PowerWorks Linux Development Environment Tutorial

0898100-000
April 2001

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 33069-4324. 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 -- April 2001	000	PowerMAX OS 4.3

Preface

General Information

The PowerWorks™ Linux Development Environment (PLDE) allows users on a Linux® PC to develop applications for Concurrent real-time computer systems. The PLDE provides cross compilation, cross linking, and cross debugging and analysis tools. Editing, compilation, linking, and scheduling, as well as debug and analysis sessions, are hosted on the Linux system while the application programs execute on a system running Concurrent's PowerMAX OS™ real-time UNIX®-based operating system.

The PowerWorks Linux Development Environment consists of high-performance Ada95 and C/C++ compilers, the NightView™ symbolic debugger, NightTrace™ event analyzer, NightSim™ frequency-based scheduler, and the NightBench™ GUI program development environment.

Utilizing the PLDE utilities on a Linux system while targeting the PowerMAX OS system offloads the heavy processing associated with compilation, linking, symbolic debug translation, and GUI network traffic from the real-time target systems.

Scope of Manual

This manual is a tutorial for the PowerWorks Linux Development Environment.

Structure of Manual

This manual consists of one chapter which is the tutorial for the PowerWorks Linux Development Environment.

Syntax Notation

The following notation is used throughout this guide:

italic	Books, reference cards, and items that the user must specify appear in *italic* type. Special terms and comments in code may also appear in *italic.*
`list bold`	User input appears in **`list bold`** type and must be entered exactly as shown. Names of directories, files, commands, options and man 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 appears in `list` type. Keywords also appear in `list` type.

<u>emphasis</u>	Words or phrases that require extra emphasis use <u>emphasis</u> type.
`window`	Keyboard sequences and window features such as push buttons, radio buttons, menu items, labels, and titles appear in `window` type.
`[ ]`	Brackets enclose command options and arguments that are optional. You do not type the brackets if you choose to specify such option or arguments.
`{ }`	Braces enclose mutually exclusive choices separated by the pipe (`|`) character, where one choice must be selected. You do not type the braces or the pipe character with the choice.
`...`	An ellipsis follows an item that can be repeated.
`::=`	This symbol means *is defined as* in Backus-Naur Form (BNF).

Referenced Publications

The following publications are referenced in this document:

0890395	*NightView User's Guide*
0890398	*NightTrace Manual*
0890458	*NightSim User's Guide*
0890514	*NightBench User's Guide*
0890516	*MAXAda Reference Manual*

Contents

Chapter 1 Using the PLDE

Illustrations

1 Using the PLDE

Concurrent's PowerWorks® Linux Development Environment (PLDE) allows users on a Linux® PC to develop applications for any Concurrent real-time computer system. The PLDE makes it easy to utilize the features of Concurrent compilers and real-time GUI tools. Application programs are compiled and debugged directly on a Linux PC while targeted to a system running Concurrent's PowerMAX OS™ real-time UNIX-based operating system.

The PowerWorks Linux Development Environment consists of high-performance C/C++ and MAXAda™ (Ada95) compilers, the NightView™ symbolic debugger, NightTrace™ event analyzer, NightSim™ frequency-based scheduler, and the NightBench™ Program Development Environment.

Overview

This is a demonstration of the PowerWorks Linux Development Environment. In this tutorial, we will use many of the PLDE tools including:

- NEdit
- NightBench
- MAXAda
- NightSim
- NightView
- NightTrace

integrating them together into one cohesive example.

Please see “Before you begin” on page 1-1 for some important recommendations and considerations.

Before you begin

In order to run the portion of the tutorial that uses the NightSim Scheduler and the NightView Source-Level Debugger, a system running PowerMAX OS should be networked to your Linux system. If you have a PowerMAX OS system networked to your Linux system, the following items must also be taken into consideration:

- Pathname conventions
- Remote shell access
- Privileges
- Additions to PATH

Proceed to “Getting Started” on page 1-5 to begin the tutorial.

NOTE

You may still run the tutorial (excluding the portions that use the NightSim Scheduler and the NightView Source-Level Debugger) even if you do not have a system running PowerMAX OS networked to your Linux system. You will be instructed as to how to skip over the sections that use the NightSim Scheduler and the NightView Source-Level Debugger.

Pathname conventions

It is *highly recommended* that the paths from the host system (the system running Linux) and the target system (the system running PowerMAX OS) to the executables and working directories be identical.

Their mount points should be based on a common name.

Consider the following entries showing a Linux filesystem mounted via NFS on a PowerMAX OS system.

The following entry is located in **/etc/fstab** on the Linux system:

```
/dev/hda5       /myspace          ext2    defaults        1       2
```

and the following entry is located in **/etc/vfstab** on the PowerMAX OS system

```
linuxsys:/myspace    - /myspace    nfs     -       yes     rw,bg,soft
```

where `linuxsys` is the name of the host system running Linux.

In the above example, the user would create their working directory under **/myspace**. Creating a working directory is covered in “Getting Started” on page 1-5.

Ensure that your **umask** setting on the Linux system will allow the PowerMAX OS system to read and write files in your working directory, or use the same user and group ID on both systems. To automatically ensure that all files your user creates on the Linux system are publicly readable and writeable, include the following command in your shell startup script:

```
umask 000
```

See “Before you begin” on page 1-1 for other important recommendations and considerations.

Remote shell access

Since NightSim uses **`rsh`** to start the *NightSim server* process on each *target system*, the user must be able to **`rsh`** to those systems.

Ensure that a login for your user name exists on the target system **and**

- the **`.rhosts`** file in the home directory for that user name on the target system contains an entry for your user name and the *NightSim host*

  An example entry might look like:

  ```
  remote_machine_name username
  ```

  where `remote_machine_name` is the name of the target system and `username` is your login name on the remote system.

  Also note that the **`.rhosts`** file must have the permissions 644.

  or

- the **`/etc/hosts.equiv`** file on the target system contains the name of the NightSim host

You may test your remote shell access by issuing the following command from your host system (the system running Linux):

```
/usr/bin/rsh remote_machine_name date
```

where **`remote_machine_name`** is the name of the target system. You should see the date and time on the remote system if successful.

See the **`rsh(1)`** man page for more details.

See “Before you begin” on page 1-1 for other important recommendations and considerations.

Privileges

For the sections of the tutorial that run on the target system (the portions that use the NightSim Scheduler and the NightView Source-Level Debugger), this tutorial requires that the user have the following privileges on the target system:

- P_CPUBIAS
- P_PLOCK
- P_RTIME

A convenient way to associate privileges with users is through the use of roles. A role is simply a named description of a set of privileges that have been registered for certain executable files, such as the shell. The system administrator creates roles and assigns users to them. During the login process, users can request that their shell be granted the privileges associated with their role. Such a request takes the form of an invocation of the **tfadmin(1M)** command. Once privileges have been granted to the user's shell, subsequently spawned processes automatically inherit those privileges.

The following commands create a role and register all the privileges required by this tutorial to three commonly used shells (**sh**, **ksh**, and **csh**). The PowerMAX OS system administrator should issue the following commands once.

```
/usr/bin/adminrole -n PLDE_USERS
/usr/bin/adminrole -a sh:/usr/bin/sh:cpubias:plock:rtime PLDE_USERS
/usr/bin/adminrole -a ksh:/usr/bin/ksh:cpubias:plock:rtime PLDE_USERS
/usr/bin/adminrole -a csh:/usr/bin/csh:cpubias:plock:rtime PLDE_USERS
```

The following command assigns an example user (JoeUser) to the PLDE_USERS role. The system administrator should issue the following command once.

```
/usr/bin/adminuser -n -o PLDE_USERS JoeUser
```

JoeUser is now allowed to request that the above privileges be granted to his shell (assuming JoeUser utilizes either the sh, ksh, or csh shell, as these are the only shell commands registered in the PLDE_USERS role). However, by default, these privileges are not granted. He must explicitly make the request by initiating a new shell with the **tfadmin(1M)** command. For convenience, it is recommended that the following command be added to the end of his **.profile** (or **.login** for csh users) file. (This file is executed during initialization of the login shell).

exec /sbin/tfadmin PLDE_USERS: *shell*

where *shell* is the shell of your choice (**sh**, **ksh**, or **csh**).

See “Before you begin” on page 1-1 for other important recommendations and considerations.

Additions to PATH

If users are interested in doing command-line compilations (although they are not covered in this tutorial), the following should be added to their PATH:

```
/usr/ada/bin
/usr/ccs/bin
```

See “Before you begin” on page 1-1 for other important recommendations and considerations.

Getting Started

We will start by creating a directory in which we will do all our work. On the Linux system, create a directory and position yourself in it:

To create a working directory

- Use the **mkdir(1)** command to create a working directory. See “Pathname conventions” on page 1-2 for recommendations as to where you should create your working directory.

  We will name our directory **tutorial** using the following command:

  ```
  mkdir tutorial
  ```

- Position yourself in the newly created directory using the **cd(1)** command:

  ```
  cd tutorial
  ```

Using NEdit

Next, we will create one of the source files that will be used by our example program. We will do this using the NEdit Editor. NEdit is the PowerWorks Linux Development Environment editor. Although other editors may be used, NEdit comes with the PLDE and thus will be demonstrated in this tutorial.

Let’s open the NEdit editor.

To start NEdit

- From the command line, type the following command:

  ```
  neditor -ktalk &
  ```

NOTE

The **-ktalk** option allows NightBench to communicate with this NEdit session later in this tutorial. We specify the **&** so that this NEdit session runs in the background.

The NEdit Editor will be opened, ready to accept input.

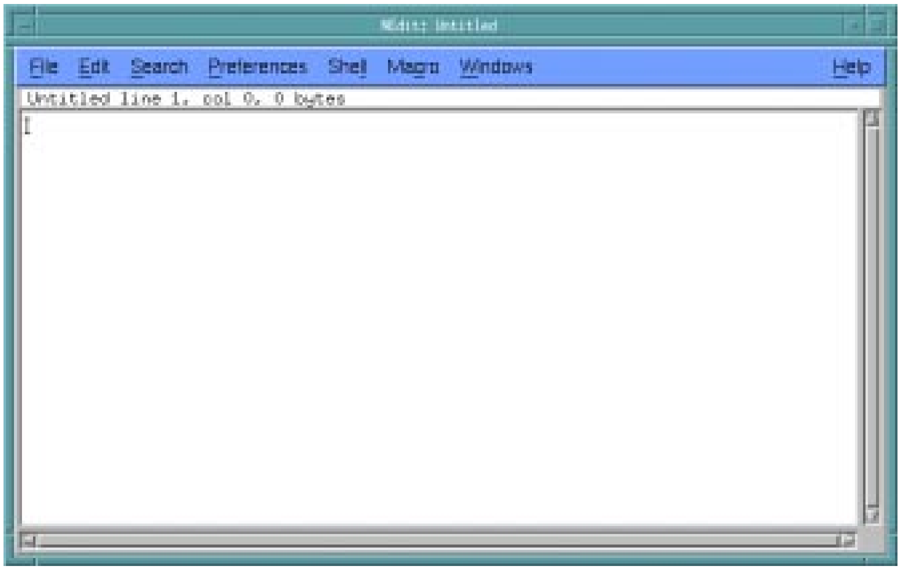

Figure 1-1. NEdit Editor

We will enter one of the source files for our example program. This program is written in Ada and is shown below:

```
with data_process;
with rt_interface;

procedure prog is
   istat : integer;
   i : integer;
begin

   i := 0;

   rt_interface.FBS_wait( istat );

   while  istat = 0   loop

      data_process.do_work;
      rt_interface.FBS_wait( istat );
      i := i + 1;

   end loop;

end;
```

This program utilizes the `FBS_wait` service. `FBS_wait` causes the calling process to go to sleep. The process will be awakened by a frequency-based scheduler at the process's scheduled frequency. At that point, it will enter the loop. The procedure `data_process.do_work` (which we will create later) will do some calculations. When `do_work` returns from its processing, the program will encounter another `FBS_wait` call which will cause the program to sleep until the frequency-based scheduler allows it continue.

To save an untitled file using the NEdit Editor

- Select Save from the File menu. This will open a file dialog.
- Ensure the Directory is the same as the one you created in “Getting Started” on page 1-5.
- Enter the name **prog.a** in the Save File As field.
- Press OK.

Now that we have saved the file, we may close it in our NEdit session.

To close a file in NEdit

- Select Close from the File menu.

We are finished using NEdit for this portion of the tutorial but will be using it in the next section with NightBench. We may leave this session open so that NightBench can communicate with it. However, if we chose to exit out of NEdit, NightBench would start its own session of NEdit, if necessary.

Using NightBench

In order to compile and link our program, we will use the NightBench Program Development Environment. NightBench is a graphical user interface that provides a common work environment for the PowerWorks Linux Development Environment editor, compilers, and development tools. NightBench organizes all of the information required for consistent, repeatable development of PowerMAX OS applications while providing an efficient interface for editing, browsing, building, and debugging.

Let's open the NightBench Project window.

To start NightBench

- From the command line, type the following command:

  nbench

Note that we have not provided **nbench** with any parameters, indicating that we want to open the NightBench Project window. **nbench** accepts a number of command line options, allowing the user to open a particular NightBench component or to provide start-up information to NightBench.

The NightBench Project window will appear, listing the environments with which it has previously interacted. If no other environments have been created under NightBench, this list will be empty.

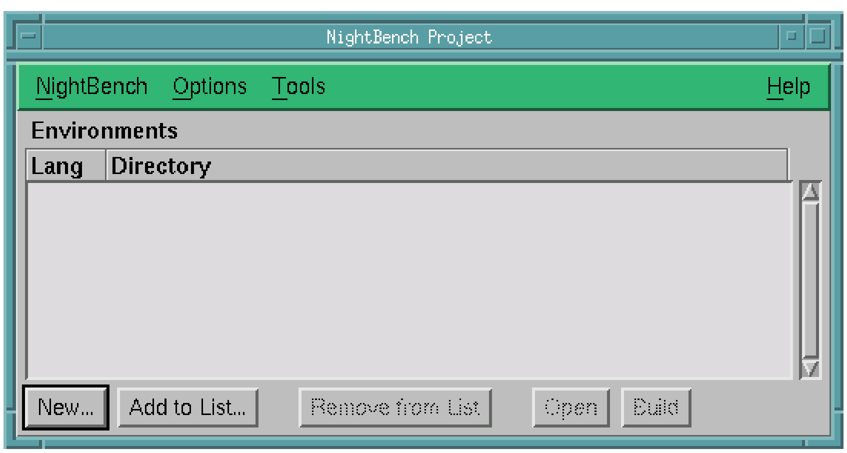

Figure 1-2. NightBench Project

Creating a new environment

One of the first steps we must take in order to use NightBench for program development is to create an *environment*. Environments are used as the basic structure of organization within NightBench.

To create a new environment from the NightBench Project window

- On the NightBench Project window, press the button marked New… so we can create our new environment. This will open a dialog in which you may select the language to be used in this environment.
- Select the language that will be used in this environment (Ada).
- Press the Next> button.

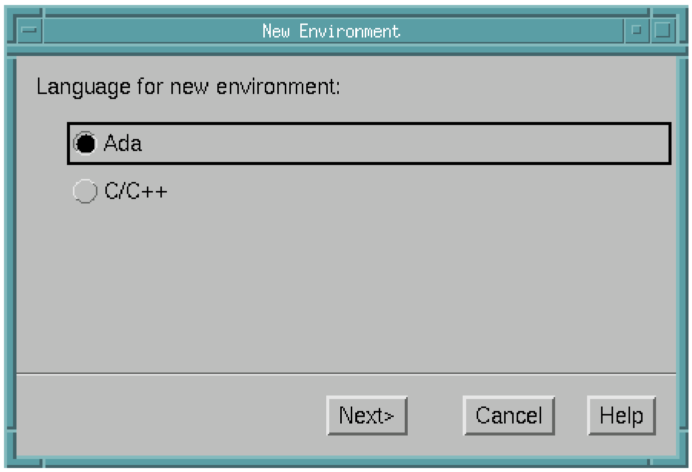

Figure 1-3. Creating a new environment - language selection

The next dialog presented allows us to specify details about the directory which will contain the new environment, the release of the compiler to be used, as well as the architecture of the target machine and the version of PowerMAX OS running on it:

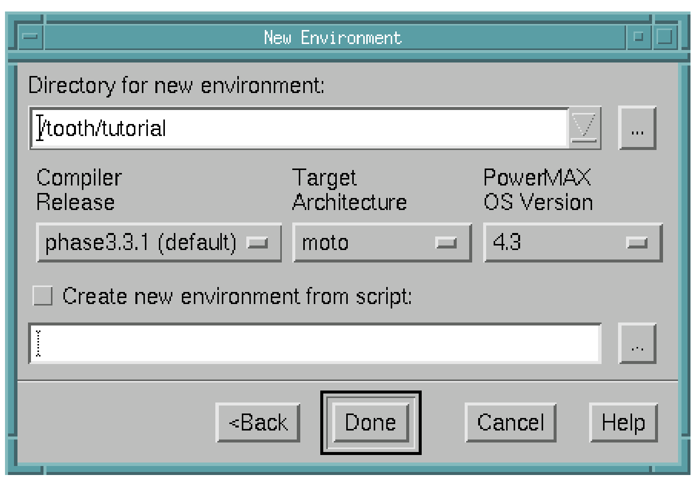

Figure 1-4. Creating a new environment - specifications

- Type the directory name in the Directory for new environment field where you want NightBench to create the new environment. This can be the name of an existing directory or NightBench can create the directory for you. (Note that NightBench can only create a subdirectory of an existing directory.) We will enter the name of the directory we created in "Getting Started" on page 1-5. The full directory name in our example is **`/tooth/tutorial`**. Since we invoked NightBench from that directory, the pathname will appear in the Directory for new environment field.

- Select a Compiler Release if you have more than one release of MAXAda installed on your system. If you have only one release of MAXAda installed on your system, it will appear here.
- Choose a Target Architecture. Because we are building an executable that will run on a Concurrent real-time computer system, we must choose which type of system we are targeting. For our example, we will be targeting a Power Hawk™ 640 so we will select moto from the drop-down list. For more information on target architectures, see the section titled "Target Architectures" in the *MAXAda Reference Manual* (0890516).
- Identify the PowerMAX OS Version. In our example, we will select 4.3 from the drop-down list for the version of the operating system running on the system we are targeting.
- Press Done

This will add the new environment to the list of Environments in the NightBench Project window. NightBench will also open the new environment in its own NightBench Development window.

Introducing an existing source file into the environment

Our next step is to populate the environment with *units*. Units are the basic building blocks for programs in NightBench. They are contained within source files and it is through these source files that they are introduced into their intended environments.

Source files may already have been created outside the NightBench environment or you may use the editing features of NightBench to create a new file. For this example, we will introduce the file, **`prog.a`**, that we created earlier in "Using NEdit" on page 1-5.

To introduce an existing source file into a NightBench environment

- Click on the Source Files tab of the NightBench Development window.
- Press the Introduce/Create... button. This will open the Introduce Source Files dialog so we can introduce our source files (and the units contained within) into the new environment.

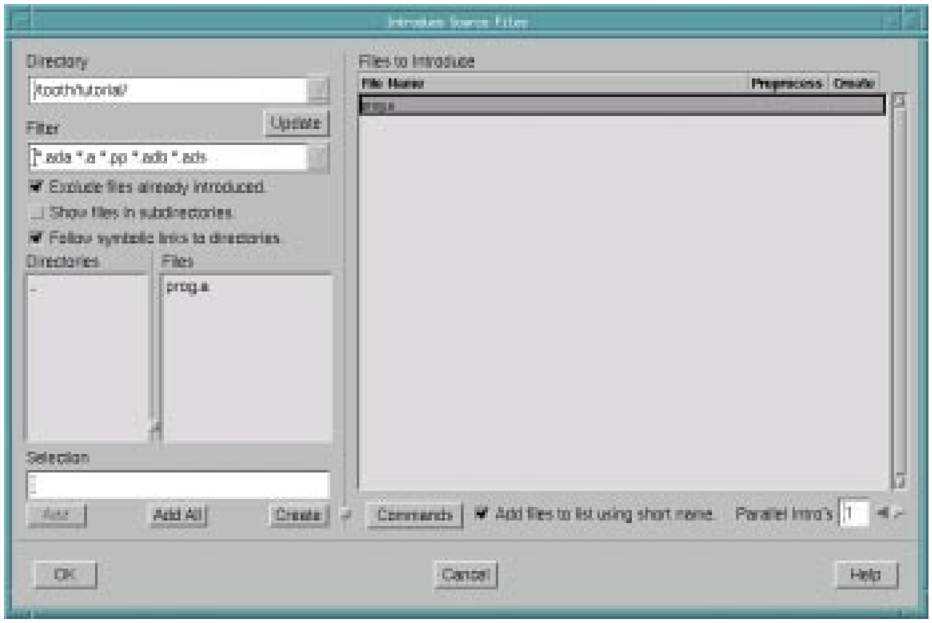

Figure 1-5. Introducing an existing source file

- Maneuver to the directory in which the **`prog.a`** source file is contained. You may type the path to the directory name in the Directory field or use the entries in the Directories list to navigate to the desired directory. Since we invoked NightBench from the directory in which our file resides, we should already be positioned in that directory.
- Select the **`prog.a`** source file by clicking once with the mouse on the name in the Files list. The name of the source file will then appear in the Selection field.
- Press the Add button to add this file to the list of Files to Introduce. (You may introduce any number of files, or create new files, by adding them here but for our example we will just introduce this one file.)
- Press the OK button to introduce the source file into the environment.

The unit `prog` that was contained in the source file **`prog.a`** is now a part of the environment **`/tooth/tutorial`**.

The source file now appears in the list of files on the Source Files page of the NightBench Development window and the unit contained within now appears on the Units page.

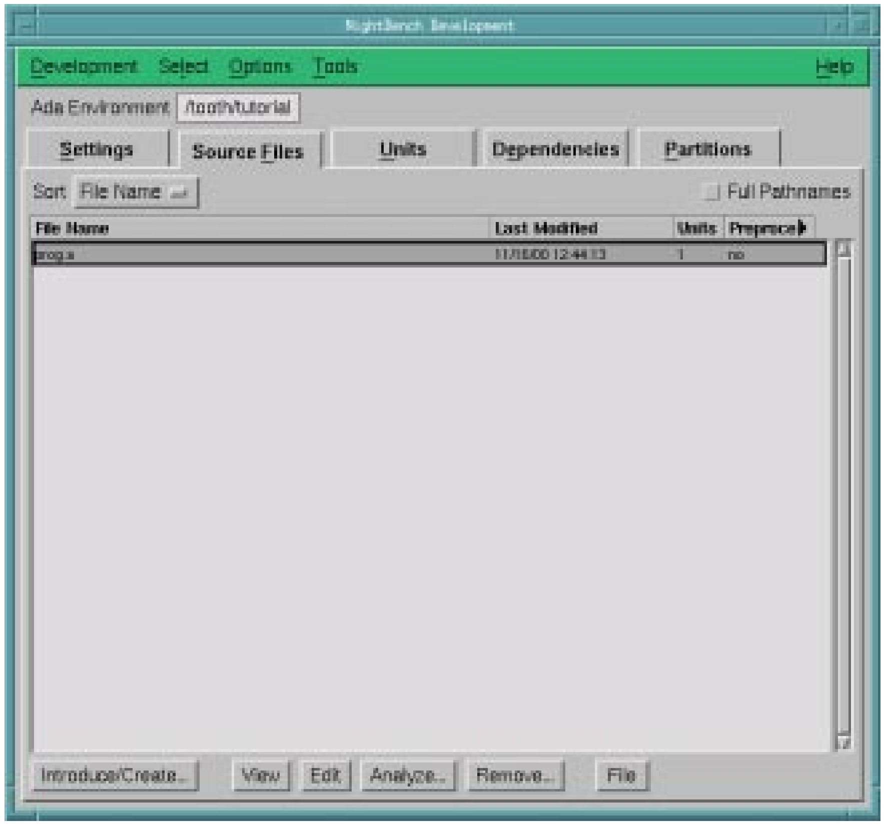

Figure 1-6. Source file, `prog.a` - newly introduced

Creating a new source file in the environment

As mentioned earlier, source files may already have been created outside the NightBench environment or you may use the editing features of NightBench to create a new file. In this part of the tutorial, we will create a new source file using the editing features of Night-Bench.

To create a new source file in a NightBench environment

- Click on the `Source Files` tab of the NightBench Development window.
- Press the `Introduce/Create...` button. This will open the `Introduce Source Files` dialog so we can create new source files and introduce them into the new environment.

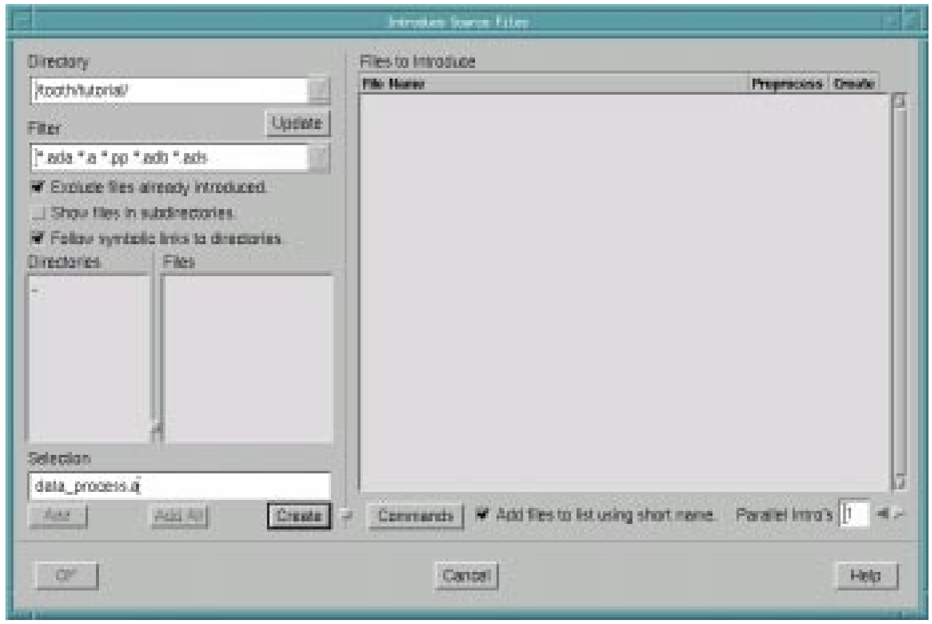

Figure 1-7. Creating a new source file

- Maneuver to the directory in which you would like to create the new source file. You may type the path to the directory name in the `Directory` field or use the entries in the `Directories` list to navigate to the desired directory. Since we invoked NightBench from the directory in which we would like to create our new source file, we should already be positioned in that directory.

- Enter the name of new source file in the Selection field. For our example, we will name our file **`data_process.a`**

- Press the `Create` button. This will add the file name to the list of of `Files to Introduce`. Note the `Yes` in the `Create` column for this file. (You may create other new source files, or introduce other existing source files, by adding them here but for our example we will just create this one file.)

- Press `OK`.

For each file with a `Yes` in the `Create` column of the list of `Files to Introduce`, an editor window will be opened. This will bring up the editor that NightBench is configured to use. NEdit is the default editor for NightBench. See the section titled “Preferences - Editor” in the *NightBench User’s Guide* (0890514).

We may now enter the other source file used in our example program:

```
package data_process is

   iteration_count : integer := 1;
   x1,x2 : long_float;
   x1_mult_x2 : long_float;
   procedure do_work;

end data_process;


package body data_process is

   procedure do_work is
   begin
      x1 := 1.0e-160;
      x2 := 1.0e-160;

      iteration_count := iteration_count + 1 ;

      for i in 1 .. 2000  loop
         x1_mult_x2 := x1 * x2 ;
      end loop;
   end;

end data_process;
```

To save a named file using the NEdit Editor

- Select Save from the File menu. Since we specified the pathname of the file to NightBench, NEdit saves our input in that file.

Now that we have saved the file, we may close it in our NEdit session.

To close a file in NEdit

- Select Close from the File menu.

Setting compile options

In order to debug the program using the NightView Source Level Debugger, we need to compile the program with debug information. We do this by setting an environment-wide compile option which will apply to all units within the current environment.

To set environment-wide compile options

- Click on the `Settings` tab of the NightBench Development window.
- Press the `Show Options Editor` button associated with the `Permanent Compile Options`.
- On the `General` page of the `Ada Environment Compile Options` dialog, select `full (2)` from the drop-down list under the `Permanent` column for `Debug Information`.
- Press `OK`.

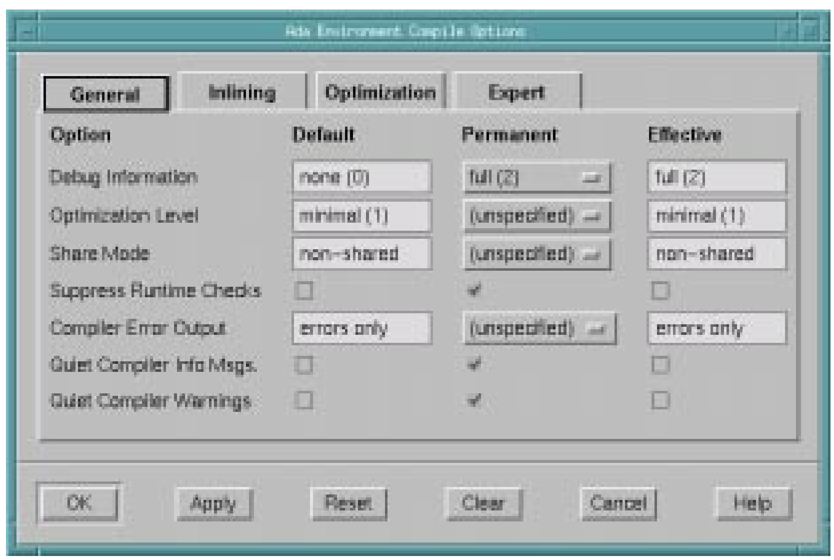

Figure 1-8. Setting environment-wide compile options

NOTE

Alternatively, you could have entered **-g** in the `Permanent Compile Options` field on the `Settings` page and pressed the `Apply` button.

Defining a partition

In order to use the units introduced into NightBench, we must include them in a *partition*. NightBench defines three types of Ada partitions:

- *active*
- *archive*
- *shared object*

For our example, we want to include our `prog` unit in an executable program so we will be defining an *active* partition.

To define all active partitions in the environment

- Click on the Partitions tab of the NightBench Development window to get to the Partitions page.
- Press the Create All button. This creates an active partition for each unit in the current environment that qualifies as a main unit.

Activating tracing for a partition

We will need to activate tracing for this partition so that we may generate trace data when we run the program and then subsequently analyze it using the NightTrace Analyzer.

To activate tracing for a partition

- Click on the Partitions tab of the NightBench Development window to get to the Partitions page.
- Click on the Tracing settings tab.
- Check the Activated checkbox.
- Select the Mechanism to be used for tracing. In our example, we will select `ntraceud` from the drop-down list so that the Ada run-time executive can log trace events using the NightTrace user daemon, **`ntraceud`**.
- Press Apply.

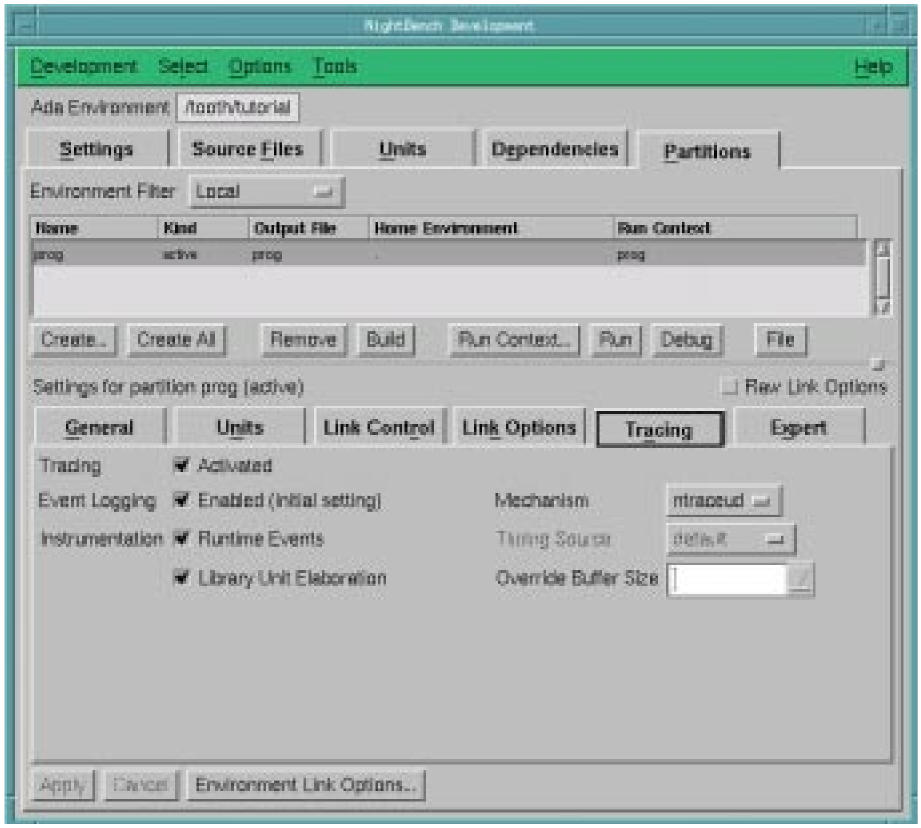

Figure 1-9. Activating tracing for a partition

Building a partition

At this point, we have an environment, **`/tooth/tutorial`**, that has within it the definition for the active partition, **`prog`**, made up of a main unit, `prog`, contained in the source file, **`prog.a`**, and another unit `data_process`, contained in the source file, **`data_process.a`**. Full debug information will be generated for the program and tracing has been activated so that we may gather tracing data for later analysis.

We can now build this partition. We do this using the NightBench Builder.

To build the partition

- In the list of partitions on the `Partitions` page, make sure the partition **`prog`** is selected.
- Press the button marked `Build`. This will open the NightBench Builder window so we can build our new partition.

  In Figure 1-10, you will see that `partition prog` has been automatically entered in the `Targets` field on the `Build` page. This is because it was selected on the `Partitions` page when the `Build` button was pressed.

- Press Start Build.

  The Build Progress bar shows the number of actions (compilations and links) left to perform in the current build as the Transcript window details each step taken during the build.

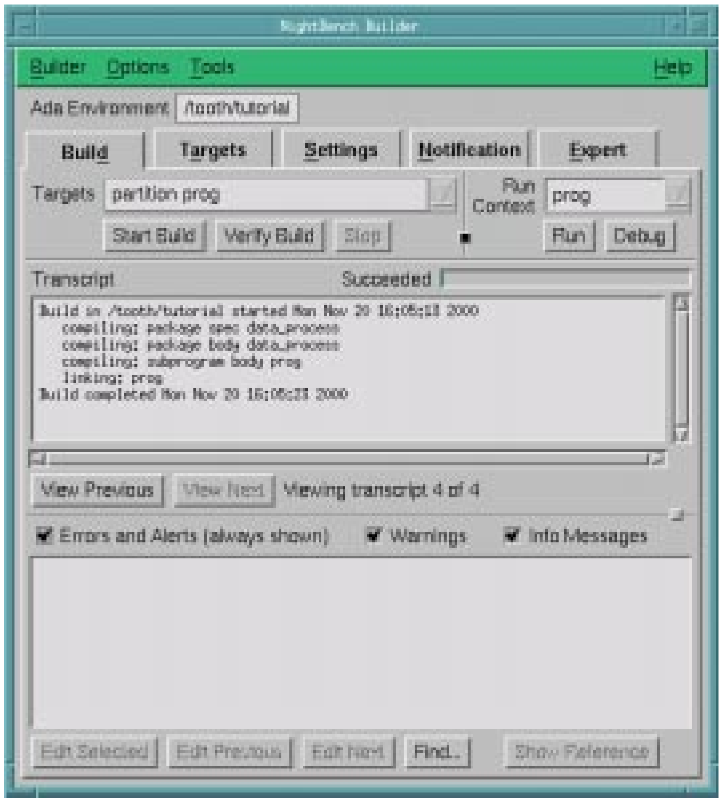

Figure 1-10. Builder window - Build page for prog partition

When the build is completed, a Build Completed dialog notifies the user.

NOTE

> The notification operations can be changed on the Notification page.

Before you continue

The following sections require a PowerMAX OS system networked to your Linux system since those sections use the NightSim Scheduler and the NightView Source-Level Debugger (see "Before you begin" on page 1-1 for important recommendations and considerations concerning this configuration).

However, if you do not have a PowerMAX OS system networked to your Linux system, you may jump to the section “Using NightTrace” on page 1-39 and continue with the tutorial.

Invoking NightSim

Because this program uses the frequency-based scheduler, we will use the NightSim Scheduler to schedule the process.

NightBench allows the user to invoke the NightSim Scheduler directly.

To invoke the NightSim Scheduler from the NightBench Program Development Environment

- Select NightSim Scheduler from the Tools menu of either NightBench Development or the NightBench Builder.

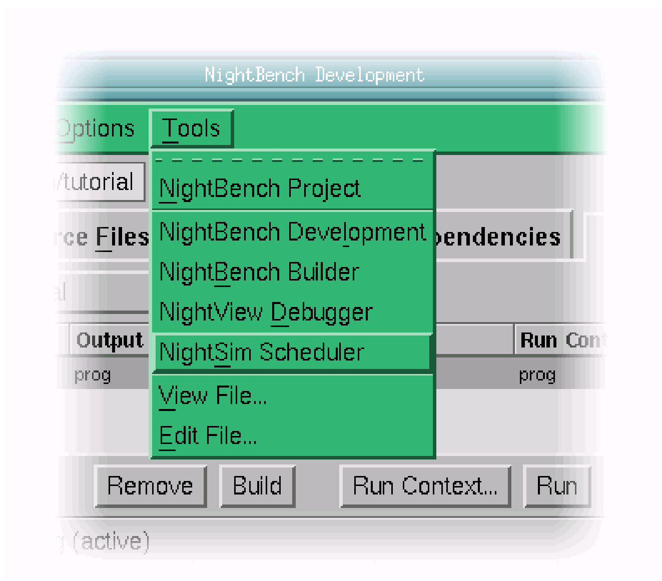

Figure 1-11. Starting NightSim from NightBench

Using NightSim

NightSim is a tool for scheduling and monitoring real-time applications which require predictable, repetitive process execution. NightSim provides a graphical interface to the PowerMAX OS frequency-based scheduler and performance monitor. With NightSim, application builders can control and dynamically adjust the periodic execution of multiple coordinated processes, their priorities, and their CPU assignments. NightSim's performance monitor tracks the CPU utilization of individual processes and provides a customizable display of period times, minimums, maximums, and frame overruns. For more information on NightSim, refer to the *NightSim User's Guide* (0890480).

Configuring the Scheduler

The NightSim Scheduler window is opened, ready for us to configure it for our particular simulation.

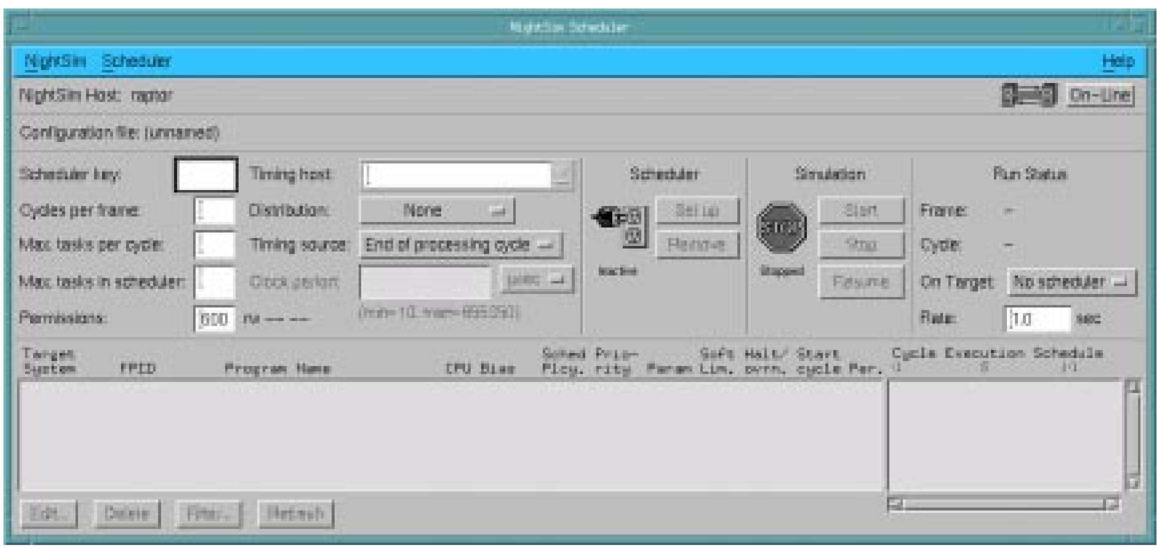

Figure 1-12. NightSim Scheduler

To configure a NightSim Scheduler

- Specify a `Scheduler key`. The key is a user-chosen numeric identifier with which the scheduler will be associated. For our example, we will use `100`.
- Specify the `Cycles per frame`. This field allows you to specify the number of cycles that compose a frame on the specified scheduler. We will use the value `1`.
- Specify the `Max. tasks per cycle`. This field allows you to specify the maximum number of processes that can be scheduled to execute during one cycle. Enter `10` for our example.

- Specify the Max. tasks in scheduler. This field allows you to specify the maximum number of processes that can be scheduled on the specified scheduler at one time. For our example, we will specify the value 10.
- Enter the name of a PowerMAX OS system which will act as the Timing host for the simulation. You may use the drop down list associated with this field for the names of systems previously used as timing hosts. For our-example, we will enter buzzard, a Power Hawk 640 system.

NOTE

When NightSim is operating in On-Line mode, an attempt will be made to communicate with the system specified as the timing host. The user may experience a slight delay and the message Talking to Server... will appear in the Configuration File Name Area of the NightSim Scheduler as this occurs. See the *NightSim User's Guide* (0890480) for more information.

- Select a Timing source from the list provided. This list contains the set of devices available on the timing host. We will use Real-time clock 0c2.

NOTE

Do not use Real-time clock 0c0 for the Timing source as it is typically used by system utilities and could cause unwanted effects if used. See **hrtconfig(1)** for more information

Since we are using the real-time clock on the target system, we need to specify the clock period. For our simulation, we would like the real-time clock to “fire” every .5 seconds (or 500 milliseconds).

IMPORTANT

The following steps should be performed in the order presented below to ensure the correct value for the clock period.

- Choose the msec from the drop-down list next to the Clock period field.
- Specify Clock period. For our example, we will specify 500 for the number of milliseconds.

Scheduling a process

Once we have properly configured the Scheduler, we can add a process to the frequency-based scheduler.

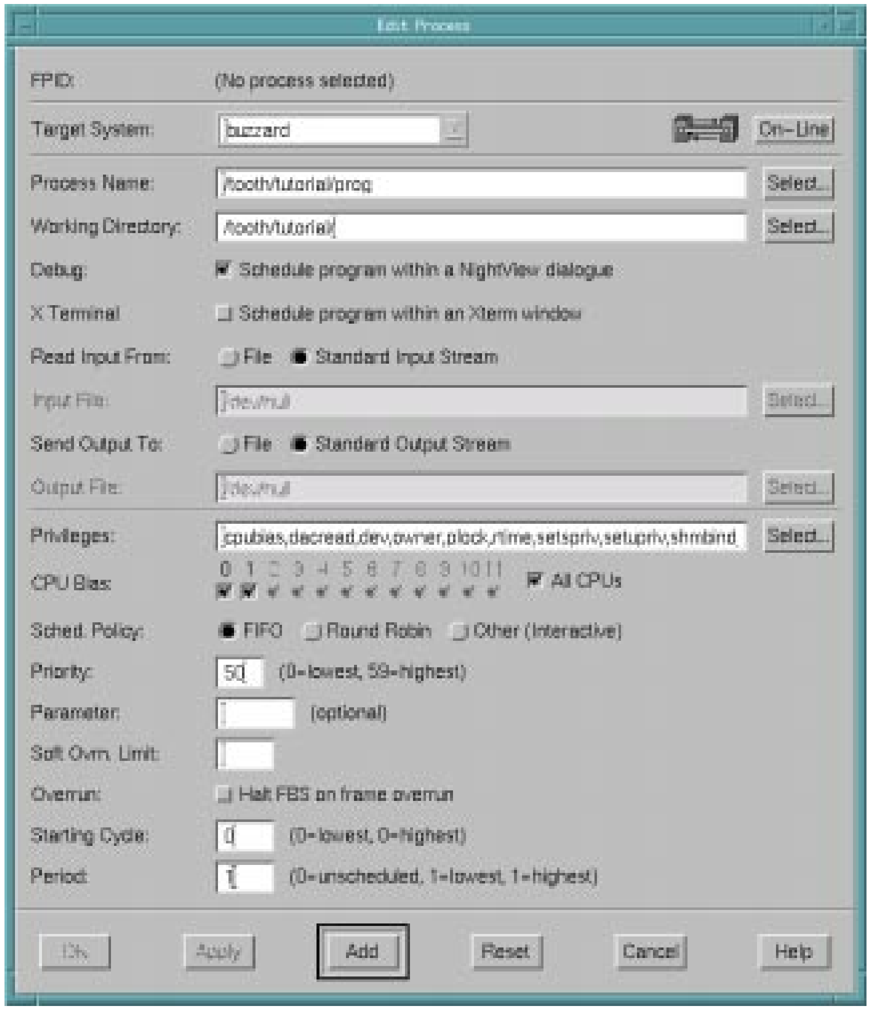

Figure 1-13. NightSim Edit Process

To add a process to the frequency-based scheduler

- Press the `Edit...` button on the NightSim Scheduler window. This will bring up the `Edit Process` window.
- Press the `Select...` button next to the `Process Name` field. This brings up the `Select a Program` dialog.
  - Since NightSim was invoked from our NightBench environment, the `Directory` field should coincide with our working directory. If it does not, either type the full pathname to our working directory,

/tooth/tutorial, in the Directory field, or maneuver to that directory using the items in the Directories list.

- Choose the program we wish to schedule from the Files list. For our example, we will select prog from the list.
- Press OK to select the program.

- Ensure that the Working Directory is the same directory that contains our program (the directory of the Process Name selected in the previous step).
- Check the Schedule program within a NightView dialogue checkbox. This will bring the program up in the NightView debugger before the program executes, allowing us to set *tracepoints* so that we may generate trace data when the program executes.
- Specify the Priority for this process. The range of priority values that you can enter is governed by the scheduling policy specified. NightSim displays the range of priority values that you can enter next to the Priority field. Higher numerical values correspond to more favorable scheduling priorities. For our example, we will give the process a priority of 50.
- Select Starting Cycle. This field allows you to specify the first minor cycle in which the specified program is to be wakened in each major frame. We will choose the lowest value, 0, for our example.
- Select Period. This field allows you to establish the frequency with which the specified program is to be wakened in each major frame. Enter the number of minor cycles representing the frequency with which you wish the program to be wakened. For our example, we will specify a period of 1, indicating that the specified program is to be wakened every minor cycle.
- Press Add to add the process to the frequency-based scheduler.
- Press the Close button to dismiss the Edit Process window.

Activating user tracing and kernel tracing

At this point in the tutorial, we are about to create the scheduler configured according to the parameters we just specified and allow the program to run. However, we would like to generate trace data from this program while it is running so we need to start the NightTrace user daemon (which we specified in the section titled “Activating tracing for a partition” on page 1-16) to log user trace events as well as KernelTrace which will collect data about the execution time of interrupts, exceptions, system calls, context switches, and I/O to various devices.

To activate the NightTrace user daemon

- Log into the PowerMAX OS system where you will be running your simulation. This is the system we specified as the Timing host in the NightSim Scheduler window (see “Configuring the Scheduler” on page 1-20). So, for our example, we will log into the system **buzzard**.

- Position yourself in the working directory you created in “Getting Started” on page 1-5. Also, see “Pathname conventions” on page 1-2 for recommendations as to where you should create your working directory.

IMPORTANT

It is essential that you are positioned in the working directory that is associated with the user program being scheduled with NightSim. The NightTrace user daemon will communicate with the user program based on the file argument supplied in the next step.

- Invoke the NightTrace user daemon. We issue the **`ntraceud`** command which takes as an argument the name of a file in which to save the trace data. This file should be named *program_name*__`.trace.data`__, where *program_name* is the name of the program generating the trace data.

NOTE

By default, **`ntraceud`** requires write access to system SPL devices, e.g. **`/dev/spl`**, **`/dev/spl1`**, etc. On most systems, these devices are only writeable by the root user; therefore, you should run the **`ntraceud`** command as root.

However, since the use of SPL devices is not strictly necessary for tracing single-threaded user applications (although, for optimal real-time performance it is recommended), the **`-ipldisable`** option to **`ntraceud`** is acceptable.

Since the application in this tutorial is single-threaded, you may use the **`-ipldisable`** option as indicated below.

For our example, we will issue the following command:

```
ntraceud -ipldisable prog.trace.data
```

Now we can activate kernel tracing.

To activate kernel tracing

- Log into the PowerMAX OS system where you will be running your simulation. This is the system we specified as the Timing host in the NightSim Scheduler window (see “Configuring the Scheduler” on page 1-20). So, for our example, we will log into the system **`buzzard`**.
- Position yourself in a directory local to the PowerMAX OS system.

- Invoke the KernelTrace utility. We issue the **`ktrace`** command which can take a number of arguments.

NOTE

The KernelTrace utility requires root access in order to run. Ensure that the Linux system has exported its filesystem to the PowerMAX OS system in a manner which allows root to write on that file system. This normally requires the `root_no_squash` option in the **`/etc/exports`** entry for the file system. For example:

```
/myspace     pmax_system(rw,root_no_squash)
```

Alternatively, you can run the KernelTrace utility on a file system local to the PowerMAX OS system and subsequently copy its output file (as a non-root user) back to Linux file system. The remainder of the steps below assume that root has write access to the Linux file system.

We will use the **`-o`** option which specifies the name of a file in which to save the kernel trace data.

When generating kernel trace data, the resultant file can grow extremely large very quickly. In order to circumvent any problems that may arise from the output file growing extremely large, we will use the **`-bufferwrap`** option which limits the size of the output file. Specifying a value of 50 to this option will limit the size of the resulting output file to a little over 2 megabytes.

NOTE

Due to a problem with the **`-bufferwrap`** option, user and kernel data may not appear synchronized when viewing the trace data in subsequent steps. This problem has been fixed in the **`ktrace`** and **`ntfilter`** commands in PowerMAX OS 4.3 Patch Set 6 (**`trace-004`** and **`base-006`**). If these packages are not installed on your system, you may omit the **`-bufferwrap`** option. However, be aware that the kernel trace file may grow extremely large in a short period of time.

So, for our example, we will issue the following command, as the root user:

```
ktrace -bufferwrap 50 -o prog.ktrace.data
```

You should see output similar to the following:

```
locking into memory
setting priority to RT 59
open /dev/trace
initialize
```

```
set trace event time stamp source to Motorola Time Base
Register
gather trace point data
```

Setting up the scheduler

To set up the scheduler

- In the NightSim Scheduler window, press the `Set up` button.

  This action:

  - creates a scheduler that is configured according to the parameters we specified
  - schedules the processes that we have added to the NightSim Scheduler window and starts them running up to the first `FBS_wait` call, and
  - attaches the timing source to the scheduler.

Because we have specified the `Schedule program within a NightView dialogue` option when we added this process to the frequency-based scheduler (see “To add a process to the frequency-based scheduler” on page 1-22), the NightView Source Level Debugger will be started.

Using NightView

NightView is a graphical source-level debugging and monitoring tool specifically designed for real-time applications. NightView can monitor, debug, and patch multiple real-time processes running on multiple processors with minimal intrusion. In addition to standard debugging capabilities, NightView supports application-speed eventpoint conditions, hot patches, synchronized data monitoring, exception handling and loadable modules.

Because we have specified the Schedule program within a NightView dialogue option when we added this process to the frequency-based scheduler (see "To add a process to the frequency-based scheduler" on page 1-22), we are presented with a NightView Global Window as well as a dialog allowing us to log into the target system on which we will be running our program.

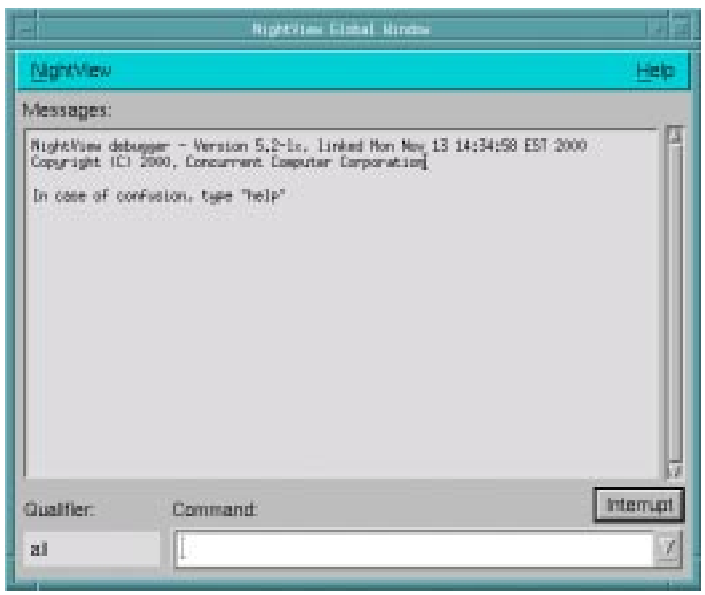

Figure 1-14. NightView Global Window

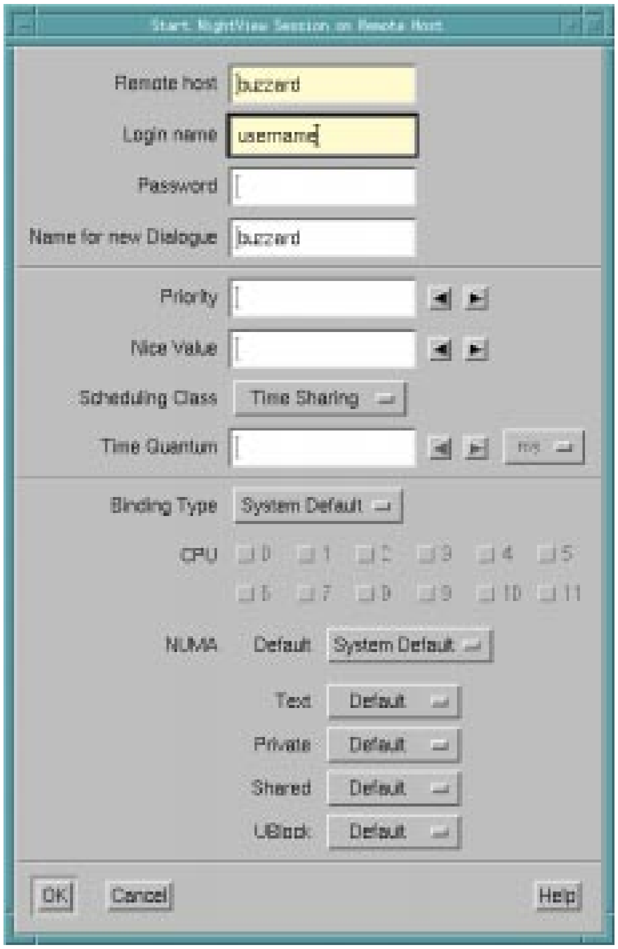

Figure 1-15. Start NightView Session on Remote Host dialog

To start a NightView session on a remote host

- In the Start NightView Session on Remote Host dialog, ensure that the values for Remote host and Login name are correct.

NOTE

> The Name for new Dialogue field is initialized to the name of the remote system on which we are debugging our program.

- Enter your Password for the Login name on the system listed in the Remote host field.
- Press OK.

When the login has completed successfully, a NightView Dialogue window for the remote host will be opened as well as a Principal Debug Window with the execution of the program stopped.

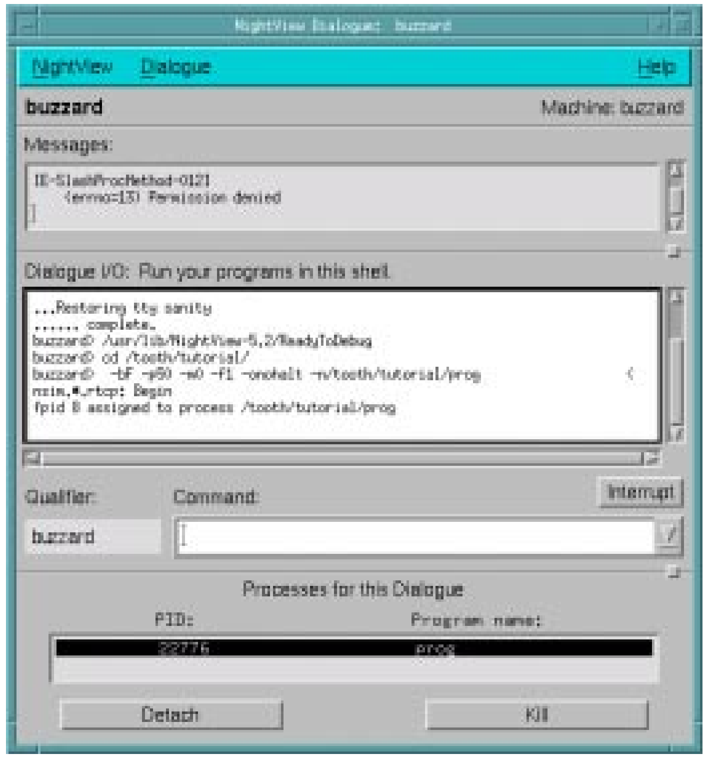

Figure 1-16. NightView Dialogue

During initialization, you will see a message similar to the following:

```
Warning: Process buzzard:3336 is no longer debuggable,
detaching.
[E-SlashProcMethod-012]
    (errno=13) Permission denied
```

This is an anomaly caused by an intermediate process which schedules the user program. You may ignore this warning.

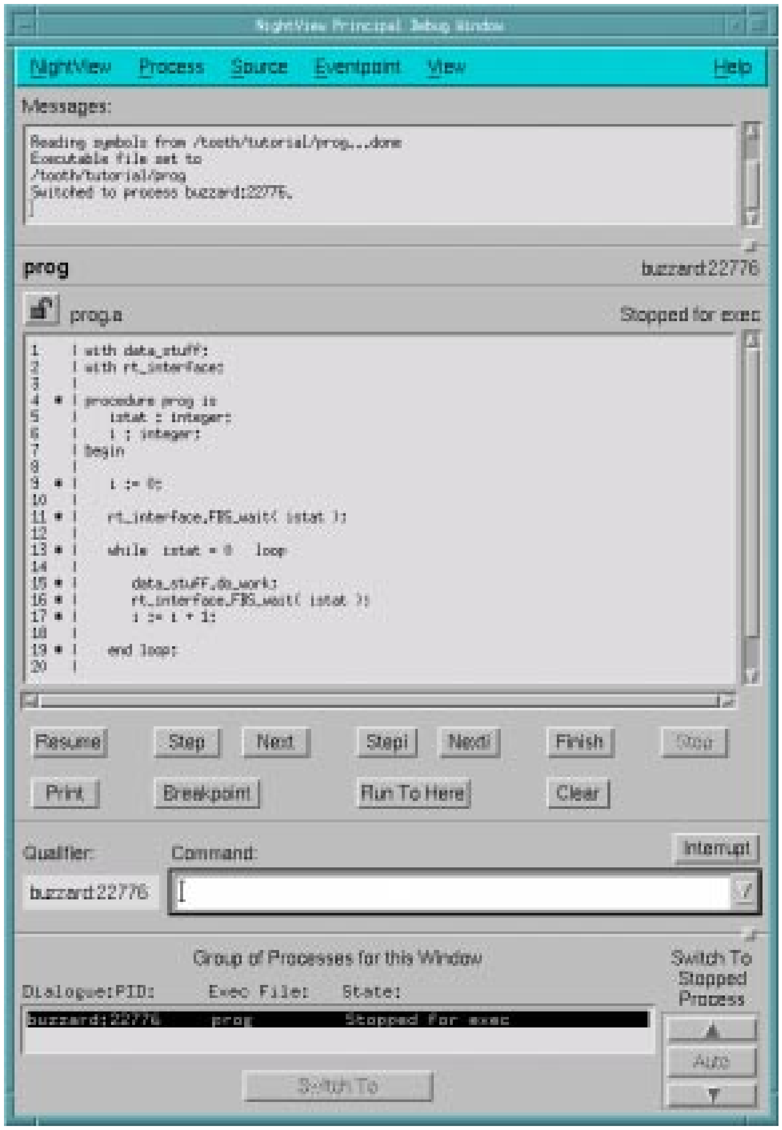

Figure 1-17. NightView Principal Debug Window

Adding a tracepoint in the program

Since we would like to generate user trace data, but did not place any calls within the code before our program was compiled, we can use NightView to insert a *tracepoint* in the code. A tracepoint is a call to one of the **`ntrace(3X)`** library routines for recording the time when execution reached the tracepoint.

To add a tracepoint in a program

- In the NightView Principal Debug Window, click on the line:

  `data_process.do_work;`

- Select `Set tracepoint...` from the `Eventpoint` menu. This will open the `Set a New Tracepoint` dialog.

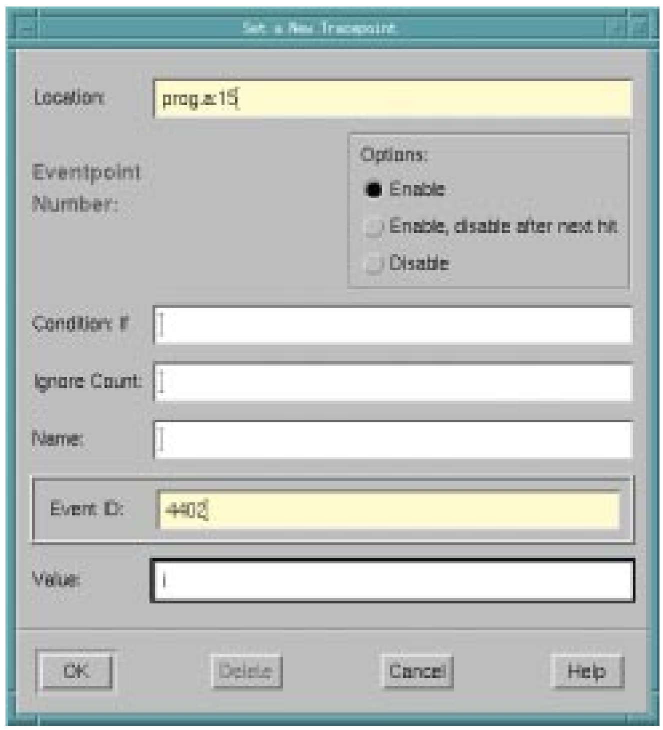

Figure 1-18. Setting a new tracepoint

- Enter the `4402` for the `Event ID`. The value 4402 is typically used as the event ID for user trace events in Ada programs. For example, the MAX-Ada utility, **`a.trace`**, expects user trace events to have an event ID of 4402.
- Enter `i` in the `Value` field. This will log the value of the variable `i` as `arg1` in the trace file every time this tracepoint is encountered.
- Press `OK`.

NOTE

You may have also entered the following command in the Command field of the NightView Principal Debug Window:

`tracepoint 4402 at` *line_number* **`value=i`**

where *line_number* coincides with the line:

```
data_process.do_work;
```

See **`tracepoint`** for details on the use of this command.

Resuming execution

Now it's time to let the program run and generate some trace data from the tracepoint we just entered.

To resume execution in NightView

- Press the Resume button in the NightView Principal Debug Window.

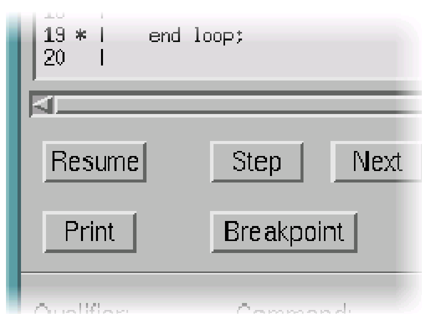

Figure 1-19. Resuming execution

Starting the simulation

Now we need to go back to our NightSim Scheduler window and start the simulation. When you click on the `Start` button, NightSim carries out the following actions:

- Attaches the timing source to the scheduler if not already attached or if the timing source has been changed
- If a real-time clock is being used as the timing source, sets the clock period in accordance with the value entered in the `Clock period` field in the Scheduler Configuration Area
- Starts the simulation with the values of the *minor cycle*, *major frame*, and *overrun* counts set to zero

To start a simulation in NightSim

- Press the `Start` button on the NightSim Scheduler window.

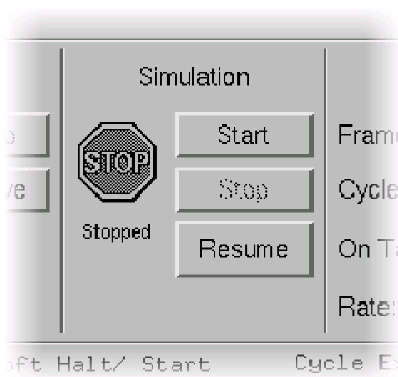

Figure 1-20. Starting the simulation

Inserting a patchpoint

NightView allows the use of *patchpoints* while debugging a process. Patchpoints are locations in the debugged process where a *patch*, usually an expression that alters the behavior of the process, is inserted.

In our example, we will insert a patchpoint in the loop to change the value of the `istat` variable in order to exit the loop:

```
while  istat = 0   loop

   data_process.do_work;
   rt_interface.FBS_wait( istat );
   i := i + 1;

end loop;
```

To insert a patchpoint in a program

- In the NightView Principal Debug Window, click on the line:

```
while  istat = 0   loop
```

- Select Set patchpoint… from the Eventpoint menu. This will open the Set a New Patchpoint dialog.

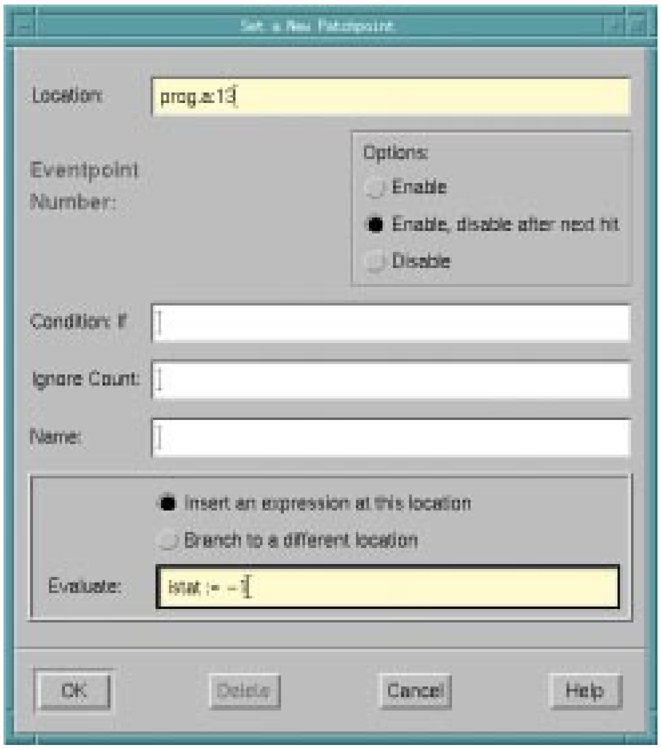

Figure 1-21. Setting a new patchpoint

- Enter the expression:

```
istat := -1
```

in the Evaluate field.

When this patchpoint is encountered during the execution of the program, the value of the variable `istat` will be set to -1, breaking out of the loop, thereby terminating the program.

- Press OK.

NOTE

You may have also entered the following command in the Command field of the NightView Principal Debug Window:

patchpoint at *line_number* **eval istat := -1**

where *line_number* coincides with the line:

```
while  istat = 0   loop
```

See **patchpoint** for details on the use of this command.

Halting user tracing and kernel tracing

Now that our program has finished, we can exit the KernelTrace utility and stop the NightTrace user daemon.

To halt kernel tracing

- On the PowerMAX OS system where you invoked the KernelTrace utility (see "To activate kernel tracing" on page 1-24), press Cntl-C.

  You should see the message:

  ```
  terminating
  ```

To halt the NightTrace user daemon

- On the PowerMAX OS system where you invoked the NightTrace user daemon (see "To activate the NightTrace user daemon" on page 1-23), enter the following command:

  ntraceud -quit *program_name***.trace.data**

  where *program_name* is the name of the program generating the trace data. So, for our example, we will issue the following command:

  ntraceud -quit prog.trace.data

Disabling the patchpoint

Before we exit NightView, we should disable the patchpoint that we set in "Inserting a patchpoint" on page 1-33. NightView retains knowledge of all eventpoints for a particular program in a current session and will reinitialize them if that program is re-run. If not disabled, the patchpoint in our program will be encountered immediately if our program is re-run under the current session of NightView, causing us to exit the loop and terminate the program.

To disable a patchpoint in NightView

- Select Summarize/Change… from the Eventpoint menu.
- Select the patchpoint from the list of eventpoints (listed with a P in the Type column).

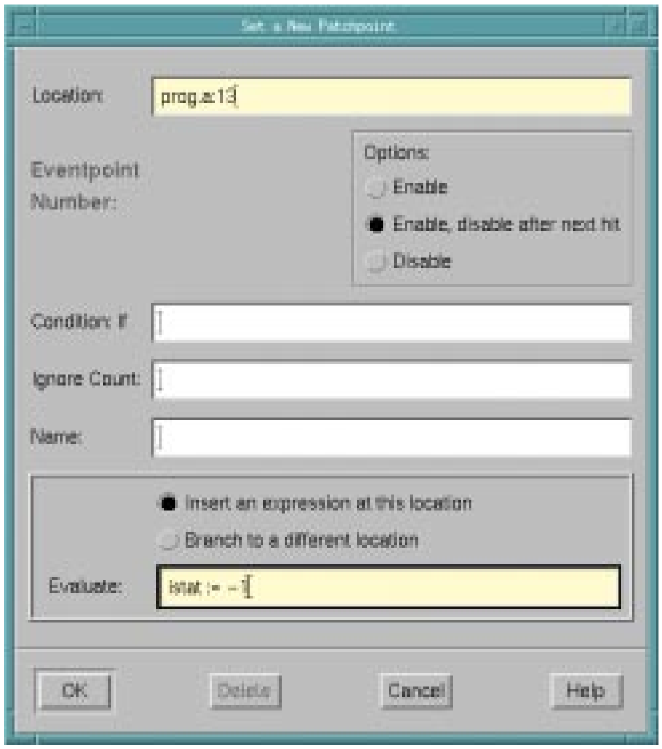

Figure 1-22. Disabling a patchpoint

- Press Disable.
- Press Close.

Exiting the program

NightView suspends the process it is debugging before it exits. We may allow the process to complete its termination by resuming its execution.

To resume execution in NightView

- Press the Resume button in the NightView Principal Debug Window.

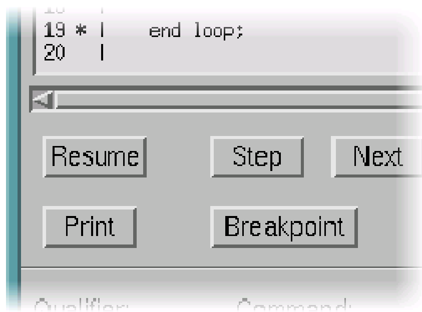

Figure 1-23. Resuming execution

Removing the scheduler

To remove the scheduler

- In the NightSim Scheduler window, press the Remove button.

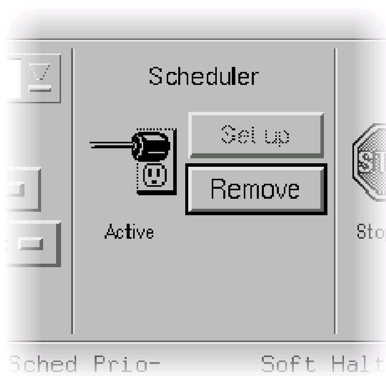

Figure 1-24. Removing the scheduler

You will be presented with the following dialog:

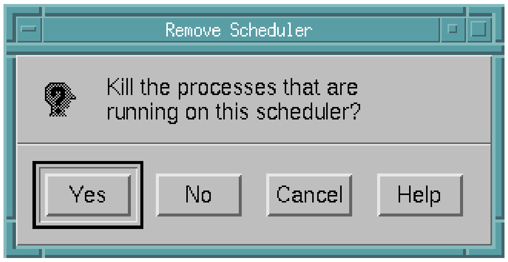

Figure 1-25. Removing the scheduler

- Press `Yes` to kill the processes that are currently scheduled on the scheduler.

Using NightTrace

NightTrace is a graphical tool for analyzing the dynamic behavior of single and multiprocessor applications. NightTrace can log application data events from simultaneous processes executing on multiple CPUs or even multiple systems. NightTrace combines application events with PowerMAX OS events and presents a synchronized view of the entire system. NightTrace allows users to zoom, search, filter, summarize, and analyze events in a wide variety of ways. PowerMAX OS events include individual system calls, context switches, machine exceptions, page faults and interrupts. Application events are defined by the user allowing logging of the data items associated with each event.

We may use NightTrace to analyze the trace data that we gathered during the execution of our program but first we will need to convert the files so that they may be used by NightTrace.

NOTE

If you do not have a system running PowerMAX OS networked to your Linux system, you will need to copy the files contained in the directory **`tutorial-sup`** from the installation CD to the working directory you created in "Getting Started" on page 1-5.

Proceed to the section titled "Invoking NightTrace" on page 1-41.

Converting kernel trace event files

To convert kernel trace event files

- On the PowerMAX OS system where you invoked the KernelTrace utility (see "To activate kernel tracing" on page 1-24), enter the following command:

  `ntfilter -v <` *raw_kernel_file* **`>`** *filtered_kernel_file*

  where *raw_kernel_file* is the file we specified using the **`-o`** option to **`ktrace`** and *filtered_kernel_file* is the name of the resultant output file from **`ntfilter`**.

  So, for our example, we will issue the following command:

  ```
  ntfilter -v < prog.ktrace.data > prog.ntrace.kernel
  ```

  The converted KernelTrace trace event file will then be saved to the file **`prog.ntrace.kernel`**. The **`-v`** option creates a **`vectors`** files that will be specified to NightTrace along with the converted KernelTrace trace event file. The **`vectors`** file is generated dynamically because it is system-configuration dependent. Without a **`vectors`** file, NightTrace will not be able to display the names of the system processes, interrupts, and exceptions that occurred during kernel tracing.

See “Converting KernelTrace Trace Event Files with ntfilter” in the *NightTrace Manual* (0890398) for more detailed information about this process.

Creating NightTrace configuration files

To create NightTrace configuration files

- On the Linux system, use the MAXAda utility, **a.trace** to create the NightTrace configuration files from the file generated by the NightTrace user daemon. The command has the following syntax:

  a.trace *program_name***.trace.data**

  where *program_name* is the name of the program that generated the trace data.

  So, for our example, we will issue the following command:

  a.trace prog.trace.data

NOTE

You may need to prepend **/usr/ada/bin** to the **a.trace** command if you did not add it to your PATH. See “Additions to PATH” on page 1-4 for more information.

This command creates the following two files:

1. *program_name***.ntrace.data**

   This file is a hard link to *program_name***.trace.data**. See for more information about this file.

2. *program_name***.ntrace.config**

   This file contains string tables, format tables, and a NightTrace display page, including descriptions of NightTrace display objects for this application’s trace events.

See Creating the NightTrace Configuration File in the *MAXAda Reference Manual* (0890516) for more detailed information about this process.

Invoking NightTrace

Now that all our files are created and converted, we may invoke NightTrace and analyze the results.

To invoke NightTrace

- On the Linux system, enter the following command

  `ntrace prog.ntrace.* vectors`

  This will start the NightTrace Analyzer and pass to it:

`prog.ntrace.*`	the files created by “Converting kernel trace event files” on page 1-39 and “Creating NightTrace configuration files” on page 1-40
`vectors`	a file created by “Converting kernel trace event files” on page 1-39 which allows NightTrace to display the names of the system processes, interrupts, and exceptions that occurred during kernel tracing.

  See ntrace Arguments for more information about invoking NightTrace.

  NightTrace will present the `NightTrace` window as well as a display page configured using the **`prog.ntrace.config`** file created in “Creating NightTrace configuration files” on page 1-40. Both windows are shown below:

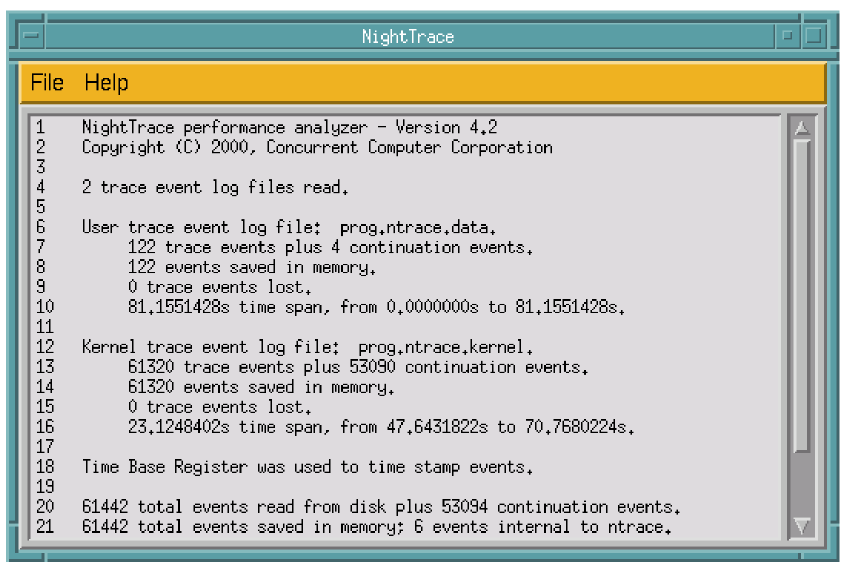

Figure 1-26. NightTrace Main window

For more information on the `NightTrace` window, see ntrace Global Window in the *NightTrace Manual* (0890398).

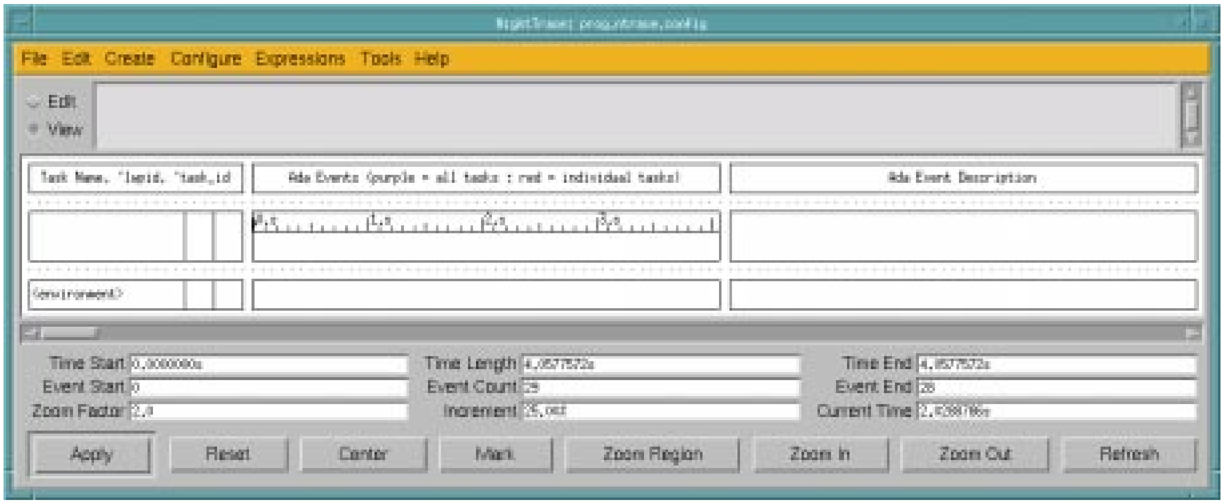

Figure 1-27. NightTrace display page

For more information on display pages, see The Display Page in the *NightTrace Manual* (0890398).

Creating a default kernel page

In order to view our kernel trace events, we need to create a default kernel page.

To create a default kernel page

- In the NightTrace window, select Default Kernel Page from the File menu.

  This will create a Default Kernel Page as shown below:

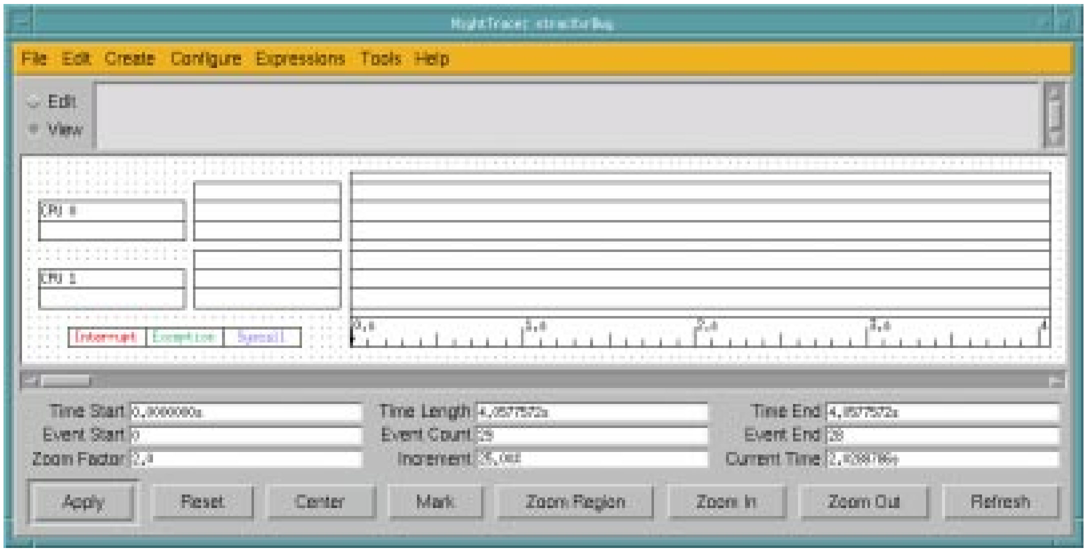

Figure 1-28. Default Kernel Page

For more information on the Default Kernel Page, see Kernel Display Pages in the *Night-Trace Manual* (0890398).

Searching for a kernel trace event

Now that we have loaded our data into NightTrace and created the appropriate display pages, we can search for the system call that corresponds to the `FBS_wait` call made in our program (see "Using NEdit" on page 1-5).

To search for a kernel trace event

- Select `Search...` from the `Tools` menu of the kernel display page (see "Creating a default kernel page" on page 1-42).

  You will be presented with the following dialog:

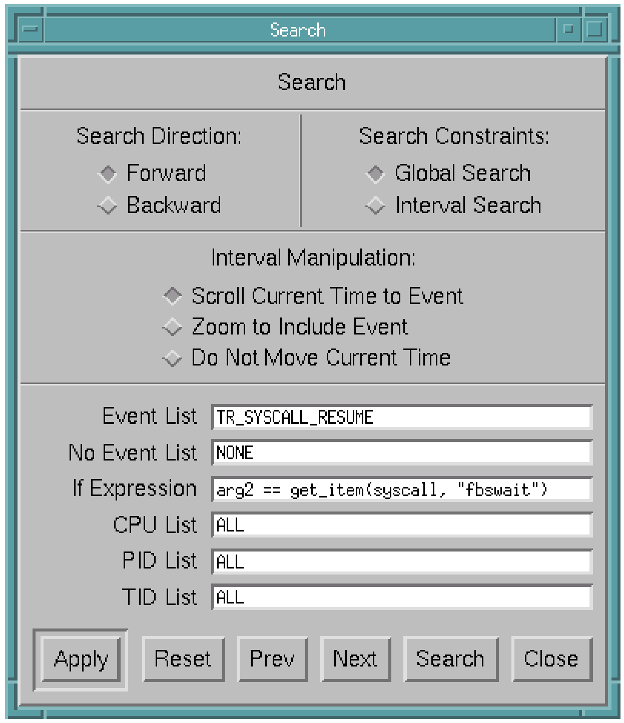

Figure 1-29. Searching for a kernel trace event

- Enter TR_SYSCALL_RESUME in the Event List field. This trace event is logged whenever a system call (syscall) is resumed (i.e., the process that caused the syscall to occur, which was switched out before the syscall could be completed, is switched back in).
- Enter arg2 == get_item(syscall, "fbswait") in the If Expression field. The `fbswait` system call corresponds to the `FBS_wait` call we made in our Ada program.
- Press Apply.
- Press Search.

NightTrace will set the current time to that of the first logged kernel trace event that matches the specified search criteria, positioning the grid on the kernel display page accordingly. This is shown in the figure below. Note the Current Time. In our example, it is set to 48.5713587 seconds.

NOTE

Since we specified the **`-bufferwrap`** option to **`ktrace`** (see "To activate kernel tracing" on page 1-24), it is likely that the earlier trace events may have been overwritten by buffer wraparound during the execution of the program. Hence, we may not actually see the *first* actual kernel trace event that corresponds to our search criteria. However, this is sufficient for our example.

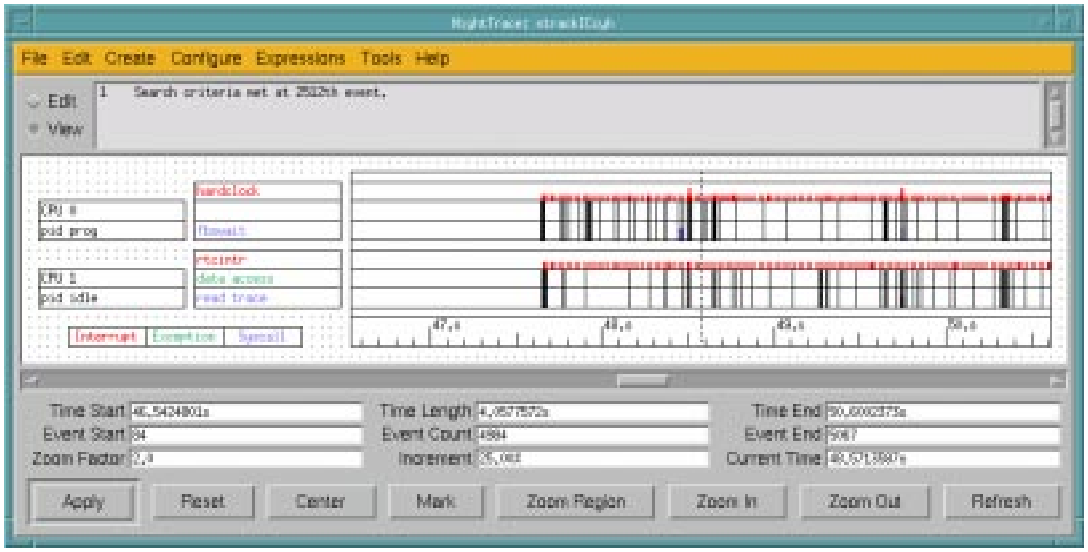

Figure 1-30. First kernel trace event

In addition to setting the current time and repositioning the grid on the kernel display page when the search for the kernel trace event was performed, NightTrace will automatically set the current time and reposition the display page that contains the user trace events as well. This is shown in the figure below.

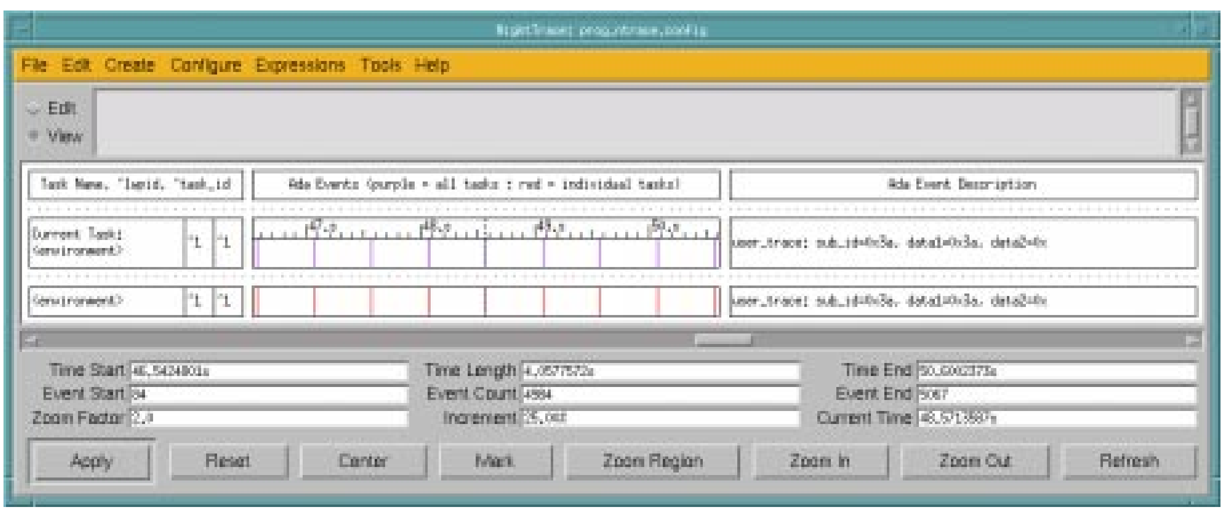

Figure 1-31. NightTrace display page repositioned accordingly

Searching for a user trace event

Now that we have found the first logged kernel trace event, we can search for the user trace events that we logged using NightView (see “Adding a tracepoint in the program” on page 1-30).

To search for a user trace event

NOTE

You may use the same search dialog that you used in the previous step, “Searching for a kernel trace event” on page 1-43.

- Select Search… from the Tools menu of the display page created when NightTrace was invoked (see “Invoking NightTrace” on page 1-41).

  You will be presented with the following dialog:

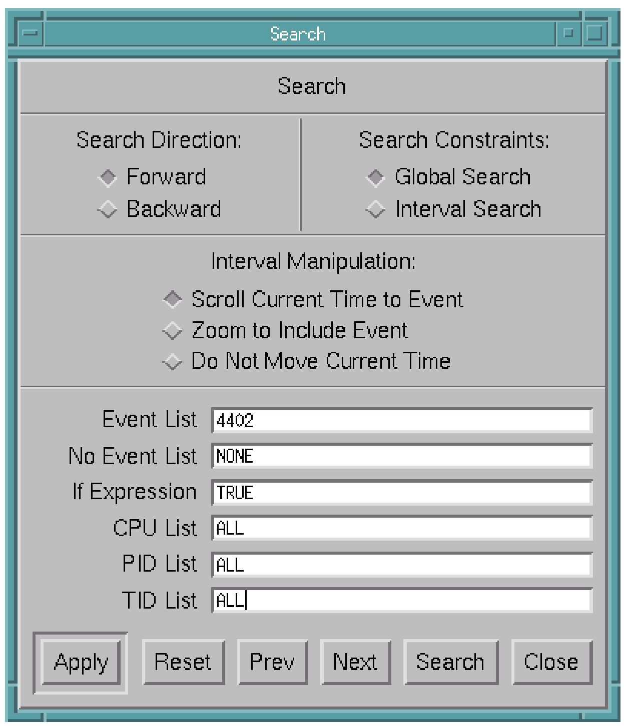

Figure 1-32. Searching for a user trace event

- Enter 4402 in the Event List field. This corresponds to the Event ID for the tracepoint we specified in NightView (see “Adding a tracepoint in the program” on page 1-30).
- Ensure that the value of the If Expression field is TRUE.

- Press Apply.
- Press Search.

NightTrace will set the current time to the first user trace event after the current time that matches the specified search criteria, positioning the grid on the kernel display page accordingly. This is shown in the figure below. Note the Current Time now. In our example, it is set to 48.5714136 seconds, 0.0000549 seconds after the `fbswait` system call we found in "Searching for a kernel trace event" on page 1-43.

You can alternately search between the kernel display page (see "To search for a kernel trace event" on page 1-43) and the display page which contains the user trace events (see "To search for a user trace event" on page 1-46) to see that an `fbswait` system call always precedes the user trace event that we logged, which is what we would expect.

NOTE

> If you used the same search dialog as you used for searching for a kernel trace event, you may use the Prev button on the search dialog for the previous search criteria. You can alternate between searching for user trace events and kernel trace events using this functionality.

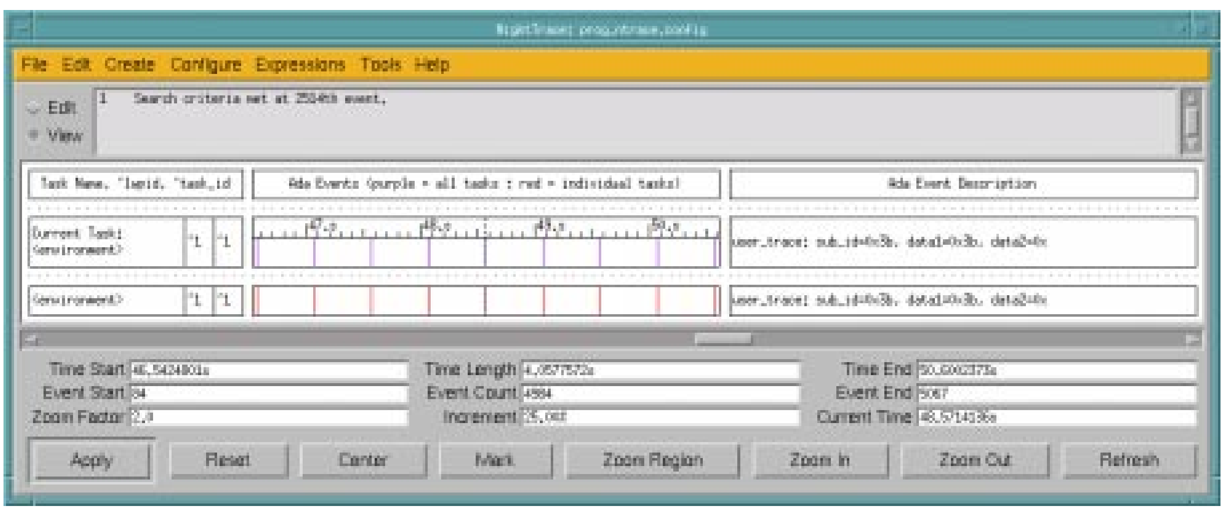

Figure 1-33. NightTrace display page

Zooming in

To zoom in:

- You may use the Zoom In button on the NightTrace Analyzer to see more details.

For our example, we zoomed in on our kernel display page 13 times to see the following level of detail.

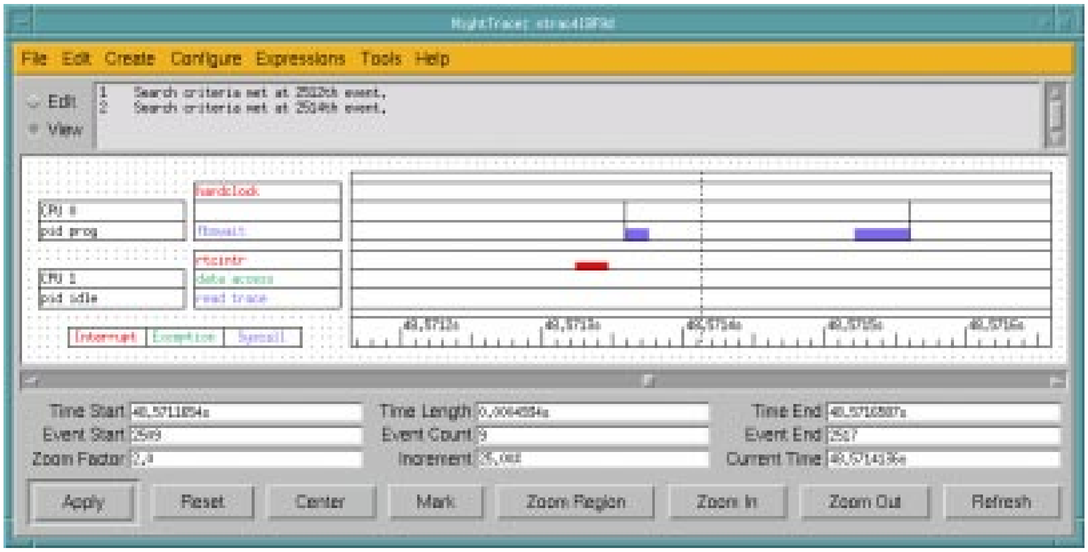

Figure 1-34. Zoomed in kernel display page

In the above figure, the first bar (red) indicates the real-time clock interrupt for this cycle. The second bar (blue) shows the target process **`prog`** exiting the `FBS_wait` call in the Ada code. The current time line is positioned at the user trace event that we previously searched for.

Looking at the other display page (which shows our user trace events), we can see the user_trace event inserted through NightView (see "Adding a tracepoint in the program" on page 1-30). Note that both displays are synchronized in time (the current time line represents the same instant in time on both display pages). You may middle-click on the line representing the user trace event to see more detailed information.

The third and final bar (blue) on the kernel display page represents the next encounter of the `FBS_wait` call in the loop.

NOTE

Due to a problem with the **`-bufferwrap`** option to the **`ktrace`** command, user and kernel data may not appear synchronized. This problem has been fixed in the **`ktrace`** and **`ntfilter`** commands in PowerMAX OS 4.3 Patch Set 6 (**`trace-004`** and **`base-006`**). See "To activate kernel tracing" on page 1-24 for more information.

Conclusion

This concludes our tutorial for the PowerWorks Linux Development Environment. We hope that we have given you a sufficient overview of the various tools and the interactions between them.

Spine for 1/2” 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

PowerWorks Linux Development Environment

Tutorial

0898100