frame (minimum frame time).

minftf — refers to a variable to which **pmqrypgm** will return an integer value indicating the number of the major frame in which the minimum frame time has occurred (minimum frame frame).

maxf — refers to a variable to which **pmqrypgm** will return an integer value indicating the greatest amount of time that the process has spent running in a major frame (maximum frame time).

maxftf — refers to a variable to which **pmqrypgm** will return an integer value indicating the number of the major frame in which the maximum frame time has occurred (maximum frame frame).

istat — refers to a variable to which **pmqrypgm** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page **pmqrypgm(3F77rt)** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Pmquerytimer – Query Performance Monitor Mode

This subroutine is invoked to determine whether performance monitor timing values include or exclude time spent servicing interrupts.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER mode

CALL Statement

CALL pmquerytimer(mode)

Parameter

Pmquerytimer requires one parameter: *mode*. *Mode* refers to a variable to which **pmquerytimer** will return a value indicating whether performance monitor timing values include or exclude time spent servicing interrupts. A value of one indicates that interrupt time is included. A value of zero indicates that interrupt time is excluded. A value of one or zero is returned if the call is successful; a negative value is returned to indicate that an error of a specific type has occurred. Refer to the system manual page **pmquerytimer(3F77rt)** for a listing of the values that may be returned and the types of errors that they represent.

Pmselect – Select Performance Monitor Mode

This subroutine is invoked to select the timing mode under which the performance monitor is to run. The timing mode can be set to include or exclude time spent servicing interrupts. Note that to set the timing mode, the calling process must have the P_RTIME privilege (for additional information on privileges, refer to the *PowerMAX OS Programming Guide* and the **`intro(2)`** system manual page).

CAUTION

The timing mode for the high–resolution timing facility is set system–wide. It affects all processes running on all CPUs.

The FORTRAN variable declarations, CALL statement, and corresponding parameters are presented in the following sections.

Variable Declarations

INTEGER mode
INTEGER istat

CALL Statement

CALL pmselect(mode, istat)

Parameters

Parameters are described as follows.

mode — refers to a variable that contains an integer value indicating whether time spent servicing interrupts is to be included in or excluded from performance monitor timing values. A nonzero value indicates that interrupt time is to be included. A value of zero indicates that interrupt time is to be excluded.

istat — refers to a variable to which **`pmselect`** will return an integer value indicating whether or not an error has occurred. A value of zero indicates that no error has occurred. A nonzero value indicates that an error of a specific type has occurred. Refer to the system manual page **`pmselect(3F77rt)`** for a listing of the nonzero values that may be returned and the types of errors that they represent.

Compiling and Linking Procedures

To link a FORTRAN program, the following library is required:

`/usr/lib/libF77rt.a`

To compile and link a FORTRAN program, the command line instruction is as follows:

`hf77` *program_name* **`-lF77rt`**

For additional information on compiling and linking procedures, refer to the system manual pages **`ld(1)`** and **`hf77(1)`**.

A
Example Rtcp Script

This appendix contains an example of a script that can be invoked at the system command prompt to execute **rtcp** commands. This script illustrates use of the commands to configure a scheduler and schedule programs on it; view information about the FBS-scheduled processes; view the scheduler configuration; attach, set, and start a real–time clock; clear performance monitor values; start performance monitoring and frequency–based scheduling; view the minor cycle and major frame count; stop performance monitoring; stop the real-time clock and the frequency-based scheduler; detach the real-time clock; and remove the scheduler and all scheduled processes.

```
rtcp cs –s37 –C10 –I664 –M5 –N10
rtcp sp –s37 -n/usr1/rtc/tests/testprogram1 –c0 -bF –p19 –f4 –m0 -ohalt
rtcp sp –s37 -n/usr1/rtc/tests/testprogram2 –c1 -bF –p19 –f2 –m1 -ohalt
rtcp vp –s37 –c*
rtcp vs –s37
rtcp ats –s37 –d/dev/rrtc/0c2
rtcp stc –s37 –O10000 –D1
rtcp rc –s37
rtcp cpm –s37 –c*
rtcp pm –s37 –c* –PON
rtcp start –s37
rtcp vc –s37
rtcp pm -s37 -c* -POFF
rtcp sc -s37
rtcp stop -s37
rtcp dts -s37
rtcp rms -s37 -a
```

B
Rtcp Error Messages

This appendix contains descriptions of the errors that may be reported by the real–time command processor, **rtcp**. System errors are listed and described in Table B-1. **Rtcp** errors are listed and described in Table B-2.

Table B-1. System Errors

Error	Description
EPERM	User not creator of FBS or does not have P_RTIME privilege
ENOENT	Scheduler not configured
ESRCH	Process not scheduled on FBS
EINTR	EOC triggering is invalid at this time
EIO	Real–time clock not configured
ENOEXEC	Process not found
EAGAIN	Unable to create (fork) the new process
ENOMEM	Unable to allocate buffers
EACCES	Permission denied
EFAULT	Invalid parameter list
EBUSY	Specified device already in use
EEXIST	Scheduler already exists
ENODEV	Specified device does not exist
EINVAL	FBS not configured in kernel
ENOTTY	Device does not support FBS ioctl call
EFBIG	Buffer too small
ENOSPC	No room in FBS table
ERANGE	Parameter out of range
EISCONN	Scheduler attached to interrupt
ENOTCONN	Scheduler is not attached to interrupt
ENAMETOOLONG	Invalid pathname specified
ENOTEMPTY	Processes still scheduled
ENOSYS	FBS not configured in kernel
ESHUTDOWN	Coupled FBS disabled
EREMOTE	Remote message communication failed

Table B-1. System Errors (Cont.)

Error	Description
EBADMSG	Invalid reply to message was received
EIDRM	Scheduler is a virtual FBS
ECOMM	Message communication failed
EHOSTUNREACH	Local hostname must be specified
EADDRNOTAVAIL	A hostname could not be found
ECONFIG	Device configuration error
ENOPKG	Kernel configuration error
ECONNREFUSED	A host refused message connection request
ETIMEDOUT	Message communication timed out

Table B-2. rtcp Errors

Error	Description
-2	Interrupt device not specified
-3	Both EOC and interrupt device specified
-4	Process not specified
-5	Invalid rtcp command specified
-6	Invalid help command specified
-7	Invalid cpu (–c) parameter
-8	Invalid frequency (–f) parameter
-9	Invalid halt flag (–h) parameter
-10	Invalid start cycle (–m) parameter
-11	Invalid priority (–p) parameter
-12	Invalid cycle count (–C) parameter
-13	Invalid clock tick duration (–D) parameter
-14	Invalid process per cycle (–M) parameter
-15	Invalid process per fbs (–N) parameter
-16	Invalid clock ticks per cycle (–O) parameter
-17	Invalid PM flag (–P) parameter
-18	Invalid parameter specified

Table B-2. rtcp Errors (Cont.)

Error	Description
-22	rj file not specified
-23	Invalid rj file specified
-24	Invalid pm viewing mode specified
-25	Invalid pm timing mode specified
-26	Unable to change timing mode to exclude interrupt time
-27	Invalid scheduling policy specified
-28	Exit rtcp
-29	Invalid soft overrun limit (-L) parameter
-30	Invalid timing device type (-T) parameter
-31	Number of hosts (-H) exceeds limit
-32	Hostname(s) (-H) not specified

C
Example: C Interface to the FBS and PM

This appendix contains an example program that illustrates use of the C library interface to the frequency–based scheduler and the performance monitor. It shows how to configure a scheduler; schedule programs on it; attach, set, and start a real–time clock; start performance monitoring for each FBS–scheduled process; start frequency–based scheduling; and obtain performance monitor values for the FBS–scheduled processes. It also shows how to monitor a processor's idle time.

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sched.h>
#include <fbslib.h>
#include <errno.h>



#define NUM_PROCS  4
#define START      1
#define STOP       0



main()
{

     struct    fbsconfig_ds   fbs_buf;
     struct    pgm2_ds        sched_buf;
     struct    pmqry_ds       pm_buf[NUM_PROCS];
     struct    fbsinfo_ds     info_buf;
     struct    fbscycle_ds    cycle_buf;
     int       idle0_fpid;         /* fpid for idle on cpu 0 */
     int       idle1_fpid;         /* fpid for idle on cpu 1 */
     int       pgm1_fpid;          /* fpid for testprogram 1 */
     int       pgm2_fpid;          /* fpid for testprogram 2 */
     int       cpu;
     int       istat;
     int       pmflg;
     int       intrflg;
     int       i;
     int       count;
     int       resolution;
     char      name[1024];         /* program's full or relative path name */
     char      devname[40];        /* device name of timing source */
     FILE      *fp;


     /* Open file to store performance information */
     fp= fopen("pmresults", "w+");

     if (fp == NULL)
          printf("open failed errno = %d\n", errno);
```

```
/*
 * CONFIGURE SCHEDULER
 */
fbs_buf.key = 37;              /* scheduler key */
fbs_buf.cycles = 10;           /* number of cycles per frame */
fbs_buf.progs = 5;             /* max. number of programs per cycle */
fbs_buf.max = 10;              /* max. number of programs allowed on the FBS */
fbs_buf.reset = -1;            /* kill & remove processes currently scheduled */

/* owner/group read/write */
fbs_buf.configflg = IPC_CREAT | IPC_EXCL | 0664;

istat = fbsconfigure(&fbs_buf);

if (istat != 0) {
     printf("could not configure scheduler: errno = %d\n" ,errno);
     return;

}

/*
 * SCHEDULE test program1 "/usr1/rtc/tests/testprogram1"
 */
sched_buf.name_ptr = "/usr1/rtc/tests/testprogram1";
sched_buf.cid = SCHED_FIFO;    /* first-in-first out (FIFO) policy */
sched_buf.prior = 19;
sched_buf.param = 0;           /* optional initiation parameter */
sched_buf.period = 4;          /* time between wakeups */
sched_buf.cycle = 0;           /* starting base cycle */
sched_buf.halt = 0;            /* halt on overrun */

/* Set cpu mask to schedule testprogram1 on cpu 0 */
sched_buf.cpu = 1;

istat =  sched_pgmadd(fbs_buf.fbs_id, &sched_buf);
printf("fbsid = %d\n", fbs_buf.fbs_id);

if (istat != 0) {
     printf("could not schedule %s on cpu %d : errno  = %d\n",
     sched_buf.name_ptr, sched_buf.cpu, errno);
     return;
}

pgm1_fpid = sched_buf.fpid;
printf("pgm1 fpid = %d\n", pgm1_fpid);

/*
 * SCHEDULE test program2 "/usr1/rtc/tests/testprogram2"
 */
sched_buf.name_ptr = "/usr1/rtc/tests/testprogram2";
sched_buf.prior = 19;
sched_buf.cid = SCHED_FIFO;    /* first-in-first out (FIFO) policy */
sched_buf.param = 0;           /* optional initiation parameter */
sched_buf.period = 2;          /* time between wakeups */
sched_buf.cycle = 1;           /* starting base cycle */
sched_buf.halt = 0;            /* halt on overrun */

/* Set cpu mask to schedule testprogram2 on cpu 0 */
sched_buf.cpu = 1;

istat =  sched_pgmadd(fbs_buf.fbs_id, &sched_buf);
printf("fbsid = %d\n", fbs_buf.fbs_id);
```

```
if (istat != 0) {
     printf("could not schedule %s on cpu %d : errno  = %d\n",
     sched_buf.name_ptr, sched_buf.cpu, errno);
     return;
}

pgm2_fpid = sched_buf.fpid;
printf("pgm2 fpid = %d\n", pgm2_fpid);


/*
 * SCHEDULE IDLE ON CPU 0
 *
 * The only parameter required for /idle is the CPU.
 */

sched_buf.name_ptr = "/idle";
sched_buf.cid = SCHED_FIFO;    /* first-in-first out (FIFO) policy */
sched_buf.prior = 20;
sched_buf.param = 0;           /* optional initiation parameter */
sched_buf.period = 1;          /* time between wakeups */
sched_buf.cycle = 0;           /* starting base cycle */
sched_buf.halt = 0;            /* halt on overrun */

/* Set cpu mask to schedule idle on cpu 0 */
sched_buf.cpu = 1;

istat =  sched_pgmadd(fbs_buf.fbs_id, &sched_buf);
printf("fbsid = %d\n", fbs_buf.fbs_id);

if (istat != 0) {
     printf("could not schedule %s on cpu %d : errno  = %d\n",
     sched_buf.name_ptr, sched_buf.cpu, errno);
     return;
}

idle0_fpid = sched_buf.fpid;
printf("idle0 fpid = %d\n", idle0_fpid);

/*
 * SCHEDULE IDLE ON CPU 1
 *
 * The only parameter required for /idle is the CPU.
 */
sched_buf.name_ptr = "/idle";
sched_buf.prior = 20;
sched_buf.cid = SCHED_FIFO;    /* first-in-first out (FIFO) policy */
sched_buf.param = 0;           /* optional initiation parameter */
sched_buf.period = 1;          /* time between wakeups */
sched_buf.cycle = 0;           /* starting base cycle */
sched_buf.halt = 0;            /* halt on overrun */

/* Set cpu mask to schedule idle on cpu 1 */
sched_buf.cpu = 2;

istat =  sched_pgmadd(fbs_buf.fbs_id, &sched_buf);

if (istat != 0) {
     printf("could not schedule %s on cpu %d : errno  = %d\n",
     sched_buf.name_ptr, sched_buf.cpu, errno);
     return;
}

idle1_fpid = sched_buf.fpid;
printf("idle1 fpid = %d\n", idle1_fpid);
```

```
/*
 * ATTACH/SET REAL-TIME CLOCK
 * Set the clock to interrupt every 10,000 usecs.
 */
count = 10000;
resolution = 1;
istat = fbsattach(fbs_buf.fbs_id, "/dev/rrtc/0c2");

if (istat != 0) {
     printf("could not attach timing source: errno = %d\n", errno);
     return;
}

istat = fbssetrtc(fbs_buf.fbs_id, count, resolution);

if (istat != 0) {
     printf("could not set rtc: errno = %d\n", errno);
     return;
}

istat = fbsrunrtc(fbs_buf.fbs_id, START);

if (istat != 0) {
     printf("could not start rtc: errno = %d\n", errno);
     return;
}


/*
 * START PERFORMANCE MONITORING
 */
pmflg = 1;

/* zero out the "name" variable (fpid must be specified).
 * Ulimately, the "name" variable can be used to store
 * the full or relative path name of a test program.
 */
bzero(name, sizeof(name));
cpu = 0;                 /* not used if fpid is being used */


/* start performance monitoring for testprogram1 */
istat = pmprogram(fbs_buf.fbs_id, name, cpu, pgm1_fpid, pmflg);

if (istat != 0) {
     printf("could not start pm for testprogram1 : errno = %d\n", errno);
     return;
}


/* start performance monitoring for testprogram2 */
istat = pmprogram(fbs_buf.fbs_id, name, cpu, pgm2_fpid, pmflg);



if (istat != 0) {
     printf("could not start pm for testprogram2 : errno = %d\n", errno);
     return;
}

/* start performance monitoring for idle on cpu 0 */
istat = pmprogram(fbs_buf.fbs_id, name, cpu, idle0_fpid, pmflg);
```

```
if (istat != 0) {
     printf("could not start pm for idle0 : errno = %d\n", errno);
     return;
}

/* start performance monitoring for idle on cpu 1 */
istat = pmprogram(fbs_buf.fbs_id, name, cpu, idle1_fpid, pmflg);

if (istat != 0) {
     printf("could not start pm for idle1 : errno = %d\n", errno);
     return;
}

/*
 * START SCHEDULING
 */
intrflg = 1;
istat = fbsintrpt(fbs_buf.fbs_id, intrflg);

if (istat != 0) {
     printf("could not start scheduler : errno = %d\n", errno);
     return;
}


/*
 * QUERY PERFORMANCE MONITOR VALUES
 * 1 second = 100 cycles  1 minute = 600 frames
 * Query once per second for 1 minute
 */
pm_buf[0].fpid = pgm1_fpid;
pm_buf[1].fpid = pgm2_fpid;
pm_buf[2].fpid = idle0_fpid;
pm_buf[3].fpid = idle1_fpid;

sleep(1);                /* sleep for a while */

printf("Please wait, performance information is being gathered. \n");

istat = fbscycle(fbs_buf.fbs_id, &cycle_buf);
while ((istat == 0) && (cycle_buf.cframe < 600)) {
istat = pmqrylist(fbs_buf.fbs_id, pm_buf, NUM_PROCS);
if (istat != 0) {
     printf("could not query process: errno = %d\n", errno);
     return;
}

/* Write performance information for each
 * process into a file
 */
for (i = 0; i < NUM_PROCS; i++) {
     fprintf(fp, "fpid %d\n", pm_buf[i].fpid);
     fprintf(fp, "last cycle tm = %d\n",pm_buf[i].lastcyc_tm);
     fprintf(fp, "total cyles = %d\n",pm_buf[i].tot_cycles);
     fprintf(fp, "total secs = %d\n",pm_buf[i].tot_sec);
     fprintf(fp, "total usec = %d\n",pm_buf[i].tot_usec);
     fprintf(fp, "overruns = %d\n",pm_buf[i].overruns);
     fprintf(fp, "mincyc_tm = %d\n",pm_buf[i].mincyc_tm);
     fprintf(fp, "mincyc_cycle = %d\n",pm_buf[i].mincyc_cycle);
     fprintf(fp, "mincyc_frame = %d\n",pm_buf[i].mincyc_frame);
     fprintf(fp, "maxcyc_tm = %d\n", pm_buf[i].maxcyc_tm);
     fprintf(fp, "maxcyc_cycle =%d\n",pm_buf[i].maxcyc_cycle);
     fprintf(fp, "maxcyc_frame =%d\n",pm_buf[i].maxcyc_frame);
     fprintf(fp, "minframe_tm = %d\n", pm_buf[i].minframe_tm);
     fprintf(fp, "minframe = %d\n", pm_buf[i].minframe);
```

```
                fprintf(fp, "maxframe_tm = %d\n", pm_buf[i].maxframe_tm);
                fprintf(fp, "maxframe = %d\n", pm_buf[i].maxframe);
                fprintf(fp, "\n");
           }

     fprintf(fp, "\n***************************\n");

     sleep(1);            /* sleep for a while */
     istat = fbscycle(fbs_buf.fbs_id, &cycle_buf);

     }

     printf("Performance data has been gathered. \n");

     if (istat != 0)
         printf("istat = %d\n", istat);

     /*Stop PM on CPUs 0 and 1 */
     pmmonitor(fbs_buf.fbs_id, 0, 3);

     /* Stop the clock */
     istat = fbsrunrtc(fbs_buf.fbs_id, STOP);

     if (istat != 0) {
          printf("could not stop rtc: errno = %d\n", errno);
          return;
     }

     /* Detach from the FBS */
     istat = fbsdetach(fbs_buf.fbs_id);

     if (istat != 0) {
          printf("could not dettach timing source: errno = %d\n", errno);
          return;
     }

     /* Remove the FBS */
     istat = fbsremove(fbs_buf.fbs_id, -1);

     if (istat != 0) {
          printf("could not remove timing source: errno = %d\n", errno);
          return;
     }

}
```

Glossary

data monitoring

Services that make it possible to monitor variables in executing processes.

end–of–cycle scheduling

A form of frequency–based scheduling in which scheduling is triggered when the last process that is scheduled to execute in the current minor cycle of the current major frame completes its processing.

FBSMNI

A system tunable parameter that is associated with the frequency–based scheduler. It specifies the maximum number of frequency–based schedulers that can be configured at one time system–wide.

FBSUNSCHEDMAX

A system tunable parameter that is associated with the frequency–based scheduler. It specifies the maximum number of unscheduled processes that is permitted on a frequency-based scheduler.

frame overrun

The condition that occurs when an FBS–scheduled process does not finish its processing before it is scheduled to run again.

frequency–based scheduler

A high resolution task synchronization mechanism that enables processes to run at user–specified frequencies.

high resolution timing facility

A feature of PowerMAX OS systems that provides a means of measuring each process's or LWP's execution time.

idle time

Time during which the CPU is not busy.

iteration

One instance of a process's being wakened by a frequency–based scheduler.

last time

A value returned by the performance monitor indicating the amount of time that an FBS–scheduled process has spent running from the last time that it has been wakened by the scheduler until it has called **fbswait**.

major frame

One pass through all of the minor cycles with which a frequency–based scheduler is configured. A major frame has associated with it a duration, which is obtained by multiplying the duration of a minor cycle by the number of minor cycles per major frame.

maximum cycle cycle

A value returned by the performance monitor indicating the number of the minor cycle in which the maximum cycle time has occurred.

maximum cycle frame

A value returned by the performance monitor indicating the number of the major frame in which the maximum cycle time has occurred.

maximum cycle time

A value returned by the performance monitor indicating the greatest amount of time that an FBS–scheduled process has spent running in a cycle.

maximum frame frame

A value returned by the performance monitor indicating the number of the major frame in which the maximum frame time has occurred.

maximum frame time

A value returned by the performance monitor indicating the greatest amount of time that an FBS–scheduled process has spent running during a major frame.

minimum cycle cycle

A value returned by the performance monitor indicating the number of the minor cycle in which the minimum cycle time has occurred.

minimum cycle frame

A value returned by the performance monitor indicating the number of the major frame in which the minimum cycle time has occurred.

minimum cycle time

A value returned by the performance monitor indicating the least amount of time that an FBS–scheduled process has spent running in a cycle.

minimum frame frame

A value returned by the performance monitor indicating the number of the major frame in which the minimum frame time has occurred.

minimum frame time

A value returned by the performance monitor indicating the least amount of time that an FBS–scheduled process has spent running during a major frame.

minor cycle

The smallest unit of frequency maintained by a frequency–based scheduler. A minor cycle has associated with it a duration, which is the time that elapses between interrupts generated by the timing source that is attached to the scheduler. If the timing source is a real–time clock, the minor cycle duration is defined by specifying the number of clock counts per minor cycle and the number of microseconds per clock count.

number of overruns

A value returned by the performance monitor indicating the number of times that an FBS–scheduled process has caused a frame overrun.

performance monitor

A mechanism that makes it possible to monitor use of the CPU by processes that are scheduled on a frequency–based scheduler.

period

A frequency–based scheduler scheduling parameter that specifies the frequency with which a specified program is to be wakened in each major frame. A period of one indicates that the program is to be wakened every minor cycle; a period of two indicates that it is to be wakened once every two minor cycles; and so on.

privilege

A mechanism through which processes are allowed to perform sensitive operations or override system restrictions.

process dispatch latency

The time that elapses from the occurrence of an external event, which is signified by an interrupt, until the process that is waiting for that external event executes its first instruction in user mode.

scheduler key

A user–supplied numeric identifier for a frequency–based scheduler.

shielded processor

A CPU that is responsible for running high–priority tasks that are protected from the unpredictable processing associated with interrupts and system daemons.

slot number

A unique frequency–based scheduler process identifier that is returned when a program is scheduled on a frequency–based scheduler.

spare time

Processor time that is composed of the following: (1) idle time, (2) CPU time of processes that are not scheduled on a frequency–based scheduler, and (3) CPU time of FBS–scheduled processes for which performance monitoring has not been enabled.

starting base cycle

A frequency–based scheduler scheduling parameter that specifies the first minor cycle in which an FBS–scheduled process is to be wakened in each major frame.

timing mode

The mode under which the performance monitor runs. It specifies whether time spent servicing interrupts is to be included in or excluded from performance monitor timing values.

total iterations

A value returned by the performance monitor indicating the number of times that an FBS–scheduled process has been wakened by the scheduler.

total time

A value returned by the performance monitor indicating the total amount of time that an FBS–scheduled process has spent running in all cycles.

unscheduled process

A process that is not wakened by the frequency–based scheduler and does not call **fbswait**; it is not scheduled to run at a certain frequency.

Index

D

E

F

G

H

I

L

M

S

T

U

V

W

Spine for 1.5” Binder

Product Name: 0.5" from top of spine, Helvetica, 36 pt, Bold

Volume Number (if any): Helvetica, 24 pt, Bold

Volume Name (if any): Helvetica, 18 pt, Bold

Manual Title(s): Helvetica, 10 pt, Bold, centered vertically within space above bar, double space between each title

Bar: 1" x 1/8" beginning 1/4" in from either side

Part Number: Helvetica, 6 pt, centered, 1/8" up

PowerMAX OS

Programmer

Guide to Real-Time Services

0890479