

NightStar RT Tutorial

Version 4.2

(RedHawkTM Linux[®])



0898009-073 March 2010 Copyright 2010 by Concurrent Computer Corporation. All rights reserved. This publication or any part thereof is intended for use with Concurrent products by Concurrent personnel, customers, and end-users. It may not be reproduced in any form without the written permission of the publisher.

The information contained in this document is believed to be correct at the time of publication. It is subject to change without notice. Concurrent Computer Corporation makes no warranties, expressed or implied, concerning the information contained in this document.

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.

Concurrent Computer Corporation and its logo are registered trademarks of Concurrent Computer Corporation. All other Concurrent product names are trademarks of Concurrent while all other product names are trademarks or registered trademarks of their respective owners.

Linux[®] is used pursuant to a sublicense from the Linux Mark Institute.

NightStar's integrated help system is based on Qt's Assistant from Trolltech.

Preface

General Information

NightStar RTTM allows users running RedHawk to schedule, monitor, debug and analyze the run-time behavior of their time-critical applications as well as the operating system kernel.

NightStar RT consists of the NightTraceTM event analyzer; the NightProbeTM data monitoring tool, the NightViewTM symbolic debugger, the NightSimTM scheduler, the Night-TuneTM system and application tuner, the Data Monitoring API, and the Shmdefine shared memory utility.

Scope of Manual

This manual is a tutorial for NightStar RT.

Structure of Manual

This manual consists of seven chapters and an appendix which comprise the tutorial for NightStar RT.

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.

emphasis

Words or phrases that require extra emphasis use emphasis 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:

0898395	NightView TM User's Guide
0898398	NightTrace [™] User's Guide
0898465	NightProbe [™] User's Guide
0898480	NightSim [™] User's Guide
0898515	NightTune TM User's Guide

Contents

Chapter 1 Overview

Getting Started	2
Setting Up User Privileges 1-2	2
Creating a Tutorial Directory 1-3	3
Building the Program 1-4	1

Chapter 2 Panels

Moving Panels	2-2
Tabbed Panels	2-6
Context Menus	2-8
Tutorial Screen Shots	2-9

Chapter 3 Using NightView

Invoking NightView
Heap Debugging 3-5
Activating Heap Debugging 3-5
Controlling the app Program 3-7
Scenario 1: Use of a Freed Pointer
Scenario 2: Freeing an Invalid Pointer Value
Scenario 3: Writing Past the End of an Allocated Block 3-14
Scenario 4: Use of Uninitialized Heap Blocks
Scenario 5: Detection of Leaks
Scenario 6: Allocation Reports
Disabling Heap Debugging 3-21
Debugging Multiple Threads 3-22
Using Monitorpoints
Using Eventpoint Conditions and Ignore Counts 3-28
Using Patchpoints 3-29
Adding and Replacing Functions Dynamically
Using Tracepoints
Conclusion - NightView

Chapter 4 Using NightTrace

Invoking NightTrace
Configuring a User Daemon 4-3
Streaming Live Data to the NightTrace GUI
Using NightTrace Timelines
Zooming
Moving The Interval 4-8
Using the Events Panel for Textual Analysis
Customizing Event Descriptions 4-10

Searching the Events List.	4-12
Halting the Daemon	4-16
Using States	
Displaying State Duration	
Generating Summary Information	4-23
Defining a Data Graph	4-27
Kernel Tracing	4-32
Obtaining Kernel Trace Data	4-32
Using Prerecorded Kernel Data	4-34
Analyzing Kernel Data	
Mixing Kernel and User Data	4-38
Using the NightTrace Analysis API.	
Automatically Tracing Your Application	4-44
nlight Wizard - Selecting Programs	4-45
nlight Wizard - Defining Illuminators	4-47
nlight Wizard - Selecting Illuminators	4-49
nlight Wizard - Relinking the Program	4-50
nlight Wizard - Activating Illuminators	4-52
nlight Wizard - Running the Program	4-53
Analyzing Application Illumination Events	
Summarizing Workload Performance	4-62
Batch Summary of Functions	4-68
Shutting Down	4-69
Conclusion - NightTrace	4-69

Chapter 5 Using NightProbe

Invoking NightProbe
Selecting Processes
Viewing Live Data
Modifying Variables
Selecting Variables for Recording and Alternative Viewing
Selection of Views
Table View 5-8
Graph View
Sending Probed Data to Other Programs
Using Datamon to Modify Program Variables
Conclusion - NightProbe

Chapter 6 Using NightTune

Invoking NightTune
Monitoring a Process
Tracing System Calls
Process Details
Process Details - Memory Details
Process Details - File Descriptors
Process Details - Signals 6-9
Changing Process Scheduling Parameters
Setting Process CPU Affinity
Setting Interrupt CPU Affinity
Shielding CPUs for Maximum Determinism and Performance
Conclusion - NightTune

Chapter 7 Using NightSim

Creating FBS Applications	1
Invoking NightSim	2
Creating a Scheduler	3
Running the Scheduler	6
Using Datamon to Modify Program Variables	8
Overrun Detection and System Tuning	9
Shutting Down the Scheduler	4

Appendix A Tutorial Files

api.c	 	 	A-2
app.c	 	 	A-5
function.c	 	 	A-9
report.c	 	 	A-9
set_workload.c	 	 	A-10
set_rate.c	 	 	A-10

Illustrations

Figure 2-1. Viewing Page with List & Graph Panels	
Figure 2-2. Panel Detaches from Page	
Figure 2-3. Panel Movement in Progress	
Figure 2-4. Graph Panel on Top of List Panel	
Figure 2-5. Table View added to Page	
Figure 2-6. Panel in Motion Creating Tab	
Figure 3-1. NightView Main Window	
Figure 3-2. app Program Loaded	
Figure 3-3. NightView Debug Heap Dialog	
Figure 3-4. Heap Totals and Configuration	
Figure 3-5. info memory Command Output	
Figure 3-6. Heap Error Description	
Figure 3-7. Heap Leaks Display	
Figure 3-8. Still Allocated Blocks Display	
Figure 3-9. Context Panel With Stack Frames Expanded	
Figure 3-10. Monitorpoint Dialog	
Figure 3-11. NightView Monitor Panel	
Figure 3-12. Patchpoint Dialog	
Figure 3-13. Result of Patching in Call to Newly Loaded Function	on
Figure 3-14. Tracepoint Dialog	
Figure 4-1. NightTrace main window	
Figure 4-2. Import Daemon Definitions Dialog	
Figure 4-3. Logging Data	
Figure 4-4. app_data Page	
Figure 4-5. NightTrace Timeline	
Figure 4-6. Timeline Interval Panel	
Figure 4-7. Events Panel	
Figure 4-8. Add Event Description dialog	
Figure 4-9. Searching using the Profiles panel	
Figure 4-10. Browse Events Dialog	

Figure 4-11. Events Panel After Search	4-15
Figure 4-12. Timeline Panel w/ Tool Tip	
Figure 4-13. Profiles Panel With Obtuse Profile Selected	
Figure 4-14. Timeline Editing	4-19
Figure 4-15. Edit State Graph Profile dialog	
Figure 4-16. Sine State in Timeline	
Figure 4-17. Summary Results Page	4-24
Figure 4-18. Summary Graph	4-25
Figure 4-19. Data Graph Profile Dialog	4-26
Figure 4-20. Modified Data Graph	4-27
Figure 4-21. Resizing in Progress	4-28
Figure 4-22. Adding a Data Graph	4-29
Figure 4-23. Edit Data Graph Profile Dialog	4-30
Figure 4-24. Display Page with Data Graph	4-31
Figure 4-25. Edit Daemon Definition Dialog	4-33
Figure 4-26. Kernel Display Page	4-35
Figure 4-27. System Call Resume for Nanosleep	4-37
Figure 4-28. Events Panel after Search	4-38
Figure 4-29. Longest Instance of State	4-40
Figure 4-30. Export Profiles to NightTrace API Source File dialog	4-42
Figure 4-31. nlight Wizard - Select Programs Step	4-45
Figure 4-32. nlight Wizard - Define Illuminators Step	4-47
Figure 4-33. nlight Wizard - Select Illuminators Step	4-49
Figure 4-34. nlight Wizard - Relink Programs Step	4-50
Figure 4-35. nlight Wizard - Activate Illuminators Step	4-52
Figure 4-36. nlight Wizard - Run Scripts Step (with Stream Mode selected). 4-54
Figure 4-37. NightTrace - Import File Name	4-55
Figure 4-38. NightTrace - Daemon Ready to Launch	
Figure 4-39. NightTrace - Daemon Collection Events	
Figure 4-40. NightTrace - /tmp/data_import Tab	
Figure 4-41. NightTrace - Customized Events Panel	
Figure 4-42. NightTrace - Search Events for Text	
Figure 4-43. NightTrace - Events Panel with Selected Events	
Figure 4-44. NightTrace - Descriptive Tool Tip	
Figure 4-45. NightTrace - Launches Editor with Source File at Line Numb	
Figure 4-46. NightTrace - Profile Definition Panel - State Mode	
Figure 4-47. NightTrace - Select Events Dialog with Search Active	
Figure 4-48. NightTrace - Profile Definition Panel with State Defined	
Figure 4-49. NightTrace - Tab with Results of Summary	
Figure 4-50. NightTrace - Summary Results Sorted By Duration	
Figure 4-51. NightTrace - Description of Longest Instance of work Function	
Figure 5-1. NightProbe Main Window	
Figure 5-2. Program Selection Dialog	
Figure 5-3. Process Selection Dialog	
Figure 5-4. NightProbe Browse Panel	
Figure 5-5. Expanded Data Item	
Figure 5-6. Variable Modification in Progress	
Figure 5-7. Mark and Record Attributes Set	
Figure 5-8. Table View Figure 5-9. Item Selection Dialog	···· 5-9
Figure 5-9. Item Selection Dialog	J-10 5 11
Figure 5-11. Graph Panel Figure 5-12. Graph Panel Actively Displaying Values	
Figure 5-12. Graph Faner Activery Displaying values	
rigure 5-15. Luit Curve Attributes Dialog	5-14

Figure 5-14. Graph Panel with Modified Curves
Figure 5-15. Recording area of Configuration Page
Figure 5-16. Clock Selection Dialog
Figure 5-17. Record To Program Dialog
Figure 5-18. Recording Area of Configuration Page w/ Destination
Figure 5-19. Example Output of Graph Program
Figure 6-1. NightTune initial panels
Figure 6-2. Expanded Process List
Figure 6-3. Process List with Threads
Figure 6-4. Strace Output of Thread
Figure 6-5. Process Details Dialog
Figure 6-6. Process Memory Details Page
Figure 6-7. File Descriptors Page
Figure 6-8. Signals Page
Figure 6-9. Process Scheduler Dialog
Figure 6-10. NightTune Process List with modified thread
Figure 6-11. CPU Shielding and Binding Panel
Figure 6-12. CPU Shielding and Binding Panel with Bound Thread
Figure 6-13. NightTune with Interrupt Activity Panel
Figure 6-14. Interrupt Affinity Dialog
Figure 7-1. NightSim Initial Window
Figure 7-2. NightSim Edit Process Dialog
Figure 7-3. Runtime Properties Tab
Figure 7-4. Scheduling Started7-6
Figure 7-5. NightSim Monitor Page7-7
Figure 7-6. NightTune with Interrupt and CPU Shielding & Binding Panels7-10
Figure 7-7. Process and Interrupt Bound to CPU 0
Figure 7-8. Change Shielding Dialog
Figure 7-9. Shielding Changes Pending

NightStar RT Tutorial

NightStar RTTM is an integrated set of debugging tools for developing time-critical Linux[®] applications. NightStar RT are designed to be minimally intrusive, preserving the execution behavior and determinism of your applications. Users can quickly and easily debug, monitor, analyze, and tune their applications.

The NightStar RT tools consist of:

- NightViewTM source-level debugger
- NightTraceTM event analyzer
- NightProbeTM data monitor
- NightTuneTM system and application tuner
- NightSimTM scheduler

In this tutorial, we will integrate these tools into one cohesive example incorporating various scenarios which demonstrate their extensive functionality.

Getting Started

Certain activities in this tutorial require enhanced user privileges which are not granted to user accounts by default. You will need to run as the root user, where indicated within this tutorial, or obtain appropriate privileges as detailed in the "Setting Up User Privileges" on page 1-2.

Setting Up User Privileges

Linux provides a means to grant otherwise unprivileged users the authority to perform certain privileged operations. **pam_capability(8)**, the Pluggable Authentication Module, is used to manage sets of capabilities, called roles, required for various activities.

Linux systems should be configured with a nightstar role which provides the capabilities required by NightStar RT. In order to take full advantages of NightStar RT features, each user must be configured to use (at a minimum) the capabilities specified below.

Edit **/etc/security/capability.conf** and define the nightstar role (if it is not already defined) in the "ROLES" section:

```
role nightstar cap_sys_nice cap_ipc_lock
```

Additionally, for each NightStar RT user on the target system, add the following line at the end of the file:

user username nightstar

where username is the login name of the user.

If the user requires capabilities not defined in the nightstar role, add a new role which contains nightstar and the additional capabilities needed, and substitute the new role name for nightstar in the text above.

In addition to registering your login name in /etc/security/capability.conf, files under the /etc/pam.d directory must also be configured to allow capabilities to be activated.

To activate capabilities, add the following line to the end of selected files in /etc/pam.d if it is not already present:

session required pam_capability.so

The list of files to modify is dependent on the list of methods that will be used to access the system. The following table presents a recommended configuration that will grant capabilities to users of the services most commonly employed in accessing a system.

/etc/pam.d File	Affected Services	Comment
remote	telnet rlogin rsh (when used <u>w/o</u> a command)	Depending on your system, the remote file may not exist. Do not create the remote file, but edit it if it is present.
login	local login (e.g. console) telnet* rlogin* rsh* (when used <u>w/o</u> a command)	*On some versions of Linux, the presence of the remote file limits the scope of the login file to local logins. In such cases, the other services listed here with login are then affected solely by the remote configuration file.
rsh	rsh (when used <u>with</u> a command)	e.g. rsh system_name a.out
sshd	ssh	You must also edit /etc/ssh/sshd_config and ensure that the following line is present: UsePrivilegeSeparation no
gdm	gnome sessions	
kde	kde sessions	

Table 1-1. Recommended /etc/pam.d Configuration

If you modify /etc/pam.d/sshd or /etc/ssh/sshd_config, you must restart the sshd service for the changes to take effect:

service sshd restart

In order for the above changes to take effect, the user must log off and log back onto the target system.

NOTE

To verify that you have been granted capabilities, issue the following command:

/usr/sbin/getpcaps \$\$

The output from that command will list the roles currently assigned to you.

Creating a Tutorial Directory

We will start by creating a directory in which we will do all our work. Create a directory and position yourself in it:

- Use the **mkdir(1)** command to create a 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

Source files, as well as configuration files for the various tools, are copied to /usr/lib/NightStar/tutorial during the installation of NightStar RT. We will copy these tutorial-related files to our tutorial directory.

- Copy all tutorial-related files to our local directory.

```
cp /usr/lib/NightStar/tutorial/* .
```

Building the Program

Our example uses a cyclic multi-threaded program which performs various tasks during each cycle. The cycle will be controlled by the main thread which uses a timeout with a configurable rate.

A portion of the main source file, **app.c**, is shown below:

```
int
main (int argc, char * argv[])
{
  pthread_t thread;
  pthread_attr_t attr;
  struct sembuf trigger = \{0, 2, 0\};
  nosighup();
   trace_begin ("/tmp/data",NULL);
   sema = semget (IPC_PRIVATE, 1, IPC_CREAT+0666);
  pthread_attr_init(&attr);
  pthread_create (&thread, &attr, sine_thread, &data[0]);
  pthread_attr_init(&attr);
  pthread_create (&thread, &attr, cosine_thread, &data[1]);
  pthread_attr_init(&attr);
  pthread_create (&thread, &attr, heap_thread, NULL);
   for (;;) {
     struct timespec delay = { 0, rate } ;
     nanosleep(&delay,NULL);
     work(random() % 1000);
      if (state != hold) semop(sema,&trigger,1);
   }
   trace_end () ;
}
```

The program creates three threads and then enters a loop which cyclically activates each of two threads based on a common timeout. The third thread, heap_thread, runs asynchronously.

To build the executable

From the local **tutorial** directory, enter the following command:

cc -g -o app app.c -lntrace_thr -lpthread -lm

NOTE

The NightStar RT tools require that the user application is built with DWARF debugging information in order to read symbol table information from user application program files. For this reason, the **-g** compile option is specified. However, the tools can be used to debug programs without symbols with reduced functionality.

NightStar RT Tutorial

NightStar provides flexibility in configuring the graphical user interface to suit your needs through the use of resizable and movable panels.

This chapter presents the concepts involved in moving and resizing panels. It is designed merely for reference, not as a step-by-step instructional guide.

Please read this chapter before proceeding to the first steps in using the tools, which follows in "Using NightView" on page 3-1.

Moving Panels

Consider the following NightProbe page which contains a List view and a Graph view each in their own panel:

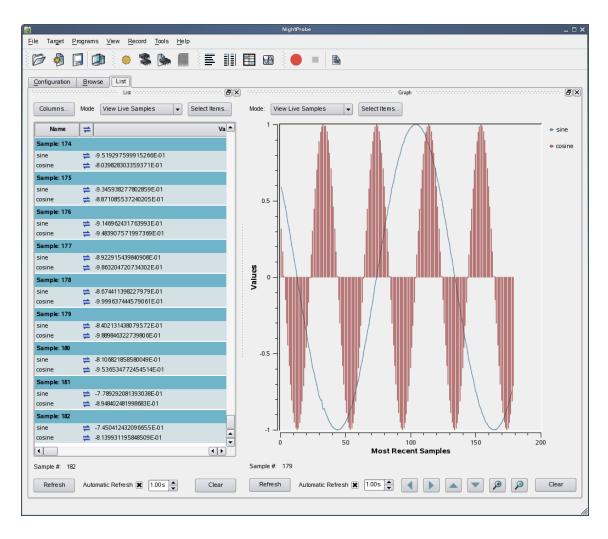


Figure 2-1. Viewing Page with List & Graph Panels

Panels are moved by left-clicking the title bar, dragging them to a new location, and then releasing the mouse button. Depending on the location of the panel when the mouse button is released, the panel will either remain detached or will be inserted into the page again.

To detach the panel from the page without inserting it, click the left-most control box in the upper right-hand corner of the panel.

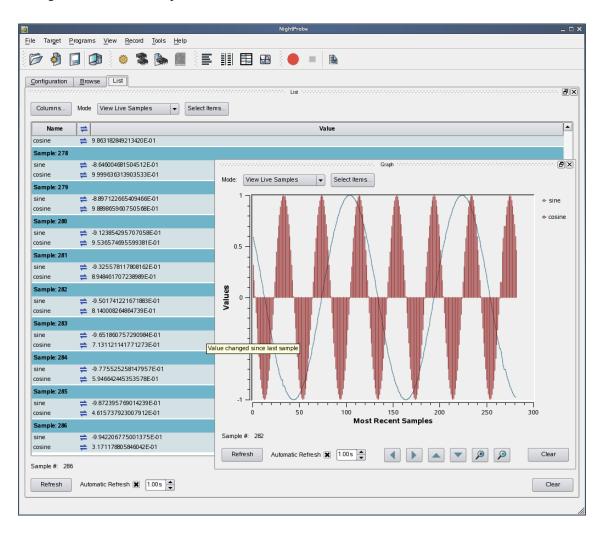


Figure 2-2. Panel Detaches from Page

The Graph panel detaches from the page and becomes free floating.

If moved outside the boundaries of the main window and released, the panel will remain detached from the main window. However, even in detached mode, if the main window is iconified, the detached panel will be iconified with it. For this reason, detached panels are not very useful in and of themselves. Detaching is most often useful as part of moving a panel and re-docking it.

To insert a panel into the page at a new location, drag the panel using the left mouse button on its title bar and move it until it approaches a boundary of the page. The window will respond by creating space indicating where the panel will be inserted.

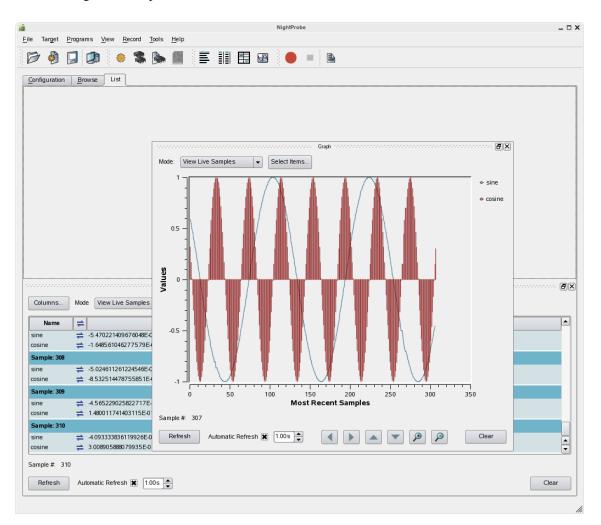
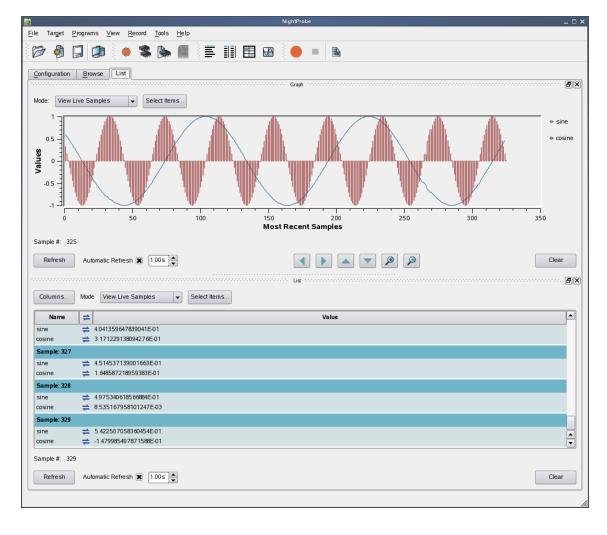


Figure 2-3. Panel Movement in Progress

The figure above shows space being created above the List panel as the Graph panel is dragged towards the upper horizontal boundary of the page.



At this point, releasing the mouse button will cause the **Graph** panel to be inserted into the page, consuming the recently created space.

Figure 2-4. Graph Panel on Top of List Panel

IMPORTANT

When attempting to move panels inside of a page, if an empty space does not appear where you desire it, try increasing the size of the main window, decreasing the size of the undocked panel, and moving an alternative edge of the undocked panel near where you want to place it. By default, the tools usually add panels to the right-hand side of the page when a new panel is created.

In the following figure, a Table panel has been added to the right-hand side of the Graph and List panels.

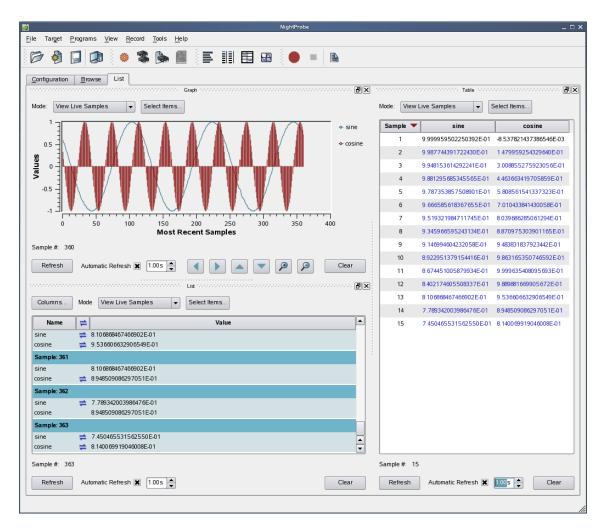


Figure 2-5. Table View added to Page

Panels can be resized by left-clicking on the separator between the panels and dragging it to the desired size.

Tabbed Panels

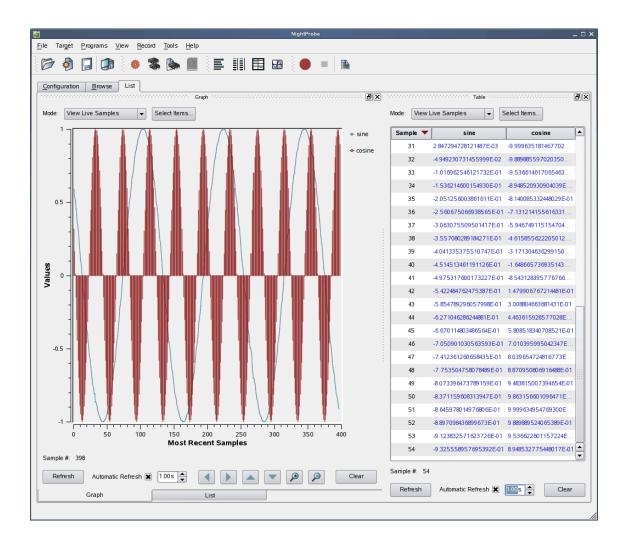
Another feature of the graphical user interface is the use of tabbed panels. Tabbed panels allow you to maximize your GUI real estate by placing two or more panels in the same location by stacking them on top of each other. You can then raise a panel to the top by clicking on its tab.

File Target <u>P</u>rograms <u>V</u>iew <u>R</u>ecord <u>T</u>ools <u>H</u>elp E III 🖽 📾 B A THE - L * Configuration Browse List ð× Tabl Ð× Select Items.. Columns... Mode View Live Samples Mode: View Live Samples ▼ Select Items... ≓ Value Sample sine cosine Nam sine = 2 615680580456744E-01 11 8.674451005879934E-01 9.999635408095693E cosine -7.010414918260884E-01 8 402174605508337E-01 9 889881669905672E-01 12 Sample: 375 13 8.106868467466902E-01 9.536606632906549E sine 2.106957496094004E-01 7.789342003986476E-01 8.948509086297051E-01 14 -8039670504967339F-01 ð× Graph 15 7.450465531562550E-01 8.140069919046008E-01 7.091167885482276E-01 7.131195552947748.. Mode: View Live Samples Select Items 16 17 6.712433873955804E-01 5.946727781278225. sine 6.315301578829990E-01 4.615832082430596E-01 18 cosine 0.5 19 5.900859510283019E-01 3.171279470253189E Values 5.470243623298700E-01 1.648639564288165E-01 20 0 5.024634204098081E-01 8.540474916611725E. 21 22 4.565252635062418E-01 -1.479933010777271. -0 5 23 4.093358047014619E-01 -3.008829969812837E. 24 3.610243868035007E-01 -4.463639674157159... 350 50 100 300 200 250 25 3.117234278270850E-01 -5.808539941043371E Most Recent Samples 2.615680580456744E-01 -7.010414918260884E-01 26 Sample #: 379 27 2 106957496094004E-01 -8039670504967339 Automatic Refresh 🕱 1.00s 🔺 🔻 🔎 🔎 Refresh Clear 1.592459397440948E-01 -8.870963055440059. 28 29 1.073596485641914E-01 -9.483823422692438E-01 51110 JZ J401217 JZL-0 -9.536614617065463E-01 cosine 30 5.517909254704709E-02 -9.863160975956257. Sample: 382 31 2.847294728121487E-03 -9 999635181467702 -1.536214600154930E-01 sine 32 -4 949230731455999E-02 -9 889885597020350 --8.948520930904039E-01 cosine 33 -1.016962546121732E-01 -9.536614617065463 •• • 34 -1.536214600154930E-01 -8.948520930904039E. Sample #: 382 Sample #: 34 Refresh Automatic Refresh 🕱 1.00 s Clear Automatic Refresh 🕱 🚺 🚺 Refresh Clear List

To create a tabbed panel, move a panel to the lower horizontal edge of another panel until a tab appears at the bottom of the panel still connected to the page.

Figure 2-6. Panel in Motion Creating Tab

In the figure above, the Graph panel is being dragged from its original position on top of the List panel towards the bottom of the List panel. A tab appears on the List panel indicating that if the mouse button is released, the Graph and List panels will be tabbed and therefore consume the same area of the page.



IMPORTANT

To move a panel above another panel, move the desired panel to the top boundary of the other panel. If you move a panel to the bottom boundary of another panel, it will become a tabbed panel instead.

Context Menus

The NightStar tools rely heavily on use of context menus.

Context menus are menus that appear when you use the mouse to right-click when the mouse cursor is positioned over an area or item of interest. They are called context menus because their content is often dependent on the context of the area in which you right-click, or the item which you right-click upon.

When in doubt, try a right-click operation and see if a menu becomes available.

Tutorial Screen Shots

In order to show full screen shots in this tutorial, the size of each main window has often been left to its default setting. Displaying larger windows would require compression in order to fit the image within the available space of a printed page; such compression obscures detail.

However, as a user of the tutorial, increasing the size of the main window is highly recommended so you can see more data without having to scroll the contents of individual panels.

In many cases within this tutorial, portions of expanded areas of the screen have been extracted from the main window and are included as stand-alone screen shots. These correspond to panels within the main window of each tool.

NightStar RT Tutorial

3 Using NightView

NightView is a graphical source-level debugging and monitoring tool specifically designed for time-critical applications. NightView can monitor, debug, and patch multiple processes running on multiple processors with minimal intrusion.

NightView supports all the features you find in standard debuggers, including:

- breakpoints
- single stepping through statements
- single stepping over function calls
- full symbolic expression analysis
- conditions and ignore counts for breakpoints
- · hardware-assisted address traps (watchpoints)
- assembly and symbolic debugging

In addition to standard debugging capabilities, NightView provides the following features:

- application-speed eventpoint conditions
- the ability to patch code to change program flow or modify memory or registers during program execution
- hot patch and eventpoint control
- synchronous data monitoring
- loadable modules
- support of multi-threaded programs
- debugging of multiple processes
- dynamic memory debugging

Invoking NightView

- Execute NightView by issuing the following command:

nview &

at the command prompt or by double-clicking on the desktop icon.

NOTE

If you do not have desktop icons for the NightStar tools, run /usr/lib/NightStar/bin/install_icons.

When we launch NightView, the NightView main window is presented.

🚯 NightView Debugger	r (on raptor)
<u>File Vi</u> ew She <u>ll</u> <u>P</u> rocess <u>S</u> ource <u>E</u> ventpoint <u>D</u> ata <u>T</u> ools <u>H</u> elp	
	🗉 💷 🗿 看 👓 🎘 🔚 🌴 📗 🎬
Source	Eventpoints
	Type ID 🔺 Location PID Enabled Ignore Hits Crossings Cmds Condition
	•
	Item Processes for shell local
	Context Locals
	Messages
	NightView 7.1.1(nview-7.1.1-1.1), Linked Mon Jan 5 17:26:39 EST
	2009
	In case of confusion, type "help"
	/usr/lib/NightView-7.1.1/ReadyToDebug ~jeffh/.bash_init complete.
	raptor> /usr/lib/NightView-7.1.1/ReadyToDebug
Command:	•
No process	

Figure 3-1. NightView Main Window

NOTE

If this is the first time you've invoked NightView since installing NightStar or upgrading to the latest version, you may see a welcome screen. If so, press the NightView button to proceed.

In our example, we'll be debugging a single application.

NOTE

If you have not yet created the **app** program, see "Building the Program" on page 1-4.

- Invoke our tutorial application in the NightView main window by selecting Run... from the Process menu and entering:

./app

in the text field of the Run on local dialog.

- Press OK to close the dialog and run the program.

Any output generated by the program will appear in the Messages panel.

When the **app** program begins to execute, NightView displays the source in the source panel and stops the program at the first line of code.

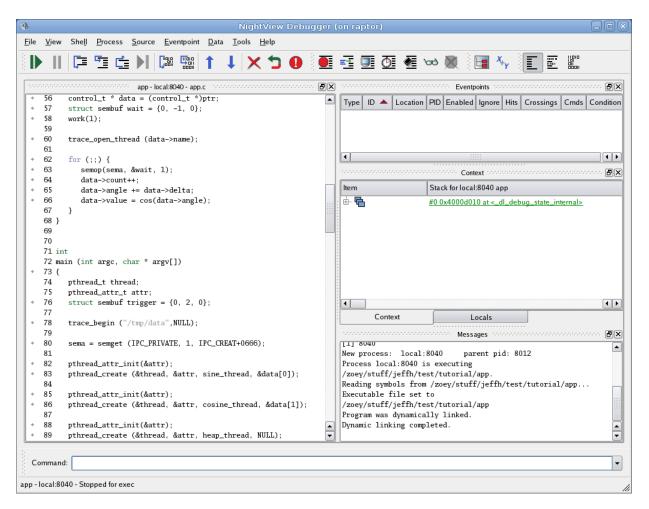


Figure 3-2. app Program Loaded

IMPORTANT

Do not resume execution of the program at this time.

NightView supports debugging multiple processes as well as single and multi-threaded processes. In this tutorial, you will be debugging a single process.

Heap Debugging

Debugging dynamic memory problems can be difficult and extremely time-consuming. The word *heap* refers to a collection of allocated and freed memory typically controlled by the malloc() and free() utilities in the C language.

NightView provides the unique ability to monitor and detect memory allocations, frees, and sets of user errors without requiring a non-standard allocator to be compiled or linked into your program.

One advantage of this is that often when you switch to a debugging allocator, the way blocks are allocated and freed changes -- often hiding the very bugs you're trying to find.

NightView offers a variety of settings and debugging levels that are useful in catching common heap-related errors. Some settings will change the behavior of the system allocator, affecting the size of allocated blocks and, ultimately, the address values returned.

Dynamic memory errors are detected in one of four ways:

- a check of the entire heap at a specified frequency in terms of the number of heap functions (e.g., malloc, free, calloc, etc.) called
- a check of an individual allocated block when free or realloc is called
- a check of the entire heap when a **heappoint** is crossed
- a check of the entire heap when a heapcheck command is issued

The frequency setting of the **heapdebug** command or **Debug** Heap window controls how often NightView should check for heap errors when a utility routine is called. Setting the frequency to 1 causes NightView to check for heap errors on every heap operation.

A **heappoint** causes NightView to check for errors when the process executes instructions where the heappoint is inserted. An unlimited number of heappoints can be inserted into your program.

The check of an individual block when free or realloc is called is automatic.

All four mechanisms are useful. With the first three mechanisms, the heap error detection is executed at program application speed without context switching to the debugger.

Activating Heap Debugging

One limitation of heap debugging is that it requires that you activate the debugging before any allocations occur in your program. If you attempt to activate the heap debugging features after allocations have already occurred, NightView will inform you of its inability to satisfy your request.

NOTE

If you have mistakenly resumed execution of the program already, kill the program and restart it in the NightView main window. Type the following commands in the **Command** area:

kill run ./app

- Select the Debug Heap... menu option from the Process menu in the NightView main window.

The Debug Heap window is shown.

- Select the Enable heap debugging checkbox at the top of the dialog.
- Press the Medium button in the Debugging Level area.
- Change the Specify check heap freq text field to 1.

	Debug Heap			
Enable heap debugging				
Debugging Level				
Disable	Medium			
- Common Errors Detection				
Block Overrun Dangling Point	ter Uninitialized Field			
-General Settings		Error Control		
Hardware overrun protection			Stop	Print
Specify check heap freq	[1]	free fill modified	×	×
		free not at beginning	×	×
Specify retained free blocks	100	free unallocated	×	×
Specify heap size	Unlimited heap size	malloc zero		
Slop size	0	memalign not power of 2		
Slop Size		out of memory		
Walkback entries / block	8	post-fence modified	×	×
Pre-fence size	4	pre-fence modified	×	×
		realloc not at beginning	×	×
Post-fence size	4	realloc unallocated	×	×
- Fill Settings				
		See for the fill busines Outlife		
Fill malloc space Malloc fill b	oyte: 0xc5	Pre-fence fill byte: 0xbf		-
Fill free space Free fill byte: 0xc3		ost-fence fill byte: 0xaf		-
X Check free fill				
	ОК	Reset Cancel	He	elp

The Debug Heap window should look similar to the following figure:

Figure 3-3. NightView Debug Heap Dialog

- Press the OK button to apply the changes and close the dialog.

These options instruct the debugger to activate heap debugging, retain freed blocks to detect certain kinds of errors, allocate some additional memory past the end of the requested size to detect errors, and stop the program when any heap error is detected.

Controlling the app Program

The third thread created by the main program executes a routine called heap_thread.

This routine iteratively executes various dynamic memory operations based on the setting of the scenario variable. These operations are representative of common user errors relating to dynamic memory.

Let's set a breakpoint on line 115.

- Scroll to line 115 in the source window:

sleep(5);

- Right-click anywhere on that line and select Set simple breakpoint from the pop-up menu.

NOTE

Optionally, you could set a breakpoint on line 115 by using either the Set Breakpoint menu item from the Eventpoint menu or enter the following command in the Command panel of the NightView main window:

break app.c:115

Scenario 1: Use of a Freed Pointer

A common error is to read or write a block of memory that has already been freed.

A way to detect this is to tell NightView to retain freed blocks and fill the freed blocks with a specific pattern. If the blocks are subsequently read, your application may more quickly discover the error since the contents are unexpected. If the blocks are subsequently written, NightView can detect this.

- Resume the process and let it reach the breakpoint on line 115 by pressing the Resume icon on the Process toolbar:



NOTE

Alternatively, you can resume the process by typing **resume** into the **Command** field:

By default, the heap_thread will not actually execute any of the five scenarios.

- To cause it to execute scenario 1, set the variable scenario to 1 by entering the following commands in the Command field:

```
set scenario=1
resume
```

This causes the following snippet of code to be executed after a delay of 5 seconds:

```
ptr = alloc_ptr(1024,3);
free_ptr (ptr,2);
memset (ptr, 47, 64);
```

The last line represents usage of dynamically allocated space that has already been freed.

NightView will detect this at a heappoint inserted by the user, or at a subsequent heap operation (based on the **frequency** setting of the **heapdebug** command), in this case on line 155.

NightView will stop the process once the heap error has been detected and issue a diagnostic similar to the following:

```
Heap errors in process local:3771:
    free-fill modified in free block (value=0x804a818)
#0 0x8048b6d in heap_thread(void*unused=0) at app.c line 155
```

The error refers to the fact that locations within the freed block were modified by the process after the block was freed.

The Data panel is useful for displaying heap-related information as well as a variety of other attributes.

- Select Heap Information from the Data menu.

The Data panel is added to the NightView main window in the same location as the Locals and Context panels. A new tab will be created for the Data panel.

- Click on the newly-created Data tab.
- Resize the first column (if necessary) by clicking on the divider between the column headings and dragging it to the right so that the items of interest below can be seen in their entirety.
- Expand the Configuration item under Heap Information in the Data panel to show the current **heapdebug** settings.
- Expand the Totals item under Heap Information to show summary statistics related to heap activity.

Item		Value		
<u>.</u>	i Heap Information	local:19671		
	⊡… Σ Totals			
	Ever allocated (blocks)	22		
	Ever allocated (size)	11922 bytes		
	Ever allocated (debugger	264 bytes		
	Ever freed (blocks)	5		
	Ever freed (size)	2121 bytes		
	Ever freed (debugger over	. 60 bytes		
	Current allocated (blocks)	17		
	Current allocated (size)	9801 bytes		
	Current allocated (debugg	204 bytes		
	Current retained freed (bl	5		
	Current retained freed (size) 2121 bytes		
	Current retained freed (de	. 60 bytes		
	🗄 🖌 🖌 Configuration			
	heap debugging	on		
	post-fence	4 bytes with 0xaf		
	pre-fence	4 bytes with 0xbf		
	slop	0 bytes		
	···· free fill	with 0xc3		
	malloc fill	with 0xc5		
	hardware overrun protection	disabled		
	frequency	every 1 heap operation		
	heap size	unlimited		
	retain	100 free blocks		
	···· walkback	8 frames		
	check free fill	enabled		

Figure 3-4. Heap Totals and Configuration

NOTE

In general, all information in the Data panel is updated whenever the process being debugged stops.

- Collapse the Totals and Configuration items.
- Click on the tab labeled Locals.

The list of items in the Locals panel changes each time the process stops to represent the local variables associated with the current frame being displayed. Note that the value of the variable ptr is displayed in red because it no longer contains a valid (allocated) heap address.

Expanding the ptr item reveals the (heap info) item. Expanding that item reveals additional information relating to the block that the pointer once referred to including:

- its state freed, but retained
- its address range
- its size
- errors
- free and allocation information, which when expanded include walkback information relating to the routines which allocated and freed the block

tem	Value
🔳 İ	-1086326084
🕂 🕞 iptr	0
≟ ▶ ptr	0x8102c68
🚊 🕂 🔶 (heap info)	
state	freed, but retained
···· range	0x08102c680x08103067
size	1024 bytes
+ errors	1 (as of last heap check)
🖃 free information	0x08048679 in free2() at app.c line 188
🕂 🛷 🖌 configuration	
🗄 🛱 walkback	0x08048679 in free2() at app.c line 188
Frame 0	0x08048679 in free2() at app.c line 188
🛱 Frame 1	0x0804869d in free1() at app.c line 194
🚰 Frame 2	0x080486df in free_ptr() at app.c line 207
🛱 Frame 3	0x08048456 in heap_thread() at app.c line 120
📖 🗖 Frame 4	0x0804e2e1 in xt_new_thread() at xt_pthreads.c line 100
	0x080485b5 in func3() at app.c line 162
scenario	1
	0

Scenario 2: Freeing an Invalid Pointer Value

Another common error is to free a pointer multiple times or to free a value which doesn't actually refer to a heap block.

- Resume the process and let it reach the breakpoint on line 115:

resume

- Set the variable scenario to 2:

set scenario=2 resume

This causes the following snippet of code to be executed after a delay of 5 seconds:

ptr = alloc_ptr(1024,3); free_ptr(ptr,2); free(ptr);

NightView will detect the failure and print a diagnostic similar to the following:

```
Heap error in process local:3771: free called on freed or
unallocated block (value=0x804ac40)
#0 0x8048a78 in heap_thread(void*unused=0) at app.c line 127
```

Another way of obtaining information about the heap block in question is to use the **info memory** command. It provides textual output of the information available in the Locals panel under the ptr item to the Messages panel of the NightView main window.

- Issue the following command in the Command panel:

info memory ptr

NightView will provide output similar to the following in the Messages panel:

```
BX
                                                                     .
Heap error in process local:19671: free called on freed or unallocated
block (value=0x8103090)
#0 0x0804849c in heap thread(void * unused = 0) at app.c line 127
info memory ptr
Memory map enclosing address 0x08103090 for process local:19671:
Virtual Address Range No. bytes Comments
0x080be000 0x08120fff
                      405504 Readable,Writable,Executable
Allocator information for address 0x08103090 for process local:19671:
freed, but retained
in block 0x08103090 .. 0x0810348f (1024 bytes)
no errors detected in block
free information:
   4 post-fence bytes with 0xaf (fence range 0x08103490 .. 0x08103493)
   4 pre-fence bytes with 0xbf (fence range 0x0810308c .. 0x0810308f)
   free fill with 0xc3
   malloc fill with 0xc5
   walkback:
    0x08048679 in free2() at app.c line 188
    0x0804869d in freel() at app.c line 194
    0x080486df in free_ptr() at app.c line 207
    0x08048492 in heap_thread() at app.c line 126
    0x0804e2e1 in xt_new_thread() at xt_pthreads.c line 100
allocation information:
   4 post-fence bytes with 0xaf (fence range 0x08103490 .. 0x08103493)
   4 pre-fence bytes with 0xbf (fence range 0x0810308c .. 0x0810308f)
   free fill with 0xc3
   malloc fill with 0xc5
   walkback:
    0x080485b5 in func3() at app.c line 162
    0x080485d9 in func2() at app.c line 167
    0x08048616 in funcl() at app.c line 173
    0x080486c6 in alloc_ptr() at app.c line 202
    0x0804847f in heap_thread() at app.c line 125
    0x0804e2e1 in xt_new_thread() at xt_pthreads.c line 100
```

Figure 3-5. info memory Command Output

Note that it reports no error in the block per se. The actual problem here is that a second attempt was made to free the block when it already had been freed previously.

In this case, the walkback information associated with the actual free is useful as you can quickly locate what code segment actually freed the block.

Scenario 3: Writing Past the End of an Allocated Block

Another common error is to allocate insufficient space or to write past the end of an allocated block.

- Resume the process and let it reach the breakpoint on line 115:

resume

- Set the variable scenario to 3:

set scenario=3 resume

This causes the following snippet of code to be executed after a delay of 5 seconds:

```
ptr = alloc_ptr(strlen(MyString),2);
strcpy (ptr, MyString); // oops -- forgot the zero-byte
```

NightView will detect the failure and print a diagnostic similar to the following:

```
Heap errors in process local:3771:
    post-fence modified in block (value=0x804b068)
#0 0x8048b6d in heap_thread(void*unused=0) at app.c line 155
```

Note that the description of the variable ptr in the Locals panel does not indicate an invalid status. That is because ptr does point to a valid heap block.

However, expanding the (heap info) information for ptr and the errors list indicates that the block referenced by the ptr is invalid because the post-fence was modified.

ltem	Value						
🔳 (i	-1086326084						
🕂 🝺 iptr	о О						
🗄 🕞 ptr	0x8102c68						
🚊 🗝 🔶 (heap info)							
state	allocated						
···· range	0x08102c680x08102c6f						
size	8 bytes						
⊨ errors	1 (as of last heap check)						
error 1	post-fence modified in block (value=0x8102c68)						
allocation information	0x080485ec in func2() at app.c line 168						
🔳 scenario	3						
🗄 🕟 unused	0						

Figure 3-6. Heap Error Description

Scenario 4: Use of Uninitialized Heap Blocks

Another common error is forgetting to initialize dynamically allocated memory before using it. Code segments may assume that dynamically allocated memory is initialized to zero, as is the case with calloc() but not malloc().

- Resume the process and let it reach the breakpoint on line 115:

resume

- Tell NightView to stop whenever a SIGSEGV is sent to the process and also set the variable scenario to 4:

```
handle sigsegv stop print pass
set scenario=4
resume
```

This causes the following snippet of code to be executed after a delay of 5 seconds:

```
iptr = (int**)alloc_ptr(sizeof(int*),2);
if (*iptr) **iptr = 2778;
```

NightView will detect the failure and print a diagnostic similar to the following:

```
Process local:3771 received SIGSEGV
#0 0x8048ad2 in heap_thread(void*unused=0) at app.c line 138
```

One heap debugging option instructs NightView to fill newly allocated, uninitialized space with a specific pattern to make it easier to detect use of uninitialized memory. The Fill malloc space field in the Debug Heap dialog that we used when enabling heap debugging specified the byte pattern to be 0xc5.

Issue the following command to view the content of the uninitialized memory block:

x/x iptr

A SIGSEGV signal is a fatal error so we must restart the process to continue the tutorial.

- Issue the following command:

kill

 Re-initiate the program by pressing the ReRun icon in the Process toolbar:

5

NOTE

Alternatively, you can issue the following command directly from the **Command** field to initiate the process:

rerun

NOTE

NightView automatically re-applies all eventpoint and heap control settings when it sees the subsequent execution of the program.

Scenario 5: Detection of Leaks

Another situation which may be indicative of error or inappropriate use of memory are leaks. In this instance, we define a leak as a dynamically allocated block of memory that is no longer referred to by any pointer in the program.

Detection of leaks is a *very expensive* process with respect to CPU utilization and intrusion on the user application. As such, leak detection is only executed when an explicit request is made from the user.

- Resume the process and let it reach the breakpoint on line 115:

resume

- Set the variable scenario to 5:

```
set scenario=5
resume
```

This causes the following snippet of code to be executed after a delay of 5 seconds:

ptr = alloc_ptr(37,1);
ptr = 0;

NightView does not detect the leak automatically, as mentioned above. The process will stop again when the breakpoint on line 115 is reached.

- At that time, specifically request a leak report by selecting Heap Leaks... from the Data menu, check the New Leaks radio button, and press OK in the Data Heap Leaks dialog to add the item to the Data panel.

This operation causes NightView to analyze the program for leaks and displays a Leak Sets item in the Data panel. On small programs, this operation may appear to be insignificant, but for larger programs it can take some significant time.

- Click on the Data tab.
- Expand the Leak Sets item, if necessary.

An additional item is displayed for every leak set with a matching block size that was allocated with a matching walkback. Expansion of individual sets provides the common walkback shown for each allocation as well as expandable descriptions of each individual leaked block.

- Expand the leak set item with size 37 and then expand the walkback item associated with it.

Note the walkback indicating that it was allocated by the heap_thread() routine on line 142 of app.c.

	unannannannan Data meannannannannannannannannannannannannann	n BX				
Item	Value					
🖃 🧯 Heap Information	local:29316					
🗄 🛛 🖸 Totals						
🗄 🖌 Configuration	3					
⊨ φ → Leak Sets	local:29316: new at heap operation 10					
⊕ φ leak set, 48 bytes	1 block of 48 bytes, 0xb7f8b218 at <xt_get_default_xtconfig+28></xt_get_default_xtconfig+28>					
😑 🗛 leak set, 37 bytes	1 block of 37 bytes, 0x08048dc3 in func1() at app.c line 174					
🚊 🚰 walkback	0x08048dc3 in func1() at app.c line 174					
Frame 0	0x08048dc3 in func1() at app.c line 174					
🚰 Frame 1	0x08048e58 in alloc_ptr() at app.c line 202					
📖 🚰 Frame 2	0x08048ca3 in heap_thread() at app.c line 142					
⊨ φ+ blocks						
i → 1 0x09d5e298						
	1 block of 16 bytes, 0xb7f8fa1e at <xt_trace_register_thread+28></xt_trace_register_thread+28>					
	1 block of 1 bytes, 0x08048d38 in heap_thread() at app.c line 155					
•						

Figure 3-7. Heap Leaks Display

NOTE

The Leak Sets display will vary depending on your system type. Concentrate on the leak set of 37 bytes as shown above.

NOTE

Unlike most items in the Data panel, the leak sets item is not automatically updated when the process stops. The description is a snapshot of the leaks at a certain moment in the execution of the program, and therefore it will remain unchanged even if additional leaks occur. To get updated information, request another leak report (select Heap Leaks... from the Data menu).

Scenario 6: Allocation Reports

NightView provides a detailed report of all allocated memory.

Construction of this report is a *very expensive* process with respect to CPU utilization and intrusion on the user application execution time. As such, allocation reports are only executed when an explicit request is made from the user.

- Set the variable scenario to 6:

```
set scenario=6
resume
```

This causes additional allocations to be made.

The process will stop again when the breakpoint on line 115 is reached.

 At that time, specifically request an allocation report by selecting Still Allocated Blocks... from the Data menu, click the All Blocks radio button, and press OK in the Data Still Allocated Blocks dialog to add the item to the Data panel.

This operation causes NightView to analyze the program and displays a Still Allocated Sets item in the Data panel. On small programs, this operation may appear to be insignificant, but for larger programs it can take some significant time.

- Resize the first column (if necessary) by clicking on the divider between the column headings and dragging it to the right so that the items of interest below can be seen in their entirety.
- Expand the Still Allocated Sets item, if necessary. An additional item is displayed for every allocation set with a matching block size that was allocated with a matching walkback. Expansion of individual sets provides the common walkback shown for each allocation as well as expandable descriptions of each individual leaked block.
- Expand the allocated set item with size 1048576 and then expand the walkback item associated with it.

tem	Value
⊡ d ♦ Leak Sets	local:29316: new at heap operation 10
Heap Information	local:29829
- ?→ Still Allocated Sets	local:29829: all at heap operation 16
? allocated set, 1048576 bytes	1 block of 1048576 bytes, 0x08048d55 in func3() at app.c line 162
🚊 🗖 walkback	0x08048d55 in func3() at app.c line 162
Frame 0	0x08048d55 in func3() at app.c line 162
🚰 Frame 1	0x08048d79 in func2() at app.c line 167
🗗 Frame 2	0x08048db3 in func1() at app.c line 173
🗗 Frame 3	0x08048e58 in alloc_ptr() at app.c line 202
Frame 4	0x08048cc3 in heap_thread() at app.c line 147
	1 block of 8177 bytes, 0x08048d55 in func3() at app.c line 162
?+ allocated set, 4564 bytes	1 block of 4564 bytes, 0x08048d55 in func3() at app.c line 162
	1 block of 1024 bytes, 0x08048d89 in func2() at app.c line 168
?+ allocated set, 136 bytes	1 block of 136 bytes, 0xb807e34b at <_dl_allocate_tls+59>
Pallocated set, 136 bytes	1 block of 136 bytes, 0xb807e34b at <_dl_allocate_tls+59>
?+ allocated set, 136 bytes	1 block of 136 bytes, 0xb807e34b at <_dl_allocate_tls+59>
	1 block of 62 bytes, 0x08048dc3 in func1() at app.c line 174
?+ allocated set, 48 bytes	1 block of 48 bytes, 0xb804c218 at <xt_get_default_xtconfig+28></xt_get_default_xtconfig+28>
	1 block of 37 bytes, 0x08048dc3 in func1() at app.c line 174
?+ allocated set, 16 bytes	1 block of 16 bytes, 0xb8050a1e at <xt_trace_register_thread+28></xt_trace_register_thread+28>
?+ allocated set, 16 bytes	1 block of 16 bytes, 0xb8050a1e at <xt_trace_register_thread+28></xt_trace_register_thread+28>
	1 block of 16 bytes, 0xb8050a1e at <xt_trace_register_thread+28></xt_trace_register_thread+28>
?+ allocated set, 16 bytes	1 block of 16 bytes, 0xb8050769 at <xt_thread_setup+29></xt_thread_setup+29>
	2 blocks of 1 bytes, 0x08048d38 in heap_thread() at app.c line 155
	N

Note the walkback indicating that it was allocated by the heap_thread() routine on line 147 of app.c.

Figure 3-8. Still Allocated Blocks Display

NOTE

The data from the Still Allocated Sets will vary depending on your system. Concentrate on the allocated set of 1048576 bytes as shown above.

NOTE

Unlike most items in the Data panel, the Still Allocated Sets item is not automatically updated when the process stops. The description is a snapshot of the leaks at a certain moment in the execution of the program, and therefore it will remain unchanged even if additional items are allocated or freed. To update the information, request another allocation report (select Still Allocated Blocks... from the Data menu).

Disabling Heap Debugging

To disable all overhead associated with heap debugging, issue the following command:

heapdebug off

This concludes the tutorial's topic on heap debugging. We will now continue on to other capabilities of NightView.

Debugging Multiple Threads

At this point in the tutorial the user application should be stopped at line 115 in **app.c**.

NOTE

If the application is not stopped at line 115, set a breakpoint on line 115 in **app.c** and resume the process until it stops on that line number. Refer to the previous sections for instructions on setting breakpoints and resuming the process.

Our application consists of the main thread and three additional ones created by the main thread.

When the application hits a breakpoint or is otherwise stopped by NightView, all threads in the application will stop. Similarly, when NightView resumes execution of a thread, all threads will resume execution.

- Click on the Context tab to raise the Context panel.
- Expand the thread which is displayed in green.
- Expand the first item in the walkback list that appeared as a result of the last step

B							
Item	Threads for local:5399 app						
<u>∎</u> @ 5393	C thread 0xb7e016c0						
i± 🎲 5397	C thread 0xb79ecb90 (sine_thread)						
🖶 🎲 5398	C thread 0xb71ebb90 (cosine_thread)						
<u>i</u> 🎲 <u>5399</u>	C thread 0xb69eab90 (heap thread)						
ė. F	#0 0x08048b7d in heap thread(void * unused = 0) at app.c line 115						
i 🗐 🖬	5						
🗄 🕞 iptr	0						
🕨 ptr	0						
🔳 scenario	0						
🛄 🕟 unused	0						
	\$						
I							

Figure 3-9. Context Panel With Stack Frames Expanded

Expanding an individual Frame in the walkback list shows all local variables for that frame. You can further expand composite and pointer variables in the local variables items.

The source shown in the **Source** panel is that associated with the program counter of the thread which caused the process to stop. You can tell which thread you are stopped in by looking for the name of the thread's start routine in paranthesis. NightView automatically assigns names to threads based on the start routine which was passed to

pthread_create(2). Additionally, you can set the name of a thread inside Night-View using the **set-thread-name** command.

You can switch to the context of other threads by clicking on the thread of interest.

Alternatively, you can use the **select-context** command and specify the thread name as shown in the C Threads display or from the output of the **info threads** command:

info threads /v select-context name="cosine_thread"

When thread names are not unique across threads, you can use the thread ID which is always unique. A thread ID is a hexadecimal number representing the thread -- it is assigned by the threads library upon thread creation. The thread ID immediately follows the words "C thread" on each thread item in the Context panel.

- Switch to the context of the thread executing sine_thread() by clicking on it (it is usually the third thread from the top).

The source displayed in the NightView main window changes to line 46 on a call to semop().

NOTE

It is possible that the context of the thread in question could be executing on any line in the range of 45-49.

NOTE

Some versions of glibc on some distributions may be missing proper walkback information for the **semop(2)** routine, which is likely where the thread is stopped. In this case, the **walkback** and **interest** instructions below will not react as described below for this specific example.

The gray triangular arrow before the line number in the source panel represents the fact that we are positioned at a stack frame which is not the topmost stack frame and that the current frame is executing a subprogram call.

By default, NightView hides uninteresting frames. If you desire to see all frames for all routines, even those that have no debug information, you can set your *interest threshold* to the keyword min:

interest threshold min

Once that command is issued, the walkback information shows all frames and you can position to any frame and debug at the assembly level if desired.

- Reset the **interest threshold** to zero via the following command:

interest threshold 0

- Delete the breakpoint on line 115 by right-clicking on that breakpoint in the Eventpoints panel and selecting Delete or by issuing the following command:

clear app.c:115

before proceeding to the next section.

- Resume execution of the process.

NOTE

A significant feature of NightView is the ability to execute most debugging operations without having to stop execution of the process.

All subsequent debugging operations in this tutorial can be done without stopping the process!

Using Monitorpoints

Monitorpoints provide a means of monitoring the values of variables in your program without stopping it. A monitorpoint is code inserted by the debugger at a specified location that will save the value of one or more expressions, which you specify. The saved values are then periodically displayed by NightView in a Monitor panel.

Unlike asynchronous sampling, monitorpoints allow you to view data which is synchronized with execution of a particular location in your application.

- Right-click on line 46 and select Set eventpoint from the pop-up menu and select Set Monitorpoint... from the sub-menu.

NOTE

Alternatively, you could select the Set Monitorpoint... option from the Eventpoint menu or click the Set Monitorpoint icon from the toolbar to launch the Set New Monitorpoint dialog.

Set New M	onitorpoint	×
Location: app.c:46		
	Options:	
	Enable	
	🔿 Enable, disable	e after next hit
	 Disable 	
Condition If:		
Ignore Count: 0		
Name:		
– Expressions		
New		
Expression	Format	Label
	default	
	OK Cance	I Help

Figure 3-10. Monitorpoint Dialog

- Ensure that the Location text field has app.c:46, correcting if it need be.

- Enter the following:

data->count

in the text field below the Expression column head, but do *not* press the Enter key yet.

- You can control the format in which the value is displayed by clicking the option list under the Format column. Using the mouse, click and select Hexadecimal from the list.
- Enter the following in the Label column:

sine count

- While still positioned in the cell under the Label column, press the Tab key. This positions you to the next row and allows you to continue adding expressions.

NOTE

If you have already left the cell and only one row is shown, press the New button.

- In the second row under the Expression column, type the following:

data->value

- Set its label value in the Label column, by typing the following there:

sine value

- Press the OK button in the Set New Monitorpoint dialog.
- A Monitor panel is created containing an entry for the commands entered above.
 - Likewise, set a monitorpoint on line 63 with the same commands as in the previous monitorpoint, substituting cosine for sine in the Label fields.

ltem	Val	lue (1000 ms between samples)
sine count	?	(n/a)
···· sine value	?	(n/a)
···· cosine count	?	(n/a)
cosine value	?	(n/a)

Figure 3-11. NightView Monitor Panel

At this point, the data values in the Monitor panel change.

The values are sampled whenever line 46 or 63 are executed, respectively. NightView displays the latest set of values in the Monitor panel at a user-selectable rate.

Using Eventpoint Conditions and Ignore Counts

All eventpoints in NightView have optional condition and ignore attributes.

A *condition* is a user-supplied boolean expression of arbitrary complexity which is evaluated before the eventpoint is executed. Conditions can involve function calls in the user application.

Similarly, the *ignore* attribute is a count of the number of times to ignore an eventpoint before actually executing it.

Conditions and ignore counts are evaluated by the application itself via patched-in code and, as such, run at full application speed. Other debuggers evaluate the conditions and ignore counts from within the context of the debugger which takes significant time and can drastically affect the behavior of your program.

- Click the cell in the Ignore Count column of the first row of the Eventpoint panel.
- Change the value to 500 and press Enter.

The Monitor panel now indicates that the values for that monitorpoint have not been sampled by displaying a question mark before the value. When the ignore count reaches zero, the values will start updating again.

Finally, monitorpoints can include complex expressions that aren't just simple variables.

- Enter the following commands in the **Command** field of the NightView main window:

```
monitor app.c:93
    p FunctionCall()
end monitor
```

A new item is added to the **Monitor** panel which represents the result of the function call FunctionCall() as executed by the user application each time line 93 is crossed.

Using Patchpoints

Unlike breakpoints and monitorpoints, patchpoints allow you to modify the behavior of your program.

Patchpoints allow you to change program flow or modify variables or machine registers.

First, we will use a patchpoint to branch around some statements in our program.

NOTE

If the source file **app.c** is not displayed, issue the following command:

l app.c:48

- Scroll the source file displayed in the NightView main window and right-click on line 48:

data->angle += data->delta

and select Set eventpoint from the context menu and select Set Patchpoint... from the sub-menu.

NOTE

Alternatively, you could select the Set Patchpoint... option from the Eventpoint menu or click on the Set Patchpoint icon in the toolbar to launch the Set New Patchpoint dialog.

🚯 🛛 Set New Patchpoint 🗖 🗙						
=→ Location: app.c:48						
Options						
Enable						
 Enable, disable after next hit 						
⊖ Disable						
Condition If:						
Ignore Count: 0						
Name:						
Action						
Insert an expression at this location						
 Branch to a different location 						
 Set thread local tag values 						
Evaluate:						
OK Cancel Help						

Figure 3-12. Patchpoint Dialog

- In the Location text area, ensure the text indicates app.c:48.
- Click on the Branch to a different location radio button in the lower portion of the dialog.
- In the Go To: text area, type:

app.c:49

then press the OK button.

This will effectively cause the application to skip execution of line 48, where it updates the angle used in the subsequent sin() call.

Note that the sine value in the Monitor panel stops changing, yet the associated sine count value continues to change.

Alternatively, we can use patchpoints to change the value of expressions or variables.

- Type the following command in the **Command** panel of the NightView main window:

patch app.c:49 eval data->count -= 2

Note that the value of sine count is decrementing, because for each iteration, it continues to be incremented by 1, but now also is decremented by 2.

We can disable the patchpoints without deleting them.

- Select both patchpoints in the Eventpoints panel (as indicated in the Type column by the word Patch), right-click and select Disable from the pop-up menu.

The patches are disabled and the values shown in the Monitor panel return to their original behavior.

Adding and Replacing Functions Dynamically

NightView provides the ability to dynamically add new functions to the application being debugged, as well as to replace existing functions.

- In a terminal session outside of NightView, compile the **report.c** source file which was copied into your current directory in the initial steps of this tutorial:

cc -g -c report.c

- Load the new module into the program using the following command in the Command panel of the NightView main window:

load report.o

NOTE

The source displayed in the NightView main window may change as a result of the **load** command. This annoyance will be addressed in the future.

We have added a simple function which prints information to **stdout**. The function could have been arbitrarily complex and referenced any variable in the application. The only limitation is that the function cannot reference symbols that are absent from the module being loaded and are not already in the user application.

- Issue the following command to see the source code for the function report():

l report.c

You will see that the report() function expects a pair of arguments whose types are char * and double, respectively.

- Go back to the application source file by issuing the following command:

l app.c

We will install a new patchpoint which will call the newly added function.

- Set a patchpoint on line app.c:63 with the following expression:

```
report("cos",data->value)
```

The program is now generating output to **stdout** in the **Messages** panel of the Night-View main window as calls to the report () function are executed.

	Messages	ð×
The value from cos is 0.130526		
The value from cos is 0.139173		
The value from cos is 0.147809		
The value from cos is 0.156434		
The value from cos is 0.165048		
The value from cos is 0.173648		
The value from cos is 0.182236		
The value from cos is 0.190809		
The value from cos is 0.199368		
The value from cos is 0.207912		
The value from cos is 0.216440		
The value from cos is 0.224951		- 1
		-

Figure 3-13. Result of Patching in Call to Newly Loaded Function

- Disable the patchpoint that was just added by clearing its Enabled checkbox in the Eventpoint panel.

Finally, we will replace a function that already exists in the application.

- In a terminal session outside of NightView, list the contents of the source file **function.c** which was copied into your current directory in the initial steps of this tutorial, and compile it with the following commands:

```
cat function.c
cc -g -c function.c
```

- Now load the replacement code by entering the following command in the Command panel of the NightView main window:

load function.o

Note how the Monitor panel value for the FunctionCall() value no longer pertains to the value computed by the application, but rather is a monotonically increasing number as per the source file **function.c**.

- Return the NightView main window source panel to the **app.c** source file via the following command:

l app.c:40

Using Tracepoints

The last portion of NightView we will cover in this tutorial is integration with NightTrace.

A tracepoint is a specialized eventpoint which essentially patches a call to log a trace event with optional arguments.

Even if the application doesn't already use the NightTrace API, NightView can link in the required components and activate the tracing module. Our application already uses the NightTrace API, so this will not be necessary (see the **set-trace** command in the *NightView User's Guide* for more information on using tracepoints in applications which don't already use the NightTrace API).

Suppose that we were interested in measuring the performance of our cycles in the sine_thread() and cosine_thread() routines and that we also were interested in logging data values during the cycle.

- Scroll the source file displayed in the NightView main window and right-click on line 48:

data->angle += data->delta

and select Set eventpoint from the pop-up menu and select Set Tracepoint... from the sub-menu.

NOTE

Alternatively, you could launch the dialog by selecting Set Tracepoint... from the Eventpoint menu or click on the Set Tracepoint icon on the toolbar to launch the Set New Tracepoint dialog.

2	Set New Tracepoint X
🕑 Location:	app.c:48
_Options:	
Enable	
🔵 Enable,	disable after next hit
🔿 Disable	
Condition If:	
Ignore Count:	0
Name:	
Event ID:	
Value:	
	OK Cancel Help

Figure 3-14. Tracepoint Dialog

- In the Location: text field ensure that **app.c:48** is displayed.
- In the Event ID field, type the following:

1

- Press the OK button

Similarly, we'll set additional tracepoints but we will also specify a value to be logged with the tracepoint.

- Set a tracepoint on line **app.c:46** and specify an Event ID of **2** and enter the following in the Value text field:

data->value

- Set a tracepoint on line **app.c:63** and specify an Event ID of **3** and enter the following in the Value text field:

data->value

Trace events can now be logged with the NightTrace tool which is described in the next section of this tutorial.

- Launch NightTrace by selecting the NightTrace Analyzer menu item from the Tools menu of the NightView main window.

The remaining sections of the tutorial do not use NightView, however, we want to keep the tracepoints patched into the executable. We will now detach the program from NightView but it will continue to execute and will retain all patchpoints and tracepoints.

- Stop the processes by typing the following into the Command field:

stop

- Select the Detach option from the Process menu
- Select the Exit NightView option from the File menu to exit NightView.

NOTE

Normally, processes started from within NightView will be killed when NightView exits, even if they have been detached. This is because the shell that is used by NightView to invoke them sends them a SIGHUP signal. Our application ignores SIGHUP, so it can continue to execute.

Conclusion - NightView

This concludes the NightView portion of the NightStar RT Tutorial.

4 Using NightTrace

NightTrace is a graphical tool for analyzing the dynamic behavior of single and multiprocessor applications. NightTrace can log user-defined application data events from simultaneous processes executing on multiple CPUs or even multiple systems. NightTrace can also log kernel events such as individual system calls, context switches, machine exceptions, page faults and interrupts. By combining application events with kernel events, NightTrace presents a synchronized view of the entire system. Furthermore, NightTrace allows users to zoom, search, filter, summarize, and analyze those events in a wide variety of ways.

Using NightTrace, users can manage multiple user and kernel NightTrace daemons simultaneously from a central location. NightTrace provides the user with the ability to start, stop, pause, and resume execution of any of the daemons under its management.

NightTrace users can define and save a "session" consisting of one or more daemon definitions. These definitions include daemon collection modes and settings, daemon priorities and CPU bindings, and data output formats, as well as the trace event types that are logged by that particular daemon.

Invoking NightTrace

NightTrace was invoked during the last step of the Using NightView section.

If you skipped the Using NightView section, execute the steps in "Using Tracepoints" on page 3-34 before beginning this section of the tutorial (and resume execution of the pro-

cess).

11					Night	Trace -	New Ses	sion (o	n rapto	7)				_ D X
<u>F</u> ile	<u>V</u> iew	<u>D</u> aemons	Sea <u>r</u> ch	S <u>u</u> mma	ary <u>P</u> rofil	es Ti <u>m</u> e	lines <u>T</u> ool	s <u>H</u> elp						
Ē	۶ 🛛	1	🚔 ≓	1	1) () 😥 E	ΞΣ		© <mark>⊪</mark> ₩	<u>H</u>	H 101 abc	T	
							Daemor	IS *******	·					. ×
Ту	pe Da	emon			Targ	et	Logged		Lost	St	ate	Attache	d	Buffer
-	🔶 ker	nel_trace_to	_gui		rap	tor	ľ		,	Hal	lted			
_	—		[=_											
	ď	<u>L</u> aunch	<u>R</u> esun	ne	<u>P</u> ause	≡ <u>H</u> alt	Elu	sh <u>D</u>	splay	<u>T</u> rigger	s	<u>E</u> nable	Events	<u>D</u> elete
							· Trace Segr	nents 🔗						P
T	уре 🔻	Trace Se	gment		Target	Log	ged	Lost	Duration	n (sec) Unsa	ved			
										Si	ave Tra	ce Data	Close Tr	ace Data
														//

Figure 4-1. NightTrace main window

Below the menu bar and toolbar, the first page of the NightTrace main window contains the following two panels:

Table 4-1. NightTrace Panels

Daemons	Shows the daemons configured.
Trace Segments	Shows each trace segment (contiguous collection of trace data).

The statistics on the Daemons panel indicate the number of raw events in the shared memory buffer used between the daemon and the user application and the number of raw events written to NightTrace by the daemon (under the Buffer and Logged columns, respectively).

The Trace Segments panel indicates the number of processed events that are currently available for immediate analysis through the Events panels and timelines, which have not been shown yet.

NOTE

The number of events shown in the **Trace Segments** panel will normally differ from the number of events shown in the **Dae-mons** panel. The former are processed events whereas the latter are raw events -- a processed event is often constructed from multiple raw events.

Configuring a User Daemon

NightTrace allows the user to configure a user daemon to collect user trace events.

User trace events are generated by user applications that use the NightTrace API.

We will configure a user daemon to collect the events that our **app** program logs.

To configure a user daemon based on a running application

- Select the Running Application option from the Import... menu option from the Daemons menu.

1		Import Daemon Definitions		×
Target narf	Refresh List			
Program ID 🔺	Program	User	Key File	
32498	арр	jeffh	/tmp/data	
		In	nport Selected Cancel	Help

The Import Daemon Definitions dialog is presented:

Figure 4-2. Import Daemon Definitions Dialog

The Import Daemon Definitions dialog allows the user to define daemon attributes based on a running user application containing NightTrace API calls.

- Select the entry corresponding to the **app** application.
- Press the Import Selected button.

The Import Daemon Definitions dialog closes and a new user daemon is created and added to the Daemon Control Area in the NightTrace main window.

Streaming Live Data to the NightTrace GUI

NightTrace allows you to use a daemon to capture trace events and store them in a file for subsequent analysis or to stream the events directly into the graphical interface for live analysis.

Our daemon is configured for live streaming.

- Select the daemon labeled app_data from the Daemons panel in the NightTrace main window.
- Press the Launch button.
- Press the Resume button.

The daemon is now collecting events which are being generated by the **app** program from the tracepoints we inserted in "Using Tracepoints" on page 3-34.

In the Daemons panel, the count of events shown in the Buffer column will begin to change.

-				nananan Daemons				····· ••· •••
	Туре	Daemon	Target	Logged	Lost	State	Attached	Buffer
	-к-	kernel_trace_to_gui	narf			Halted	· · · · · · · · · · · · · · · · · · ·	
	-U-	app_data	narf	0	0	Logging	3	2311
		U Launch	e II <u>P</u> ause	<u>H</u> alt	lush <u>D</u> isplay	Triggers	<u>E</u> nable Ev	ents

Figure 4-3. Logging Data

NOTE

A tabbed page is created in the NightTrace main window when Launch is pressed. This page is an automatically customized page containing a list of the events logged and a timeline for graphical representation of those events.

- Click on the newly-created tab labeled app_data which contains the Events panel and the timeline associated with those events.

9		N	ightTra	ce - New	/ Sess	ion(Uns	avec	d) (on r	apto	r)					
<u>F</u> ile <u>V</u> iew	<u>D</u> aemons Sea <u>r</u>	:h S <u>u</u> mn	nary <u>P</u> ro	files Ti <u>m</u>	elines	<u>T</u> ools <u>I</u>	<u>H</u> elp								
6	े 🎾 🦉 =	• 🕼	1	ri 🗩	9	ð 🔎	Σ		0 <mark> </mark>	*	<u>H</u> .	101 abc	Ŧ		
Trace ap	p_data														
						Events 🔗									Ð×
Offset	E	vent CPl	J	Process		Threa	d	Time	(sec)	Tag	Des	ription	_		
0	NT_TI	MER	0	0			0	0.00000	0000		arg1	=0x0			
1	NT_TI	MER	0	0			0	0.00800	0000		arg1	=0x0			
					·i·i·i· a	ipp_data 🖓									
															-
															-
User Events															-
User Events	•.														-
		0.0000	001s												1
		0.0001	0s			0.0	0310s			1		0.00610s			
				<u> </u>	1			I		<u> </u>					9
Current Tim	e 0.000000000	Hover	time from	current tim	eline =	0.0000000	00								ר
Start Time	0.00000000														
End Time Span	0.000000000	Currer	nt offset=0	cpu=0 id=	NT_TIN	MER proc	=0 thr	=0 time(s	ec)=0.	000000	000 (0.	000000000) from cu	urrent time	j –
	5.000000000	arg1=	0x0												
															Ē
•														•	
over time fro	m current timeline	= 0.00000	0000												

Figure 4-4. app_data Page

Initially, the panels will be mostly blank.

You can force events to be flushed from the daemon buffer and output stream to be brought into the segment area for immediate viewing by zooming out on a timeline.

- Click anywhere in the display area containing the timelines.
- Press UpArrow to zoom out
- Press Alt-UpArrow to zoom out completely.

The Events list will be populated with the events currently logged and the timeline will graphically display those events.

NOTE

If you plan to leave the tutorial for an extended period of time before returning, press the Pause button on the Trace page to temporarily prevent the collection of trace points. When you return, press the Resume button.

Using NightTrace Timelines

	NightTrace - New	Session(Unsav	ved) (on ra <u>pto</u>	r)		0 8
jle <u>V</u> iew <u>D</u> aemons Sea <u>r</u> ch	n S <u>u</u> mmary <u>P</u> rofiles Ti <u>m</u> e	lines <u>T</u> ools <u>H</u> e	lp			
Final app_data	· 🕼 🕅 🔊 🌶		· ·	*⊕ ⊭		
1	ent CPU Process	Thread	Time (sec)	Tag	Description	
16739	3 app	cos	209.935575514		arg1=-0.829038	
16740	1 app	sin	209.935582448			
16741	2 app	sin	209.935585971		arg1=-0.573576	
16742	1 app	cos	209.980145884			
1						
Jser Events:	0.1s	00.1s	200.1s		300.1s	
Jser Events:		00.1s 1 1 1 1 1 1 1 1 10.1s 1 1 1 1 1 1 1 1	200.1s 		300.1s	
Current Time 209.9507025 Start Time 0.00000000	0.1s 10	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	200.1s		300.1s	
	0.1s 10	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	200.1s	85971 (0.	300.1s	rrent time)
Current Time 209.9507026 Start Time 0.00000000 End Time 419.9014053	0.1s 10 Hover time from current time	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	200.1s	85971 (0.	300.1s	rent time)

Figure 4-5. NightTrace Timeline

The timeline contains static and dynamic labels and event and state graphs.

By default, NightTrace detects the threads that have registered themselves through Night-Trace API calls and creates individual labels and graphs for each thread. In addition, there is a user events graph near the bottom that shows events for threads.

NOTE

You may see a blank label and graph in your timeline. This is likely the label and graph for the main thread. The contents of the label are not shown until at least one event is logged by the main thread. In our application, the main thread does not log events so the row will remain blank. In "Using Tracepoints" on page 3-34 in the Using NightView section, we inserted tracepoints into the sine thread, which registered itself with the string "sin".

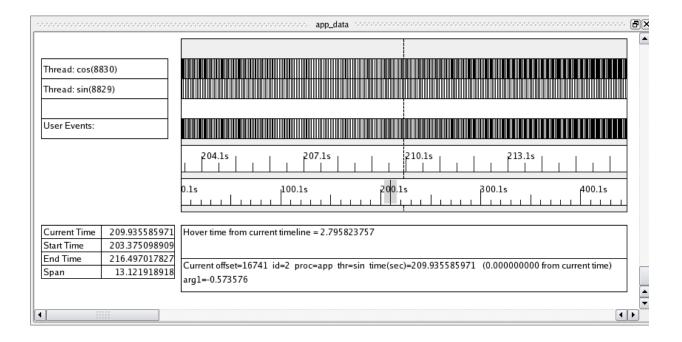
Zooming

Each vertical line in the graph represents at least one event. You can zoom in and zoom out to adjust the level of detail.

- Left cllick anywhere within the timeline
- Press the DownArrow key repeatedly until you can see individual lines in the graph
- Press the UpArrow key to zoom back out
- If you have a mouse wheel, move the wheel back and forth to zoom in and out

The vertical dashed line is the current timeline and is directly connected to the highlighted event in the Events panel.

Left-clicking the mouse in the display area moves the current timeline. The information in the Event Detail area below the timeline changes to reflect the event closest to the left of the current timeline.



Moving The Interval

Figure 4-6. Timeline Interval Panel

By default, each timeline panel has two ruler rows positioned below the event graphs and above the descriptive boxes at the bottom of the panel.

The ruler on top indicates the timespan currently shown.

The ruler on the bottom indicates the timespan for all data currently available for viewing. This ruler is called the control ruler and has a gray area within it. The gray area represents the amount of the entire timespan that is currently shown in the panel. Thus zooming in will decrease the width of the gray area and zooming out will have the opposite effect.

NOTE

If you do not see a gray area, zoom out until you do.

There are several methods of moving through the entire timeline.

- Press the RightArrow key

This causes the current timeline to go to the next event. If you are zoomed out too far, you may not notice the timeline moving. In this case, either zoom out or hold the Right key down until you can see the timeline move.

Alternatively, pressing the LeftArrow key causes the current timeline to go to the previous event.

- Press Ctrl+RightArrow

This causes the displayed interval to move 25% of a section to the right by default. The section is the amount of time currently visible in the interval. Notice how the gray are in the control ruler moves.

Alternatively, pressing Ctrl+LeftArrow causes a shift one section to the left.

- Click midway between the gray area and the right hand portion of the control ruler

Clicking anywhere in the control ruler causes the interval to shift to be centered at the selected time at the current zoom setting.

Thus to move the very beginning of the data set or the end, you can click the beginning or end of the control ruler.

Using the Events Panel for Textual Analysis

		energian Events					Ð
Offset	Event CPU	Process	Thread	Time (sec)	Tag	Description	
16733	2	app	sin	209.835132237		arg1=-0.559193	_
16734	1	app	cos	209.879990717			
16735	3	app	cos	209.885256169		arg1=-0.833886	
16736	1	app	sin	209.885263100			
16737	2	app	sin	209.885266825		arg1=-0.566406	
16738	1	app	cos	209.930036313			
16739	3	app	cos	209.935575514		arg1=-0.829038	
16740	1	app	sin	209.935582448			
16741	2	арр	sin	209.935585971		arg1=-0.573576	
16742	1	app	cos	209.980145884			
16743	3	app	COS	209.985604692		arg1=-0.824126	
16744	1	app	sin	209.985613353			
16745	2	app	sin	209.985616224		arg1=-0.580703	
16746	1	app	cos	210.030265007			
16747	7			210.025644020		1 0.010150	

Figure 4-7. Events Panel

The events shown in the Events panel are synchronized with the events shown in the timeline. The highlighted event indicates the current timeline.

- Click on a line in the Events panel
- Press the DownArrow key to advance to the next event.
- Press the UpArrow key to advance to the previous event.

Whenever an event is selected or the current event line moves, the Event Detail area below the timeline on the right shows additional information about the event, if available.

- Press the PageDown to advance to the next set of events.
- Press the PageUp to shift to the previous set

These actions only move the current timeline by the number of events that can be shown in the Events panel.

Customizing Event Descriptions

The event values we logged with the **tracepoint** commands in NightView were event IDs 1-3. We will customize the description of these events.

- Click on a row in the Event panel that shows event ID 1.

- Right-click that row and select Edit Current Event Description... from the context menu.

Add Event Description	×
	_
Code 1	
Name 🚺]
Description	
	_
OK Cancel Help	

Figure 4-8. Add Event Description dialog

- Enter:

cycle_start

in the Name field.

- Press OK.
- Right-click on an entry whose value in the Event column has the value **2**.
- Select Edit Current Event Description... from the context menu.
- Enter:

cycle_end

in the Name text field.

- Press the OK button.

Offset	Event CPU	Process	Thread	Time (sec)	Tag	Description	ŀ
16733	cycle_end	app	sin	209.835132237		arg1=-0.559193	_
16734	cycle_start	app	cos	209.879990717			
16735	3	app	cos	209.885256169		arg1=-0.833886	
16736	cycle_start	app	sin	209.885263100			
16737	cycle_end	app	sin	209.885266825		arg1=-0.566406	
16738	cycle_start	app	cos	209.930036313			
16739	3	app	cos	209.935575514		arg1=-0.829038	
16740	cycle_start	арр	sin	209.935582448			
16741	cycle_end	app	sin	209.935585971		arg1=-0.573576	
16742	cycle_start	app	cos	209.980145884			
16743	3	app	cos	209.985604692		arg1=-0.824126	
16744	cycle_start	app	sin	209.985613353			
16745	cycle_end	app	sin	209.985616224		arg1=-0.580703	
16746	cycle_start	app	cos	210.030265007		-	
16747				210 025644020		1 0.010150	

The descriptions of the events in the Events panel now correspond to the textual identifiers we assigned to them.

Searching the Events List

We can use the search capabilities of NightTrace to search for a specific occurrence of an event or condition relating to an event or its arguments.

- Select the Change Search Profile... menu item from the Search menu in the NightTrace main window or press Ctrl+F.

A new page is created containing the Profile Status List panel and the Profile Definition panel:

				Nig	htTra	ce - (New	Ses	sion(Unsa	aved							
<u>F</u> ile	<u>V</u> iew	<u>D</u> aemon	s Se	ea <u>r</u> ch	S <u>u</u> m	mary	Prot	files	Ti <u>m</u> e	lines	Tool	s <u>H</u> e	lp					
B			\$	#		E	1	Þ	P	P	Þ	Σ		■	o∥ II	* ⊕		
Trace		p_data	Prof															
	Name					itus	ronie	Cou			Last		_					۳
., , , , , , , , , , , , , , , , , , ,		-				itus	_				Lust							
																		-
							Profile	Defin	ition									Ŀ
	Ke	ev / Value	Con	ndition					Res	et		Choos	e Pr	ofile				
	Ke	ey / Value	_					•	Res	et		Choos	e Pr	ofile			_	
		Events	ALL	-				•	Res	et		Choos	e Pr	ofile	B	rowse		
I		•	ALL	-				•	Res	et		Choos	e Pr	ofile	B	rowse rowse		
I	Exclud	Events		- NE				•	Res	et		Choos	e Pr	ofile	B			
I	Exclud	Events le Events		- NE JE					Res	et		Choos	e Pr	ofile	B			
I	Exclud	Events le Events Condition		- NE JE -					Res	et		Choos	e Pr	ofile		rowse		
I	Exclud (Events le Events Condition Processes Threads	ALL NOI TRU ALL	- NE JE -	ohtīra	ce/bir)/ever					Choos	e Pr	ofile	B	rowse rowse		
I	Exclud (Events le Events Condition Processes Threads out Script	ALL NOI TRU ALL /usi	- NE JE -	ghtTra			nt-sur	mmary			Choos	e Pr	ofile	B	rowse rowse rowse		
1	Exclud (Events le Events Condition Processes Threads out Script CPUs	ALL NOI TRU ALL ALL Jusi	- JE - r/lib/Ni	ghtTra		n/ever (masł	nt-sur	mmary			Choos	e Pr	ofile	B	rowse rowse rowse		
	Exclud (Events le Events Condition Processes Threads out Script	ALL NOI TRU ALL ALL Jusi	- JE - r/lib/Ni	ghtTra			nt-sur	mmary			Choos	e Pr	ofile	B	rowse rowse rowse		
	Exclud (Events le Events Condition Processes Threads out Script CPUs	ALL NOI TRU ALL ALL Jusi	- JE - r/lib/Ni	ghtTra			nt-sur	mmary			Choos	e Pr	ofile	B	rowse rowse rowse		
	Exclud (Events le Events Condition Processes Threads ut Script CPUs Name	ALL NOI TRU ALL (usi all con	- JE - r/lib/Ni			(mask	nt-sur k=all)	mmary	y.sh		Halt				rowse rowse rowse		2
	Exclud (F Outp	Events le Events Condition Processes Threads ut Script CPUs Name	ALL NOI TRU ALL (usi all con	- NE JE - - r/lib/Ni			(mask	nt-sur k=all)	nmary	y.sh						rowse rowse rowse		•

Figure 4-9. Searching using the Profiles panel

- Press the Browse... button to the right of the Events field.

Select	Events (on	raptor)	(
NONE			
ALL			
ALLADA			
ALLAI			
ALLKERNEL			
ALLUSER			
BKL_LOCK			
BKL_SPIN			
BKL_UNLOCK			
BUFFER_END			
BUFFER_START			
CUSTOM			
EVENT_CREATED			
EVENT_DESTROYED			
EVENT_LOST			
EVENT_MASK			
FBS_OVERRUN			
FBS_SYSCALL			
FILE_SYSTEM			
GLOBAL_CLI			
GLOBAL_STI			
GRAPHICS_PGALLOC			
IPC			
IRQ_ENTRY			
IRQ_EXIT			
KERNEL_TIMER			
MEMORY			
NETWORK			
NT_ASSOC_PID			
NT_ASSOC_TID			
NT_BEGIN_BUFFER			
NT_BEGIN_SEGMENT			
NT_BEGIN_STREAM			
NT_CONTINUE			
NT_DBL_CONTINUE			
NT_DISCARDED_DATA			-
× Search:		🕇 Previous 🗌	Match case
·	Select	Cancel	Help

Figure 4-10. Browse Events Dialog

- Click in the Search text field and type cycle. The first event name that includes that word is shown. Ensure that cycle_end is selected in the event list, or press the Next icon until it is. Then press the Select button.
- Enter the following text in the Condition text field of the Profile panel:

arg_dbl > 0.8

- Enter the following text into the Name text field:

obtuse

- Press the Add button in the Profiles panel.

A profile called obtuse is now defined and appears in the Profile Status List panel.

- Press the Search Forward button at the bottom of the Profiles panel.

The current timeline is moved to the first event that matched the search criteria, that being the end of a cycle when the sine value exceeded 0.8.

NOTE

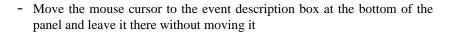
If a pop-up dialog telling you that NightTrace has reached the end of the available dataset and asks you whether it should resume the search at the beginning, press OK.

- Click on the tab labeled app_data and verify that the current event listed in the Events panel indicates arg1 with a value exceeding 0.8.

Offset	Event C	PU Process	Thread	Time (sec)	Tag	Description	
5146	3	app	cos	89.274949717		arg1=0.953717	
5147	cycle_start	app	sin	89.274988678			
5148	cycle_end	app	sin	89.275119031		arg1=0.798636	
5149	3	app	COS	89.326889883		arg1=0.951057	
5150	cycle_start	app	sin	89.326929355			
5151	cycle_end	арр	sin	89.327062778		arg1=0.803857	
5152	cycle_start	app	sin	89.379099168			
5153	cycle_end	app	sin	89.379254269		arg1=0.809017	
5154	3	app	cos	89.379630470		arg1=0.948324	

Figure 4-11. Events Panel After Search

Similarly, the timeline shows a description of the current event in the Event Detail area in the bottom portion of the panel.



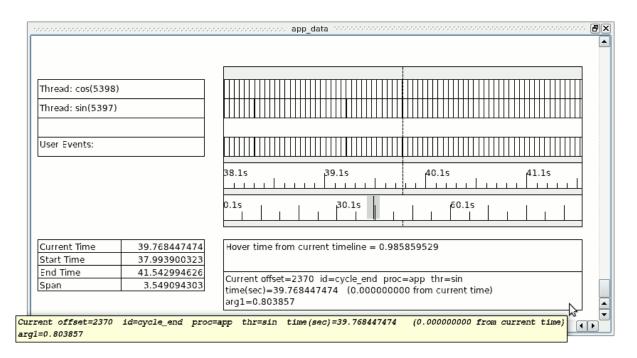


Figure 4-12. Timeline Panel w/ Tool Tip

A tool tip is also displayed which describes the current event.

NOTE

It is possible that the search will fail if an insufficient number of events have been brought into live analysis. If this occurs, bring in more events using the Event list scroll bar and retry the search by pressing the forward search icon on the tool bar.

Halting the Daemon

Since the NightTrace portion of the tutorial is rather lengthy and may likely be a new experience for many users, we will halt the daemon to reduce memory usage.

Examine the daemon statistics in the Daemon Control Area on the first page. If the application has logged over 100,000 events, halt the daemon by pressing the Halt button to reduce memory usage as we slowly move through the NightTrace portion of the tutorial.

NOTE

Do not be concerned if the number of events shown in the **Trace Segments** panel is smaller than the number of events shown in the Daemon Control Area just before you halted the daemon. The latter shows raw event counts whereas the **Trace Segments** panel shows processed event counts -- a processed event is often constructed from multiple raw events.

If it has not reached this stage yet, you may leave the daemon running and occasionally glance at the statistics. If NightTrace becomes unresponsive or slows down as the event counts reach into the millions, halt the daemon. NightTrace has a configurable memory consumption limit that will automatically halt the daemon when the limit is reached; a dialog will be presented informing the user when this occurs.

Using States

In addition to displaying individual events, NightTrace can display states.

- Click on the **Profiles** tab created in "Searching the Events List" on page 4-12.

~~	View Daemor		-	_		_		_		-	_		_		v		
D	- 🕞 🕴 🎇	3	≠ 0				٦	P	P	P	Σ		Ů	H.	₩	<u>H</u>	
Trace		Profi	i														_
						ofile	Statu Co				Offse	_		.*.*.*.		·····	6
ype	Name obtuse	-		Stat Tru				148	-	648	Oπse	eτ					
					Pr	rofile	Defin	ition	· · · · · · · · · · · · · · · · · · ·						.*.*.*.*.*		E
	Key / Value		dition				•	Res	et		hoose	Profi	le				
		Con					•	Res	et		choose	Profi	le	Br	owse		
	Key / Value	Con Cycl	e_end				-	Res	et		Choose	Profi	le)		owse owse		
	Key / Value Events	Con cycl	e_end E).8			•	Res	et		Choose	Profi	le)				
	Key / Value Events Exclude Events	Con cycl NON arg	e_end E).8			•	Res	iet		Choose	Profi	le	Br			
	Key / Value Events Exclude Events Conditior Processes Threads	Con cycl NON arg_ ALL	e_end E dbl > 0								Choose	Profi		Br	owse		
	Key / Value Events Exclude Events Conditior Processes Threads Output Script	Con cycl NON arg ALL ALL	e_end E dbl > 0		e/bin/	/ever					Choose	Profi	le)	Br Br	owse owse		
	Key / Value Events Exclude Events Conditior Processes Threads	Con cycl NON arg ALL ALL (usr, all	e_end E dbl > 0 Ylib/Nigł					nmar			Choose	Profi		Br Br	owse owse owse		
	Key / Value Events Exclude Events Conditior Processes Threads Output Script	Con cycl NON arg ALL ALL (usr, all	e_end E dbl > 0 Ylib/Nigł				nt-sur	nmar			Choose	Profi		Br Br	owse owse owse		

The Profiles page is displayed with the previously defined profile selected.

Figure 4-13. Profiles Panel With Obtuse Profile Selected

- Press the Reset button.
- Select State in the Key / Value option list.
- Enter:

cycle_start

in the Start Events text area

- Enter:

```
cycle_end
```

in the End Events text field.

- Enter:

sin

in the Threads text field.

- Enter:

sine

in the Name text field.

- Press the Add button.

A state named sine has now been defined and occurrences can be displayed in the graphs in the display page.

- Click on the tab labeled app_data to show the timeline.
- Right-click anywhere in the display area and select Edit Mode from the context menu or press Ctrl-E to enter *edit mode*.

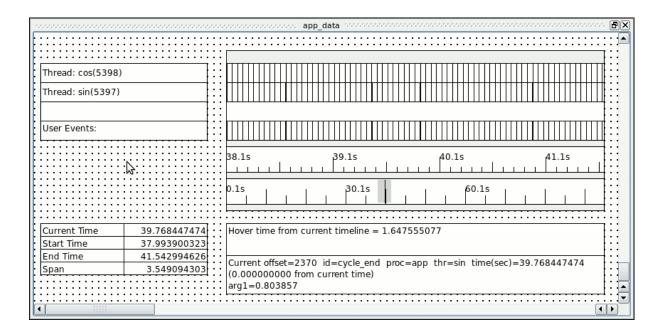


Figure 4-14. Timeline Editing

- Double-click on the graph associated with the row labeled "Thread: sin". That graph is a row with vertical lines representing events inside the larger graph area, aligned with the label "Thread: sin".

8	Edit State Graph Profile		
Key / Value	State Reset Choose Profile		
Start Events	NONE	Browse	
End Events	NONE	Browse	
Events	ALLUSER	Browse	
Start Condition	TRUE		
End Condition	TRUE		
Events Condition	TRUE		
Processes	ALL	Browse	
Threads	sin	Browse	
Event Color	black		
State Color	blue		
CPUs	all (mask=all)		
	Cane	cel He	lp

The Edit State Graph Profile dialog is displayed as shown below:

Figure 4-15. Edit State Graph Profile dialog

- Select State from the Key / Value option list.
- Press the Choose Profile... button.

The Choose Profile dialog is displayed.

- Select the sine state from the list.
- Make sure the Import by reference checkbox is checked.
- Press Select.

- Click on the colored button to the right of the row labelled State Color. The Select color dialog is presented.

Sel	ect color X
Basic colors	
Custom colors	Hue: 240 Red: 0 Sat: 255 Green: 0 Val: 255 Blue: 255 Add to Custom Colors Add to Custom Colors

- Select a pleasing color in the Select color dialog and press OK.
- Press OK in the Edit State Graph Profile dialog.
- Right-click anywhere in the display area and select Edit Mode from the pop-up menu or press Ctrl-E to return to *view mode*.

The graph has now been configured to display the sine state as a solid bar in the lower portion of the state graph. Events will still be displayed as vertical black lines that extend over the entire vertical height of the graph.

It is likely that the display page has not changed in a significant way. This is because the cycle_start and cycle_end events occur so closely together in time that you cannot distinguish them at the current zoom setting.

- Click in the middle of the state graph.
- Zoom in using the mouse wheel or using the Zoom In icon on the toolbar or the Down Arrow key until the two events can be distinguished and a state bar is shown.

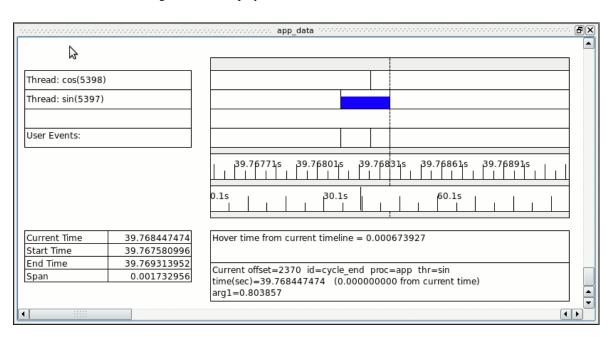
You may need to readjust the current timeline as you zoom in.

NOTE

If the Down Arrow key has no effect, press the Num Lock key and try again.

NOTE

The state may vanish at some zoom levels where it is still very small compared to the zoom level's scale. If so, just continue to zoom in and it will reappear.



The figure below displays an instance of the sine state.

Figure 4-16. Sine State in Timeline

NOTE

If no states are visible, recheck the definition of the sine profile in the Profiles panel as described in "Using States" on page 4-17.

Displaying State Duration

The duration of the most recently completed state can be displayed via a data box.

- Right-click anywhere in the display area on the page labeled app_data and select Edit Mode from the pop-up menu or press Ctrl-E to enter *edit mode*.
- Right-click anywhere in the grid and select Add Data Box option from the pop-up menu.

The cursor will turn into a + character.

- Using the left mouse button, click an empty area in the left-side of the display page on the grid (outside of any currently displayed graph or data box
 -- i.e. only on an available area whose background shows the dotted grid) and drag the mouse to create the outline of the new data box -- release the mouse button.
- Double-click the data box. The Edit Data Box Profile dialog is presented.
- Enter the following into the Output field:

```
format ("cycle = %f ms", state_dur(sine)*1000.0)
```

- Press the OK button.
- Right-click anywhere in the display area and select Edit Mode from the pop-up menu or press Ctrl-E to return to *view mode*.

The data box now displays the length of the most recently completed instance of the sine state in milliseconds.

Generating Summary Information

In addition to obtaining detailed information about specific events and states, summary information is easily generated.

- Select the Change Summary Profile... menu item from the Summary menu.
- Select the profile matching the sine state from the list of profiles shown in the Profile Status List panel.

It is likely that the sine profile is already selected. Check the profile name shown in the Name text area near the bottom of the dialog.

- Press the Summarize button.

le <u>V</u> iew	<u>D</u> aemons	Sea <u>r</u> ch S <u>u</u> mmary	<u>P</u> rofiles Ti <u>m</u> elines	<u>T</u> ools <u>H</u> elp			
3 🛛	ें 🎾 🎽	; = 77 🕅	¢ 🤉 🖉 🔊	🖻 🔎 Σ 📃	o <mark>⊪ *</mark> <u>⊎</u> ≝	표 101 abc @	ם
ace app	p_data Pro	ofiles sine (0 to 50	0457)				
			sine ((0 to 50457)			*******
umber of sta	ates found:	16819					
nimum stat erage state tal of state		0.000083790 at offs 0.000001712 at offs 0.000003453 0.058079649 d: 16819					
winnum cto	te gan:						
inimum stat /erage state	tegap: (egap: 0	0.050539933 at offse 0.049730076 at offset 0.050169556 3.801760127			····		
nimum stat rerage state stal of state	tegap: (egap: 0).049730076 at offset .050169556		Event CPU	Process	Thread	Time (sec)
nimum stat rerage state tal of state	te gap: (e gap: 0 gaps: 84	0.049730076 at offset .050169556 3.801760127	: 2589	Event CPU cycle_start	Process app	Thread	Time (sec) 0.032374959
nimum stat erage state tal of state Offset T	te gap: (e gap: 0 gaps: 84	0.049730076 at offset .050169556 3.801760127 Duration (sec)	t: 2589 Gap (sec)				
nimum stat rerage state tal of state Offset ▼ 2	te gap: (e gap: 0 gaps: 84 End Offset 3	0.049730076 at offset 050169556 3.801760127 Duration (sec) 0.000003815	t: 2589 Gap (sec) 0.00000000	cycle_start	app	sin	0.032374959
nimum stat rerage state tal of state Offset 2 5	te gap: (e gap: 0 gaps: 84 End Offset 3 6 9	0.049730076 at offset 0.050169556 3.801760127 Duration (sec) 0.000003815 0.000005352	Contemporary Conte	cycle_start cycle_start	app app	sin sin	0.032374959
onimum stat verage state tal of state Offset 2 5 8	te gap: (e gap: 0 gaps: 84 End Offset 3 6 9 12	0.049730076 at offset 0.050169556 3.801760127 Duration (sec) 0.000003815 0.000005352 0.000003998	Cap (sec) 0.00000000 0.050172504 0.050299223	cycle_start cycle_start cycle_start	app app app	sin sin sin	0.032374959 0.082551278 0.132855853
Offset 2 2 5 8 11	te gap: (e gap: 0 gaps: 84 End Offset 3 6 9 12	0.049730076 at offset 0.050169556 3.801760127 Duration (sec) 0.000003815 0.000005352 0.000003998 0.000002304	Gap (sec) 0.000000000 0.050172504 0.050299223 0.050332313	cycle_start cycle_start cycle_start cycle_start	app app app app	sin sin sin sin	0.032374959 0.082551278 0.132855853 0.183192163
5 8 11 14	te gap: (e gap: 0 gaps: 84 End Offset 3 6 9 12 15	0.049730076 at offset 0.050169556 3.801760127 Duration (sec) 0.000003815 0.000003998 0.000002304 0.000002314	Gap (sec) 0.00000000 0.050172504 0.05032313 0.050095281	cycle_start cycle_start cycle_start cycle_start cycle_start	app app app app app	sin sin sin sin	0.032374959 0.082551278 0.132855853 0.183192163 0.233289748
Offset Contract of state Cont	te gap: (e gap: 0 gaps: 84 End Offset 3 6 9 12 15 18	0.049730076 at offset 0.050169556 3.801760127 Duration (sec) 0.000003815 0.000003998 0.000002304 0.000002314 0.000001938	Cap (sec) 0.00000000 0.050172504 0.05032313 0.050095281 0.050209080	cycle_start cycle_start cycle_start cycle_start cycle_start cycle_start cycle_start	app app app app app app	sin sin sin sin sin	0.032374959 0.082551278 0.132855853 0.183192163 0.233289748 0.283501141

A new page is created displaying the results of the summary.

Figure 4-17. Summary Results Page

The summary results page provides a number of columns of information including the state's starting and ending offsets, the state's duration, and the gap between a state and its most recent previous occurrence. You can click on the column headers to control how the list is sorted.

Double-clicking on a row in the list positions the current timeline to the beginning of that instance of the state and creates a tag at that position.

To go to the instance of the longest state duration, do the following:

- Click on the Duration header to select duration as the sort key

Repeated clickin on the header toggles the direction of the sort.

- Click the Duration header until the sort order is largest to smallest.

- The instance of the state with the longest duration is shown in the top row
- Double click on that row

The current timeline is moved to that instance of the state, as shown in the Events and Timeline panels.

The minimum and maximum state occurrences are often of interest. However, a graphical display of state durations can be more enlightening.

- Select the Graph State Durations... option from the Summary menu in the Profiles dialog.
- Change the standard deviation value in the dialog to **0**.
- Press the OK button.

1		l la	 	u d.	 	 <u>.</u>	l	N	
0.1s	30	0.1s	1		60.1s				90.1s
0.1s	30).1s			60.1s				90.1s
Hover time from current time thr=sin time(sec)=39.2623 Current offset=3925 id=cyc	36148						·		

Figure 4-18. Summary Graph

A new page is created with a summary graph and a textual description of the instances of the state.

The row with blue shown indicates individual instances of the state. If the blue bar appears to be a single bar, zoom in until individual instances can be seen.

- Zoom all the way out by pressing Alt+UpArrow.

A data graph is shown in the wide column beneath the row with blue state indicators.

Each red line indicates the duration of an instance of the state.

Sometimes a single occurrence of the state may be much longer than most occurrences. In such cases, the detail is obscured.

- Click anywhere in the data graph and enter Edit mode by pressing Ctrl+E.
- Double-click anywhere in the data graph.

	Edit Data Graph Profile (on raptor)
Key / Value	Condition Reset Choose Profile
Events	
Exclude Events	NONE Browse
Condition	offset==end_offset(sine)
Processes	ALL Browse
Threads	sin Browse
CPUs	
Value	X X X X X X X X X X X X X X X X X X X
Max Value	0.000083790
	Drawing and Coloring Options
	OK Cancel Help

Figure 4-19. Data Graph Profile Dialog

- Change the Max Value text field to 0.001 or a value representative of most of typically long state durations (refer to the sorted list of state durations in "Generating Summary Information" on page 4-23).
- Press the OK button.

- Return from Edit mode by pressing Ctrl+E.

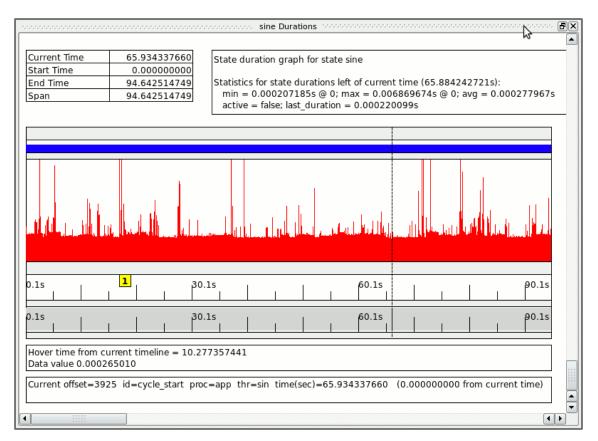


Figure 4-20. Modified Data Graph

The graph now shows more detail. The current timeline in the data graph is linked to the current timeline in all timelines and the Events panel. Clicking anywhere in the graph will move the current timeline in all such panels.

Defining a Data Graph

The area containing the timelines has a blank area above the graphs for each of the threads in the program. We will now add a data graph in this area.

- Raise the app_data timeline page by clicking on its tab.
- Remove the Events panel by clicking the close box at the upper right-most portion of the panel's title bar.
- Right-click anywhere in the display panel labeled app_data and select Edit Mode from the pop-up menu or press Ctrl-E to enter *edit mode*.
- Click on the middle of the upper horizontal line of the column containing the graphs in the panel.

- Move the mouse cursor so that it hovers over the middle of the upper horizontal line of the column.
- When the cursor changes to two arrows pointing up and down, click and drag the upper boundary of the column upward to make space for the data graph.

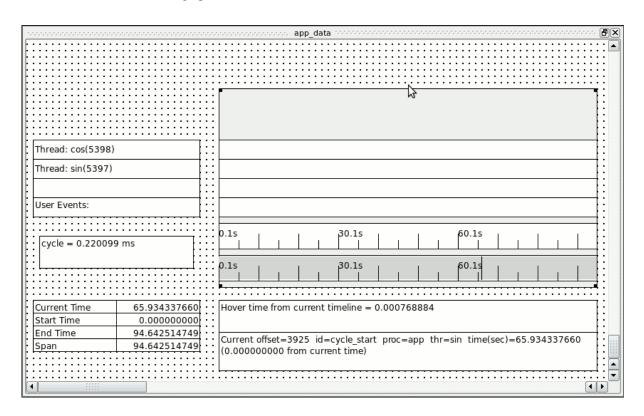


Figure 4-21. Resizing in Progress

- Release the mouse button when sufficient space has been made (approximately an inch or more vertically).
- Click on the upper horizontal line of the column.
- Right-click inside the graph container and select Add to Selected Graph Container from the pop-up menu and select Data Graph from the sub-menu.

The cursor changes to a block plus sign

- Click in the space created by the previous steps.

·····		annanananan app_data aanaanananananananananananananananan
· · · · · · · · · · · · · · · · · · ·		
		· · · ·
		L
		·
		I
Thread: cos(5398)		
Thread: sin(5397)	··	· · ·
filledd: Sill(5557)		L !:
	· · ·	·
User Events:	1::	
User Events.		
L	·····	
		0.15
· 0.22000/		
cycle = 0.220099	9 ms	
	···	
:		0.1s , 30.1s , 60.1s ,
		······································
Comment Trees		Hover time from current timeline = 0.000768884
Current Time	65.934337660	Hover time from current timeline = 0.000768884
Start Time	0.000000000 · ·	· · · ·
End Time	94.642514749	
		Current offset=3925 id=cycle_start proc=app thr=sin time(sec)=65.934337660
Span	94.642514749	(0.00000000 from current time)
		(or so the second mean cancer
		·
		······································
•		
•		

Figure 4-22. Adding a Data Graph

- Click inside data graph you just inserted.
- Drag the top border to the top of the graph container and the bottom border to the bottom of the graph container so that the data graph fills the graph container you created.
- Click and drag the upper and lower lines of the newly inserted data graph to fill the available space.
- Double-click in the middle of the data graph.

	Edit Data Graph Profile		
Key / Value	Condition Reset Choose Profile		
			•
Events	ALL	Browse	\searrow
Exclude Events	NONE	Browse	
Condition	TRUE		
Processes	ALL	Browse	
Threads	ALL	Browse	
CPUs	all (mask=all)		
Value	NONE		
Min Value	CALC		
Max Value	CALC		
	Drawing and Coloring Options		
	OK Can	cel He	lp

The Edit Data Graph Profile dialog is presented.

Figure 4-23. Edit Data Graph Profile Dialog

- Enter:

cycle_end

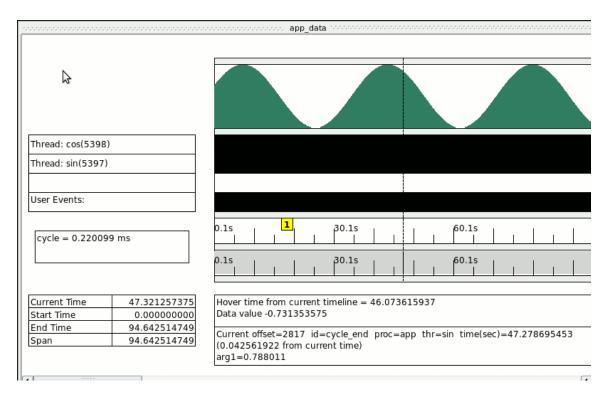
in the Events text field.

- Enter:

arg1_dbl

in the Value text field.

- Press OK to close the Edit Data Graph Profile dialog.
- Right-click inside the data graph and select Adjust Colors in Selected from the pop-up menu and select Data Graph Value Color... from the sub-menu.
- Select a pleasing color from the Select color dialog for the data graph. Click OK to close the Select color dialog.
- Right-click anywhere in the display panel labeled app_data and select Edit Mode from the pop-up menu or press Ctrl-E to return to *view mode*.



- Zoom the display to see the sine wave generated by the program.

Figure 4-24. Display Page with Data Graph

Kernel Tracing

Kernel tracing provides amazing insight into the activities of the system and how applications interact with each other and the kernel.

In order to use kernel tracing you must be running a trace-enabled kernel.

Kernels names ending in **-trace** and **-debug** have kernel tracing enabled. You may check to see which kernel is running by using the following command:

uname -r

If you are not running a trace-enabled kernel, reboot now and select it from the GRUB menu at boot time. If you are unable to reboot your system at this time, please follow the tutorial and load the pre-recorded kernel data as instructed.

- Click on the first tab of the NightTrace main window.
- Ensure the user daemon is stopped by pressing the Halt button in the Daemon Control Area if it is sensitized.
- Select the app_data segment in the Trace Segments panel.

CAUTION

If the trace segment was not removed it is likely that you selected the app_data line from the Daemon Definition Area and not the Trace Segments panel.

- Press the Close Trace Data button in the Trace Segments panel.

NightTrace will pop up a dialog warning you that the trace data has not been saved and will be discarded; the data does not need to be saved for this tutorial.

Obtaining Kernel Trace Data

If you are not running a trace-enabled kernel, skip this section and refer to the section "Using Prerecorded Kernel Data" on page 4-34.

🕫 Edit Daemon Definiti	on	
General Settings	Enabled Events	
Name kernel_daemon RCIM Clock	State 🔻 Code 🛛 Name	
Target Zoey User jojo	Disabled 4100 4100	
	Disabled 4101 4101	
Output 🔿 File 🖲 Stream 🔿 Consumer	Disabled 4102 4102	
	Disabled 4103 4103	
- Stream Settings	Disabled 4104 4104	
Stream Buffer Size (bytes) 8388608	Disabled 4105 4105	
Trees Buffer Cettings	Disabled 4106 4106	
Trace Buffer Settings	Disabled 4107 4107	
🗌 Buffer Wrap	Disabled 4108 4108	
Specify Non-Default Number Buffers	Disabled 4109 4109	
Specify Non-Default Buffer Size (bytes)	Disabled 4110 4110	
Trace CPU List all (mask=all)	Disabled 4111 4111	
	Disabled 4112 4112	
Trace Daemon Runtime Settings	Disabled 4113 4113	
Policy First in First Out 💌	Disabled 4114 4114	
Priority 50	Enabled 4115 EVENT_LOST	
	Disabled 4116 4116	
CPU List all (mask=all)	Disabled 4117 4117	 ▲ ▼
	OK Reset Cancel	Help

- Double-click on the kernel_trace_to_gui entry in the Daemons panel on the first page of the NightTrace main window.

Figure 4-25. Edit Daemon Definition Dialog

- Check the Buffer Wrap checkbox in the Trace Buffer Settings section of the Edit Daemon Definition dialog.
- Press OK.

The kernel daemon is now configured to run in bufferwrap mode. This means that kernel events are collected in kernel memory buffers and are not passed to NightTrace except by explicit flush operations.

Depending on system activity, huge amounts of kernel trace data can be generated in a relatively short period of time. Since operation of NightTrace is likely a new experience for many users, we will restrict the data flow to a manageable size for new users.

- Ensure that kernel_trace_to_gui is selected in the Daemon Control Area.
- Press the Launch button.
- Press the Resume button.

- Watch the daemon statistics in the Daemon Control Area; once 30,000-50,000 events are present in the Buffer column, press the Flush button and then the Halt button.

Skip the next section and jump directly to "Analyzing Kernel Data" on page 4-34.

Using Prerecorded Kernel Data

This section is provided only for those using the tutorial that have not booted a trace-enabled kernel.

If you collected live kernel trace data in the preceding section, skip to "Analyzing Kernel Data" on page 4-34.

The NightStar RT **tutorial** directory contains some pre-recorded kernel data which can be used in the section titled "Analyzing Kernel Data" on page 4-34.

- Select the Open Trace File... menu item from the NightTrace menu in the NightTrace main window.
- Type the following into the file dialog in the Selection text field:

```
/usr/lib/NightStar/tutorial/.kernel-data
```

- Press the OK button.

Proceed to the next section.

Analyzing Kernel Data

NightTrace automatically generates a default kernel display page that is customized to the system from which the kernel data was captured.

- Click on the tab created in the NightTrace main window to display the newly-created kernel display page. The tab will have a name like <machine_name> Timeline.
- Zoom out until the data and state graphs are populated with events.

7	,		-	y <u>P</u> rofiles Ti <u>r</u>				*		•T	
	op_data	Profiles s	ine (0 to	50457) sine	Durations	raptor 7	Fimeline	-			
Offset		Event	<u> </u>	Process	1	Thread	Time (sec)	Tag	Description		····· (P
26509		IRQ_ENTRY	2	ntrace	2	11524	1524.480945653		Interrupt invalid	late_tlb (IRQ=1))
26510		IRQ_EXIT		ntrace	2	11524	1524.480947355		Interrupt handli	_ , , ,	
26511		TRAP_EXIT	3	ntrace	e	11745	1524.480948026		Exception han	dling for Page-	Faul
26512	Т	RAP_ENTRY	3	ntrace	2		1524.481219344		Exception Page	e-Fault occured	at p
Page-Fault gettimeofda ntrace IRQ_EXIT raptor CPU local_timer Page-Fault _newselect ntrace TRAP_EXIT	3 t	P									
Interrupt Syscall	Exception KernelEv			4.41s 152		1524.61s	s 1524.71s		4.81s 1524		
Current Tin Start Time			lover tim)ata valu	e from current tir e 0	neline = 0.00	06857584	1				
End Time	1525.03	38727479	urrento	ffset=26511_cpu	=3 id=TRAP		roc=ntrace thr=117	45 time(s	sec)=1524 48094	8026	_
End time	0.67	705004001	anento		-5 IG-TRAF	_can p	ise-made un=11/	-s ane(s	500,-1324.40034	0020	
Span	0.07	76590402	0.00000	0000 from curren	it time)						

- Click in an active area and zoom in until detail can be seen.

Figure 4-26. Kernel Display Page

NOTE

Your timelines may look significantly different if you have a different number of CPUs. Additional system activity can make the display vary as well. Do not be concerned about such differences at this step.

For each CPU, the following information is displayed:

- interrupt activity (in red)
- machine exception activity (in green)

- system call activity (in blue)
- per-process CPU utilization (shown in a variety of colors)
- detailed kernel events (in dark red)

The data boxes on the left hand side of the display page are color coded to match the information they describe. Their contents change dynamically based on the position of the current timeline.

- Press Ctrl+F to switch to the Profiles panel.
- Click the Reset button to the right of the Key/Value selection area.
- Press the Browse... button to the right of the Processes text field.

The Select Processes dialog is presented.

- Select the **app** process from the list of known processes.
- Press the Select button to close the Select Processes dialog.
- Select the System Call Enter Events option from the Key / Value option list.

The Select System Calls dialog is presented.

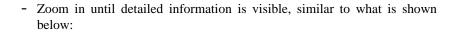
- Select nanosleep from the list of system calls shown.
- Press the Select button to close the Select System Calls dialog.
- Change the list of events in the Events text field to include only SYSCALL_RESUME.
- Press the Search Forward button.

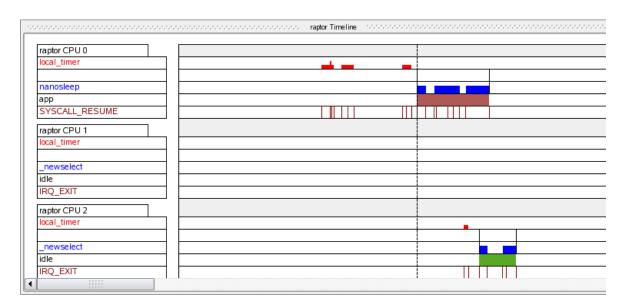
A new profile based on the information entered is added to the **Profile Status List** and the current timeline is changed to the next occurrence of a resumption of a suspended nanosleep system call in process **app**.

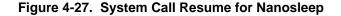
NOTE

If NightTrace fails to find an occurrence matching the sort criteria just entered, recheck the search criteria. It is likely that you may have skipped pressing the **Reset** button in the steps above. Ensure that the **Threads** text field indicates ALL and not sin.

- Click on the tab corresponding to the kernel display page.







NOTE

Your timeline may look significantly different if you have a different number of CPUs. Additional system activity can make the display vary as well. Repeat the search a few times to find an occurrence that looks similar to the row which indicates the **app** process. You can repeat the last search by pressing the forward search icon on the toolbar or by pressing the **Ctrl-G**.

The red bar to the left of the current timeline indicates that an interrupt occurred. In this case, it was a local_timer interrupt.

The tall vertical black line spanning the system call and exception rows represents a context switch. The current timeline (dashed line spanning the entire rectangular display area) is likely overlaid with the context switch line at this zoom setting.

- Select the highlighted event in the Events panel. This is the event at the current timeline, which should be SYSCALL_RESUME.

Offset	Event	CPU	Process	Thread	Time (sec)	Tag	Description
44075	KERNEL_TIMER	0	idle	0	1.723053551		
44076	IRQ_EXIT	0	idle	0	1.723056368		Interrupt handling for timer (IRQ=0) exited
44077	IRQ_ENTRY	0	idle	0	1.723077693		Interrupt local_timer (IRQ=1)
44078	PROCESS	0	idle	0	1.723079367		Wake process app (23540)
44079	IRQ_EXIT	0	idle	0	1.723081714		Interrupt handling for local_timer (IRQ=0) exited
44080	SCHEDCHANGE	0	app	23540	1.723084038		idle switched out (runnable); app (23540) switched in
44081	SYSCALL_RESUME	0	арр	23540	1.723084039		Resuming system call nanosleep
44082	SYSCALL_EXIT	0	app	23540	1.723087713		Exited system call nanosleep
44083	SYSCALL_ENTRY	0	app	23540	1.723091497		Entering system call ipc from pc=0x40
44084	IPC	0	app	23540	1.723092484		System V IPC call to semop (arg1=0xf800d)
44085	PROCESS	0	app	23540	1.723097665		Wake process app (23541)
44086	PROCESS	0	app	23540	1.723100008		Wake process app (23542)
44087	IRQ_ENTRY	3	idle	0	1.723101277		Interrupt reschedule (IRQ=1)

The **Description** column in the **Events** panel for the currently highlighted event describes the event in more detail with:

Figure 4-28. Events Panel after Search

- While the current timeline is at the SYSCALL_RESUME event, press the Up Arrow key.

The current timeline is changed to the preceding event and the text description indicates a context switch with text similar to the following:

idle switched out (runnable); app (5336) switched in

The blue bar represents system call activity. The data box to the left will describe the system call name for the system call at or to the left of the current time line.

- Press the Ctrl-G key to advance back to the SYSCALL_RESUME event.

In the instance shown in the screen shot above, shortly after the sine thread returns from nanosleep, the main thread is exiting the nanosleep call on line 93 of **app.c**. It then enters an ipc system call to execute the semop library call on line 94.

NOTE

On some systems, the system call may be described as semop instead of ipc.

Mixing Kernel and User Data

If you are not running a trace-enabled kernel, skip this section and proceed to "Using the NightTrace Analysis API" on page 4-42.

- Click on the first tab of the NightTrace main window.
- Ensure the kernel daemon is halted by pressing the Halt button if it is sensitized (it should have been halted in a previous step).

- Select the kernel_trace_to_gui segment in the Trace Segments panel and select the Close Trace Data menu option of the context menu.
- Select both daemons in the Daemon Control Area using Click and Shift+Click mouse and keyboard actions.
- Press the Launch button.

Read the next four steps before proceeding, then execute them in order.

- Press the Resume button.
- Wait about 2 seconds.
- Press the Flush button.
- Press the Halt button.

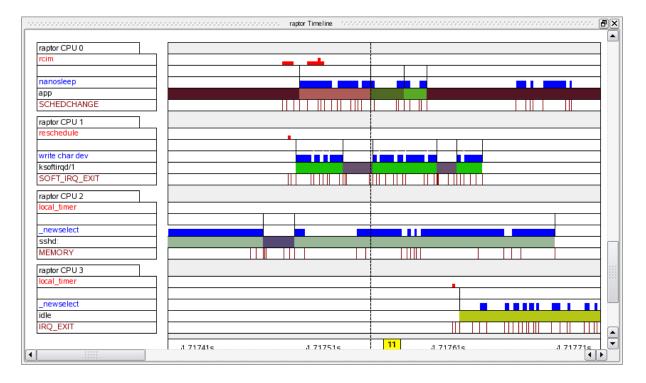
Data from both the user application and the kernel have been captured and brought into NightTrace.

- Click on the Profiles tab.
- Select the sine profile from the **Profile Status List** at the top of the page.
- Press the Summarize icon on the toolbar.

The last action caused a new page to be created containing a summary of the sine state defined in "Generating Summary Information" on page 4-23.

The current timeline is automatically positioned to the longest instance of the state.

- Click on the tab corresponding to the kernel display page.
- Zoom in or out as required until you can clearly see the detail relating to the sine thread's cycle.



In the graphic shown below, the sine thread was preempted by a kernel processing of a rcim interrupt.

Figure 4-29. Longest Instance of State

The reason for the extended cycle in your trace data may be due to other circumstances.

- Was the sine_thread() preempted by another process?
- Did an interrupt occur during the cycle?
- Was there significant activity on the hyper-threaded sibling CPU which stole cycles from the CPU where the sine thread was executing?
- Did the application get a page fault or other machine exception?
- Did activity on a hyper-threaded sibling CPU interfere with the CPU where app is executing.

Some of these circumstances are discussed in more detail in "Overrun Detection and System Tuning" on page 7-9.

Machine exceptions include information detailing the type of exception, the faulting address (when applicable), and the PC at which the exception occurred.

- Type Ctrl+F while the kernel display page is selected.
- Select Exception All Events from the Key / Value option list.
- Select Page-Fault from the list of exceptions.
- Press the Select button.

- Press the Search/Forward button.

If a page fault is located, the current timeline is moved to the next occurrence of a page fault. The text area at the top of the kernel display page includes detailed information about the exception, including the PC at which the fault occurred and the faulting address.

You can use NightView to see the actual line number of programs (if they have debugging information) based on the PC information with a command like: **list** **pc-address*.

Using the NightTrace Analysis API

NightTrace provides a powerful API which allows user applications to analyze pre-recorded trace data or to monitor and analyze live trace data.

Users can write programs that define states and conditions and process events as they occur.

In this tutorial, we will instruct NightTrace to build an API program automatically.

- Click on the Profiles tab.
- Select the sine profile from the Profile Status List.
- Select the Export to API Source... menu item from the Profiles menu.

The following dialog is displayed:

Export Profile(s) to NightTrace API Source File					
■ Define main() function ■ State start callbacks					
Define callback functions State end callbacks					
E Default printf()'s in callbacks					
Report analysis API errors State inactive callbacks					
🕱 Read trace data from stdin					
Trace Data File Stdin]				
Profiles Source export_analysis_0.c]				
Callbacks Source export_analysis_0.c]				
Export Reset Cancel Help					

Figure 4-30. Export Profiles to NightTrace API Source File dialog

- Clear the State start callbacks checkbox.
- Press the Export button.
- Select the Exit Immediately menu item from the NightTrace menu to exit NightTrace.

NightTrace has created an API program which listens for occurrences of the state defined by the **sine** profile and prints out some information for each instance.

- Build the API program using the following command:

```
cc -g export_analysis_0.c -lntrace_analysis
```

This program expects to consume live trace data.

You can configure a user daemon with the NightTrace GUI and have NightTrace launch the analysis program automatically.

Alternatively, you can use the command line user daemon program **ntraceud** to achieve the same effect.

- Type the following command:

```
ntraceud --stream --join /tmp/data | ./a.out
```

This command instructs **ntraceud** to start capturing trace data from a running application which is using the file /tmp/data as a handle. The --stream option indicates that instead of logging the data to the named file, it should be sent to **stdout**.

The application program may not immediately begin generating output because the data rate is fairly low and buffering is involved.

- To flush the current buffers for immediate consumption by the application, issue the following command in a different terminal session:

ntraceud --flush /tmp/data

NOTE

You may need to repeat that command several times over a period of a few seconds to allow the data to pass through system buffers.

Data similar to the following will appear on **stdout** in the terminal session where the analysis program was launched:

sine (end)offset 665 occur 333 code 2 pid 3399 time 16.628649 duration 0.00003 sine (end)offset 667 occur 334 code 2 pid 3399 time 16.678631 duration 0.000003 sine (end)offset 669 occur 335 code 2 pid 3399 time 16.728655 duration 0.000003 sine (end)offset 671 occur 336 code 2 pid 3399 time 16.728656 duration 0.000003 sine (end)offset 673 occur 337 code 2 pid 3399 time 16.828693 duration 0.000003 sine (end)offset 675 occur 338 code 2 pid 3399 time 16.828693 duration 0.000004 sine (end)offset 677 occur 339 code 2 pid 3399 time 16.878716 duration 0.000003 sine (end)offset 679 occur 340 code 2 pid 3399 time 16.978760 duration 0.000003 sine (end)offset 681 occur 341 code 2 pid 3399 time 17.028779 duration 0.000003

- Issue the following command to terminate the daemon:

```
ntraceud --quit-now /tmp/data
```

If you are not running a trace-enabled kernel daemon, skip the remaining of this section and proceed to "Conclusion - NightTrace" on page 4-69.

Several sample API programs are provided with NightTrace.

- Type the following commands to build the watchdog example program:

cp /usr/lib/NightTrace/examples/watchdog.c .
cc -g -o watchdog watchdog.c -lntrace_analysis

This simple sample program watches for context switches on a specific CPU and prints the name of the process that is switching in.

This time the **ntracekd** kernel daemon will be used to capture 5 seconds of kernel data and stream the output to the **watchdog** program.

- Issue the following command:

```
ntracekd --stream --wait=5 /tmp/x | ./watchdog 1
```

The program will generate output similar to the following:

```
context switch: 4.979350027
                               4 ksoftirgd/0
context switch: 4.979358275 2846 X
context switch: 4.983906074
                               0 idle
context switch: 4.983960385 2846 X
context switch: 4.994892976 3167 firefox-bin
context switch: 4.994989171
                            4492 ntfilterl
context switch: 4.995070736 4489 watchdog
context switch: 4.995092415 4492 ntfilterl
context switch: 4.995173214 4489 watchdog
context switch: 4.995188096 4492 ntfilterl
context switch: 4.995256175 4489 watchdog
context switch: 4.995270824 4492 ntfilterl
context switch: 4.995332743 4489 watchdog
context switch: 4.995355783 2846 X
context switch: 5.000351519
                            4 ksoftirqd/0
context switch: 5.000360675 2846 X
```

Automatically Tracing Your Application

This section will utilize a new invocation of the NightTrace analysis tool.

- If you still have a NightTrace session active, exit NightTrace by selecting Exit NightTrace Immediately from the File menu.

NightTrace provides a component called Application Illumination, which automatically instruments your application with trace points that record the entry and exit of subprograms.

The arguments and return values to those subprogram calls, among other things, can be included as part of the trace data, so that you can see them when you analyze the data.

Not all subprograms can be automatically instrumented. Application Illumination cannot detect functions which do not have globally visible external symbol names (e.g. static void func(); in the C programming language). Similarly, it cannot detect functions which are completely internal to a linked shared library (i.e. functions that have no external entry point). Similarly, by default, Application Illumination only operates on functions which have compiler-generated debug information -- although you can change this behavior.

The utility **/usr/bin/nlight** is the primary interface used to instrument your application.

nlight provides for selection and exclusion of subprograms as well as customization of detail levels.

In this tutorial, we'll use **nlight**'s wizard to quickly and easily instrument the **app** program we've been using thus far.

nlight Wizard - Selecting Programs

- While positioned in the tutorial test directory you created in the initial stages of this tutorial, invoke the **nlight** tool:

nlight &

The following window is displayed.

	NightLight - New Session
<u>F</u> ile <u>V</u> iew <u>T</u> ools <u>H</u> elp	
Manager Wizard	
 Select Programs Define Illuminators Select Illuminators 	Select Programs with Debug Information One or more programs may be instrumented with trace points at function calls. By building the executable file with debug information, function retums may also be instrumented, and information about function arguments, return values, and global variables may be recorded as arguments to the events.
 Relink Programs 	-
 Activate Illuminators 	Browse Delete
🔿 Run Scripts	NightLight will use the Build Command to build any missing programs. The Build and Build All buttons may be used to build the current program or all programs respectively at any time. Build Command:
	Build All
	As an advanced feature, the Manager may be used to identify object files, archives, shared objects, and programs, and to create illuminators for them.
	Advanced Prev Next Help

Figure 4-31. nlight Wizard - Select Programs Step

The Wizard tab is raised by default and provides step-wise instructions for instrumenting your application.

The bullet list on the left side of the page indicates what step you're currently working on within the wizard, while the Prev and Next buttons at the bottom navigate through the steps.

The initial step is Select Program, in which we tell **nlight** which program to illuminate.

- Press the Browse... button and select the **app** program file from the file selection dialog, then press Save to close the file selection dialog.

Note that the Build Command text area below the program selection now contains a default **make** command. While not specifically required, it is convenient to provide **nlight** a command which can rebuild your original program, in case you should choose to do so from within **nlight**. Further, **nlight** will automatically invoke this command if it finds that the specified program file does not exist.

- Press the Next button to proceed to the next step.

nlight Wizard - Defining Illuminators

The **Define Illuminators** step is displayed, which allows us to select the portions of code in the application that we want to illuminate.

	NightLight - New Session	
<u>F</u> ile <u>V</u> iew <u>T</u> ools <u>H</u> elp		
Manager Wizard		
 Select Programs Define Illuminators Select Illuminators Relink Programs Activate Illuminators Run Scripts 	Define an Illuminator for each Program An illuminator is a directory containing object code to record trace events for functions in the stati portion of each program, descriptions of those events for NightTrace, and various other files. An illuble created for each program and will be called programName.ai. Program: app Image: Control of each program and will be called from programName.ai. Program: app Image: Control of each program and will be called from being traced by matching their names against regulations may be included or excluded from being traced by matching their names against regulations. The inclusions and exclusions in the list below are applied in order from top to bottod default, all functions are included except those beginning with underscore, those in C++ std namain, and Ada's internal I/O routines. Functions Included or Excluded from Being Traced:	uminator may vilar om. By
Creating Illuminator Done	As advanced features: (1) the Editor may be used to customize the user-defined illuminator, (2) th may be used to customize additional illuminators, including the predefined ones, (3) to assist with advanced customizations, the user-defined illuminator may be populated with all functions and glu found in the program, and (4) a detailed report about the user-defined illuminator may be written Console. Advanced Build Prev Next	n doing obal variables

Figure 4-32. nlight Wizard - Define Illuminators Step

The term *illuminator* refers to a directory which contains the **nlight**-generated files required for instrumenting code. Normally, you don't interact directly with the contents of that directory; **nlight** does all the work. The Define an illuminator for this program checkbox tells **nlight** that we want to instrument the statically-linked portions of the **app** program.

This page also includes a selection and exclusion area which allows you to specify specific subprograms you want to include or exclude from instrumentation. You can also specify patterns via regular expressions to include or exclude multiple functions easily.

We'll just let **nlight** illuminate all the statically-linked portions of our **app** program at this step.

- Ensure the checkbox labeled Define an illuminator for this program is checked.
- Press the Next button to proceed to the next step.

nlight Wizard - Selecting Illuminators

	NightLight - New Session
<u>F</u> ile <u>V</u> iew <u>T</u> ools <u>H</u> elp	
Manager Wizard	
Manager Wizard Select Programs Define Illuminators Select Illuminators Relink Programs Activate Illuminators Run Scripts	Select Predefined Illuminators for each Program Some predefined Illuminators are provided with NightTrace and may be linked into each program. Program: app The main Illuminator initiates tracing with a trace_begin() call before main() begins running. Programs that already initiate tracing on their own should not include this illuminator. If main These Illuminators trace calls to functions in the corresponding shared system libraries. glibc glibc ccur_rt
	customize the predefined ones. Glibc's debuginfo package(s) must be installed to customize glibc and pthread. Advanced Prev Next Help

The Select Illuminators step is now displayed.

Figure 4-33. nlight Wizard - Select Illuminators Step

This step allows us to select additional, predefined illuminators for our program.

The **main** illuminator is special and is only needed if your application doesn't already use the NightTrace API. Our **app** program already does, so we should clear this checkbox.

- Clear the main checkbox.

Additional illuminators are already built and shipped with NightTrace. In the middle section of the page, we can include illuminators for system libraries that our program uses.

- Check the glibc checkbox to include the glibc illuminator.
- Check the pthread checkbox to include the pthread illuminator.
- Press the Next button to proceed to the next step.

nlight Wizard - Relinking the Program

The Relink Programs step is now displayed.

	NightLight - New Session 📃 🗆 🗙
<u>F</u> ile <u>V</u> iew <u>T</u> ools <u>H</u> elp	
Manager Wizard	
Select Programs Define Illuminators	Relink Illuminated Programs Illuminators have object files that must be linked with programs along with <i>libntrace</i> . Each program is relinked with these files and library as a separate executable file. The illuminators are initially not activated. Unactivated illuminators have zero run-time overhead. Program: app
 Relink Programs 	By default, the copy of the program with the illuminators and <i>libntrace</i> linked in is named <i>originalName</i> AI.
 Activate Illuminators 	Illuminated Program Path: appAl Browse
O Run Scripts	The command to relink the program with illuminators may be specified using some substitution variables (<i>*keyword</i>) for the illuminated program path, the options that must be passed to the compiler, and the dependency list. Click on the View buttons for further assistance.
	Relink Command: View Typical Makefile Target View Substitution Variables
	make %RELINK ILLUMINATOR_OPTIONS="%GCC" ILLUMINATORS="%AI"
	Default Make Default a.link Relink All
	There are no additional advanced features available on the Manager, but it may be used to make the same settings.
	Advanced Prev Next Help

Figure 4-34. nlight Wizard - Relink Programs Step

In order to utilize the illuminators, we need to create a new version of our executable program which links with exactly the same objects and libraries as the original program, but also includes the **nlight**-generated illuminator files.

The resultant executable will contain the unmodified object files and libraries from the original program, but it will also include instrumented "wrapper" functions which inject the actual trace event calls at runtime.

Since we need to essentially recreate the original program and add some new link options, the wizard needs you to enter a command that will do this. The default "relink" command is already filled in and assumes you will use the **make** utility to build the program. It passes some **make** parameters which make it very easy for you to form the **Makefile** rule to build the new program.

In most cases, you can simply copy the final rule required to create your original application and rename it and add the options passed by the wizard on the link line.

Our **Makefile** in the tutorial test directory already has a rule defined for the instrumented program name, which, by convention, is the original name of the program with the letters "AI" appended to it. The following is an excerpt from the **Makefile** that shows the rules to build **app** and **appAI**.

```
app: app.c
    cc -g -o app app.c \
        -lntrace_thr -lpthread -lm
appAI: app.c $(ILLUMINATORS)
    cc -g -o appAI app.c \
        $(ILLUMINATOR_OPTIONS) -lntrace_thr -lpthread -lm
```

Notice that the rule to build **appAI** (the instrumented version of the program) is exactly the same as the rule to build the original **app** program, except that we also include the options passed in by the wizard in the "relink" command.

Note that we also included a dependency on the illuminator itself (**\$ILLUMINTATORS**), so that the **make** command will definitely relink the **appAI** executable if we make custom changes to the illuminator, even if no other changes are made to your application.

- Press the Next button to proceed to the next step.

nlight Wizard - Activating Illuminators

	NightLight - app.nl 📃 🗆 🗙
<u>F</u> ile <u>V</u> iew <u>T</u> ools <u>H</u> elp	
Manager Wizard	
Manager Wizard Select Programs Define Illuminators Relink Programs Activate Illuminators Run Scripts	Activate Illuminators in each Program Use the check box to activate or deactivate the illuminators linked into each program. Deactivated illuminators have zero execution-time overhead. Options may be specified for each illuminator. Program: app Detail Level controls how much detail is recorded as arguments to events. The gilice illuminator traces function calls to the system C library. If gilic Detail Level: 2 • The pthread illuminator traces function calls to the POSIX threads library. If pthread Detail Level: 2 • This illuminator is the user-defined illuminator for the current program. If app.ai Detail Level: 2 • As an advanced feature, the Manager may be used to configure multiple activation sets, set additional options, and advanced feature, the Manager may be used to configure multiple activation sets, set additional options, and select a different default activation set (if no default activation set existed, the wizard created one called <i>vizard</i> .

The Activate Illuminators step is now displayed.

Figure 4-35. nlight Wizard - Activate Illuminators Step

An important feature of Application Illumination is that once you relink your program and include the illuminators, the illuminators are inert. You can run your application with zero overhead while the illuminators are inert.

In this step, we'll activate them so that when we run the program trace data will be logged.

The default activation level is 2, which provides a medium amount of detail with each event. In this tutorial we want to see more detail, so we'll increase the detail level of each illuminator.

- Change the Detail Level for the glibc illuminator to 3.
- Change the Detail Level for the pthread illuminator to 3.
- Change the Detail Level for the appAI illuminator to 3.
- Press the Next button to proceed to the next step.

nlight Wizard - Running the Program

The Run Scripts step is now displayed.

The wizard provides this step for convenience, but in reality, you're just ready to run your instrumented program right now outside of the wizard.

We'll go ahead and use the wizard to invoke NightTrace and the instrumented program for demonstration purposes.

-	Change the Mode to Stream n	node by selecting	Stream	from the option
	list.			

	NightLight - app.nl	X
<u>F</u> ile <u>V</u> iew <u>T</u> ools <u>H</u> elp		
Manager Wizard		
Select Programs Define Illuminators Select Illuminators Relink Programs	Run Scripts to Launch Programs and NightTrace File mode provides a single script to launch the programs to collect data in a trace f NightTrace to analyze the file. Stream mode provides two separate scripts: one to launch NightTrace in streaming launch the programs separately, so they can send their trace data directly to Night Mode: Stream	mode and the other to
 Activate Illuminators 	Script to Launch NightTrace in Stream Mode:	
Run Scripts	# Invoke NightTrace importing daemons from the programs ntraceimport=appAI & # Wait a few seconds for messages from ntrace to go to console	Run Default
	Script to Launch Programs:	
	# Launch Programs	Run
	./appAI	Default
		Terminal Session:
	As an advanced feature, the Manager may be used to create an unlimited number of wizard's scripts are Wizard, Wizard Stream, and Wizard Launch.	·
Activating Illuminator Done	Advanced Prev	Next Help

Figure 4-36. nlight Wizard - Run Scripts Step (with Stream Mode selected)

In Stream mode, nlight presents two scripts with example commands.

The first script launches NightTrace so that you can start a streaming daemon and immediately collect trace data from your program when you launch it.

The second script simply launches the instrumented program.

- Press the Run button to the right of the first script, entitled Script to launch NightTrace in Stream Mode.

Analyzing Application Illumination Events

1			nt	race								X
<u>F</u> ile <u>V</u> iew	<u>D</u> aemons	Sea <u>r</u> ch	S <u>u</u> mmary	<u>P</u> rofi	les	Ti <u>m</u> el	ines	<u>T</u> ools	<u>H</u> e	lp		
6	1 1	≩ ≓	F		+	P	Þ	P	Σ		© ∥	»
Interval : 1	events (0 to	PI	Import F ease specify OK	y trace	file r Can	cel		TO 0 00	040	0000)	Curren	

The NightTrace analysis interface appears.

Figure 4-37. NightTrace - Import File Name

Since NightTrace was invoked with the **--import** option, it prompts you for the name of the trace data file, which is the first parameter your program passed to the trace_begin call.

- Enter /tmp/data in the prompt dialog and press OK.

NOTE

If the main illuminator had been selected in **nlight**, **ntrace** would have already known the name of the trace file. In our example, we didn't include the main illuminator, because our program already initiated tracing independently of **nlight**.

The Daemons panel now includes a user daemon which is ready to collect trace points from our instrumented **appAI** program.

1			Night	Trace - N	lew Sessi	on(Unsave	d)			
<u>F</u> ile <u>V</u> iew <u>D</u>	aemons Sea <u>r</u> o	ch S <u>u</u> mn	nary <u>P</u> rofi	les Ti <u>m</u> e	lines <u>T</u> ools	<u>H</u> elp				
	🎏 👼 =	• 🔊 I	🕲 🔎 j	•	<i>P P</i>	Σ	Ů ₩	H H	101 abc 🖸	0
					Daemons					
Type Daem			Target	: L	ogged	Lost	Sta		Attached	Buffer
—U— /tmp/d	ata_import		nar	í i i			Halt	ed		
						D'- I				
U <u>L</u> aunch	<u>R</u> esume	<u>P</u> au	se <u>H</u>	aic	Flush	<u>D</u> isplay	<u>T</u> riggers	•	<u>E</u> nable Ev	ents <u>D</u> elete
				Τ	race Segmer	its more services				
Type 🔻 Tr	ace Segment		Target	Logged	Los	st Duration	(sec) Unsa	ved		
	j		3	55			•			
							Sa	ve Trace [Data	Close Trace Data

Figure 4-38. NightTrace - Daemon Ready to Launch

- Press the Launch button to launch the daemon.
- Press the **Resume** button to start collecting trace events.

Now we're ready to run our instrumented application and collect trace data as the application executes.

- In the **nlight** wizard, press the Run button next to the script entitled Script to Launch Programs.

Returning to the NightTrace window, you can see that the user daemon is collecting events as the number in the Buffer column in the Daemons panel is steadily increasing.

8							Nigh	tTra	ce - N	lew 9	Sessi	on(Ur	nsave	d)					_	
<u>F</u> ile	<u>V</u> iew	<u>D</u> aemon	s Se	a <u>r</u> ch	Sum	nmary	<u>P</u> rof	files	Ti <u>m</u> e	lines	<u>T</u> ools	<u>H</u> elp	D							
B	7	ê 🎾	\$	≠		B		Þ	P	Þ	P	Σ		() () () ()	u -) 1	<u>*</u> F	L 101 abc	T		
Trac		mp/data_i																		
nononon.										Dae	mons					.*.*.*.*.*.*.				* F×
Тур	e Da	emon					Targe	et	L	ogge	d		Lost		State		Attach	ed	Buffe	
— U	_ /tm	p/data_im	port				na	rf			0		0	Lo	gging			3	1107	1
	<u> L</u> aun	ch 🕩	<u>R</u> esun	ne	II <u>P</u> a	ause	[•]	<u>H</u> alt		<u>F</u> lus	sh	<u>D</u> ispla	ау	<u>T</u> rigge	ers		<u>E</u> nable	e Even	ts Del	ete
			_			_			<u>ا</u>							I (_		
10000									~~ Т	race S	egmen	ts 🕬								te ex
Ту	pe 🔻	Trace S	egme	nt		Tar	get	Lo	gged		Los	st Du	ration	(sec) U	nsave	d				
															Save	Trace	Data	C	lose Trace D	ata
																				//

Figure 4-39. NightTrace - Daemon Collection Events

It is likely that the Attached count in the Daemons panel will indicate three attachments, since we left the **app** program running earlier in this chapter. The daemon itself is included in the count of attached processes; thus our two applications **app**, **appAI** and the daemon bring us to a total of three processes that are associated with the logging session.

- Press the Halt button in the Daemons panel to stop the daemon.
- Click on the /tmp/data_import tab to bring the Events and Timeline panels to the top of the NightTrace window.

The NightTrace display now includes a timeline which describes the applications events graphically as well as an events panel which lists the events in chronological order.

			NightTr	race - I	New Sessio	n(Unsa	ved)					
le <u>V</u> iew <u>D</u> aem	ons Sea <u>r</u> ch	S <u>u</u> mmar	ry <u>P</u> rofile	s Ti <u>m</u> e	elines <u>T</u> ools	<u>H</u> elp						
> 🛛 🏋) 🗢 🗳	<i>(</i> 7	i 🕼 🔎	• 🗩	2 کر	Σ	ڻ ا	*_	L L	101 abc	Œ	
ace /tmp/dat	a_import					· _						
					ere Events ree							 B
Offset	Event	CPU	Proc	ess	Thread	Ti	me (sec)	Tag	Dese	ription		
319	3	;		app	COS	5.35	0346771		arg1	=-0.6225	515	
320	1			арр	sin		0358028			=0.7880		
321	2			арр	sin	5.35	0392961		arg1	.=0.782	608	
												▲ ►
	nation											
vents												
Current Time 5.	nation 350392961 00000000	0.1s			3.1s		6.	15			9.1s	
Current Time 5. Start Time 0.	350392961											
Current Time 5. Start Time 0. End Time 10	350392961 000000000	0.1s			3.1s			1s .1s			9.1s	
Start Time 0. End Time 10	350392961 000000000 0.79361821					· · · ·						

Figure 4-40. NightTrace - /tmp/data_import Tab

In the following screen shot, the Timeline panel has been closed and the Events panel has
some columns hidden (use the context menu to select columns to show or hide).

View Daemons Search	n S <u>u</u> mmary <u>P</u> rofi	les Timeline	es Tools Help		
/ 📮 🖗 🍄 🗳 =		२ 🔉 🔍	الله 🍳 🍯	· " ★ 표 표 101	•I
e /tmp/data import					
			Events popposoo		
Event	Process	Thread	Time (sec)	Tag Description	
NT RESUME STREAM	983	983	0.000000000	- -	
3	арр	COS	0.042854292	arg1=0.250380	
1	app	sin	0.042867890	arg1=0.965926	
2	арр	sin	0.042902765	arg1=0.968148	
3	арр	COS	0.092940456	arg1=0.241922	
1	app	sin	0.092953147	arg1=0.968148	
2	app	sin	0.092987573	arg1=0.970296	
3	app	COS	0.143032892	arg1=0.233445	
1	арр	sin	0.143045809	arg1=0.970296	
2	app	sin	0.143080064	arg1=0.972370	
3	app	COS	0.193112354	arg1=0.224951	
1	app	sin	0.193124737	arg1=0.972370	
2	app	sin	0.193158589	arg1=0.974370	
3	app	COS	0.243207216	arg1=0.216440	
1	арр	sin	0.243219777	arg1=0.974370	
2	арр	sin	0.243254511	arg1=0.976296	
3	арр	cos	0.293320072	arg1=0.207912	
1	app	sin	0.293334222	arg1=0.976296	
2	арр	sin	0.293377943	arg1=0.978148	
3	арр	cos	0.343321310	arg1=0.199368	
1	арр	sin	0.343334694	arg1=0.978148	
2	арр	sin	0.343370519	arg1=0.979925	
3	арр	cos	0.393390454	arg1=0.190809	
1	арр	sin	0.393403042	arg1=0.979925	
2	арр	sin	0.393437486	arg1=0.981627	
3	арр	cos	0.443456007	arg1=0.182236	
1	app	sin	0.443468932	arg1=0.981627	

Figure 4-41. NightTrace - Customized Events Panel

- Close the **Timeline** panel by selecting the upper-right control box in the panel.
- Activate the Textual Search dialog by pressing Ctrl+T while the focus is in the Events panel.

The Search Events for Text dialog is shown.

😼 Se	earch Events for Text	
🗌 Treat search text as Search text 💫	s regular expression	
ENTER_work		_
Event attributes to ma	atch against:	
CPU 🕱	Process 🕱	
Description 🕱	Thread 🕱	
Event Name 🕱	Time 🕱	
Search Bac <u>k</u> ward		Close

Figure 4-42. NightTrace - Search Events for Text

- Type ENTER_work into the text field and press the Search Forward button.
- Press the Close button.

The Events panel now has the first occurrence of the ENTER_work event selected. In the figure below, the RETURN_work event was also selected manually.

ace /tmp	p/data_import	Summary Profiles			*	H 101 T	
Offset 0 N 1	p/data_import	·		Σ υ	₩	H 101 abc T	
Offset 0 N 1	Event		www.event				
Offset 0 N 1	Event		concentration Event				
0 N 1				s			
0 N 1		CPU Proces	1	1		scription	_
1	NT RESUME STREAM						_
-	NT ASSOC PID					L=0x41707061 arg2=0x49	
	NT ASSOC TID					L=0x6e69616d	
3	ENTER semget					ng semget(key=0,nsems=1,semf	a
4	RETURN semget			4.276973124		rning from semget()=294918	er
5 EN	NTER pthread attr			a 4.276978079		ng pthread_attr_init(attr=0xbf9d1	10.
	ETURN pthread at			4.276981550		rning from pthread_attr_init()=0	
7 EI	NTER pthread cre	appA	Al mair	1 4.276989146		ng pthread create(newthread=0x	bf.
8 RE	ETURN_pthread_cr	appA	Al mair	1 4.277034453	retu	rning from pthread_create()=0	
9 EN	NTER_pthread_attr	appA	Al mair	1 4.277035057	calli	ng pthread_attr_init(attr=0xbf9d1	10.
10 RE	ETURN_pthread_at	appA	Al mair	n 4.277035739	retu	rning from pthread_attr_init()=0	
11 E	NTER_pthread_cre	appA	Al mair	1 4.277036350	calli	ng pthread_create(newthread=0x	bf.
12	NT_ASSOC_TID	appA	Al sir	n 4.277036351	arg:	L=0x33303631 arg2=0x39	
13	NT_ASSOC_TID	1603	9 sir	1 4.277036351	. argi	L=0x33303631 arg2=0x39	
14	ENTER_sine_thread	appA	Al sir	1 4.277072322		ng sine_thread(ptr=0x80e3080)	
15	ENTER_work		Al sir	1 4.277073043		ing work(control=1) caller=	=0.
16	RETURN_work	app4	Al sir			rning from work() errno=0	
17	NT_ASSOC_TID	app/	Al sir	1 4.277103907		L=0x6e6973	
18	NT_ASSOC_TID	1603	9 sir	1 4.277103908	5	L=0x6e6973	
19	ENTER_semop	appA	Al sir	n 4.277121455	calli	ng semop(semid=294918,sops=0)x

Figure 4-43. NightTrace - Events Panel with Selected Events

These events correspond to the sin thread executing the work function.

NightTrace provides "tool-tip" displays of additional data when you hover the mouse cursor over various areas of the display. For the Events panel, the tool-tip text is dependent on which row and column you hover over.

- Place the mouse cursor over the description cell for the ENTER_work event and leave it there.

A tool-tip appears which contains the entire description of the event.

```
calling work(control=1)
    caller=0x80489e1 [sine_thread() at app.c:43]
    frame=0xb79413ac
Raw Arguments: 0x80489e1, 0xb79413ac, 0x1
```

Figure 4-44. NightTrace - Descriptive Tool Tip

Notice that the description includes the location of the caller using both the hexadecimal PC location as well as the name of the subprogram and file and line number information:

```
caller=0x80489e1 [sine_thread() at app.c:43]
```

NOTE

Depending on compiler versions and actual source line contents, the line number displayed may actually be associated with the next code-generating source line after the actual call. This is because the value of the PC that is included with the trace event is the "return address"; the instruction that will execute after the called function returns.

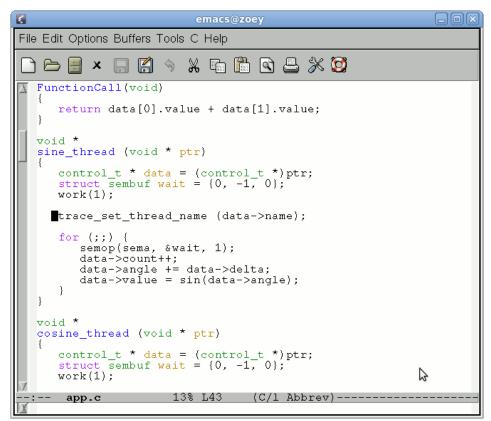
NightTrace will always attempt to map the PC address in the caller portion of the description to the subprogram and file/line descriptions, but it will not be able to provide this information if the corresponding routine wasn't built with debug information.

When a file and line number is available in an event's description, you can ask NightTrace to show you the source line in a text editor using the context menu.

Text Search	Ctrl+T
🕸 Sea <u>r</u> ch Forward	Ctrl+G
陰 Search Bac <u>k</u> ward	Ctrl+B
🗊 <u>G</u> oto	Ctrl+I
Distinguish Process Name by PID	
Edit Current Event Description	Ctrl+D
Close All Trace <u>D</u> ata	Alt+W
Show Source File from Description	
<u>D</u> isplay Fields	•

- Right-click the mouse on the description of the ENTER_pthread_create event and select the Show Source File From Description... option from the context menu.

NightTrace will load the source file and position your text editor at the appropriate line number, as shown in the following figure.





NOTE

NightTrace selects your editor via the EDITOR environment variable.

Summarizing Workload Performance

Remember that we summarized the workload performance of our threads in a previous section of this tutorial? We used trace points that we inserted via NightView and defined states for them.

We'll do the same basic thing here, but this time we'll just use the trace events that were automatically created for us by **nlight**.

The workload of the main thread is defined by the subprogram work. We'll use the entry and subsequent return of that subprogram to define a state which represents the work the main thread does each cycle.

- Select Change Summary Profile from the Summary menu.

le <u>V</u> ie	w <u>D</u> aemon	s Sea <u>r</u> ch	S <u>u</u> mmary	Profiles	Timelines	5 <u>l</u> ools	<u>H</u> elp						
7	3 🖇	😫 ≓	ז 🕼 🖏	S	\wp) 🔎	Σ	≣	©∥ ∥	*-		н	01 15c 🖸
pplicati	on Illuminati	on Trace	Profiles										
				Profi	le Status L			1					r" r" r" r" r" r
/pe Na	ime		Status	Co	unt	Last	Offset						
					ile Definitio								
	Key / Value	State		-	Reset)			Choose	e Profi	le		
1	Start Events	ALL										Browse	·
	End Events	ALL										Browse	·
Sta	art Condition	TRUE											
.	nd Condition	TRUE											
Er	Processes	ALL										Browse	·
Er												Browse	
Er	Threads	ALL										Browse	
			ightTrace/bin,	/state-sun	nmary.sh								
		/usr/lib/N	ightTrace/bin,	/ <mark>state-sun</mark> mask=all)									
	utput Script CPUs	/usr/lib/N											
	utput Script CPUs	/usr/lib/N all											
	utput Script CPUs	/usr/lib/N all state				Search	Forwar	d F		arch		Σ Sumr	

The Profile Definition panel appears.

Figure 4-46. NightTrace - Profile Definition Panel - State Mode

- Select State from the Key/Value option box to make the profile panel look the same as it does in the figure above.
- Press the Browse... button to the right of the Start Events row.

1	Select Events	
ENTER wcstof		
ENTER wcstof I		
ENTER_wcstoimax		
ENTER wcstok		
ENTER_wcstol		
ENTER_wcstol_l		
ENTER wcstold		
ENTER wcstold I		
ENTER_wcstoll_l		
ENTER wcstombs		
ENTER_wcstoq		
ENTER_wcstoul		
ENTER_wcstoul_l		
ENTER_wcstoull_l		
ENTER_wcstoumax		
ENTER_wcstouq		
ENTER_wcswidth		
ENTER_wcsxfrm		
ENTER_wcsxfrm_l		
ENTER_wctob		
ENTER_wctomb		
ENTER_wctrans		
ENTER_wctrans_l		
ENTER_wctype		
ENTER_wctype_l		
ENTER_wcwidth		
ENTER_wmemchr		
ENTER_wmemcmp		
ENTER_wmemcpy		
ENTER_wmemmove		
ENTER_wmempcpy		
ENTER_wmemset		
ENTER_wordexp		
ENTER_wordfree		
ENTER_work		
ENTER_wprintf		-
🗙 Search: work	📕 Next 👚 Previous 🗌 M	latch case
	Select Cancel	Help

The Select Events dialog appears.

Figure 4-47. NightTrace - Select Events Dialog with Search Active

For every function that has been illuminated, NightTrace assigns two events; ENTER_function_name and RETURN_function_name.

- Click in the Search text field and type work.

As you type the characters, the dialog automatically searches for an event whose name includes a sub-string of the characters you've typed. In the figure above, the first event found, ENTER_work, is selected.

NOTE

On your system, ENTER_work may not be the first event that matches the substring work. Press the Next button if required to advance to the correct event.

NOTE

You can use the Ctrl+F and Ctrl+G key sequences to initiate and repeat a search instead of using the mouse.

- Once the ENTER_work event is selected, press the Select button to choose this event and close the dialog.
- Repeat the procedure for the End Events selection using the Browse... button at the right, but select RETURN_work as the events of interest.

The **Profile Definition** panel now reflects our selection of the ENTER_work and RETURN_work events which define the start and end of our state.

1	Night	Trace - Ne	w Sessio	n(Uns	aved)) o x)
<u>F</u> ile <u>V</u> iew <u>D</u> aemons	Sea <u>r</u> ch S <u>u</u> mma	ry <u>P</u> rofiles	Ti <u>m</u> elines	<u>T</u> ools	<u>H</u> elp				
🖻 📮 🎽 🎽	s = 🔊 🛯	i 📨 🔎	P	P	Σ	∎ v [⊪]	*	H 101	Œ
Application Illumination		l							
									··· B×
Type Name	Statu	s Co	unt	Last	Offset				
	**********************************	Prot	file Definitio	n (2000)					··· B×
Key / Value	State	_	Reset	1		Choos	e Profile		
Start Events	ENTER work							Browse	1
End Events	RETURN_work							Browse	
Start Condition	TRUE								
End Condition	TRUE								
Processes 🛛	ALL							Browse)
Threads 🖌	ALL							Browse	
Output Script 📝	/usr/lib/NightTrace/	bin/state-sur	nmary.sh					Browse	
CPUs a	all	(mask=all)						
Name s	state								
Add	Apply 🎥	Search Bac <u>k</u> v	vard 🏾 😫	Sea <u>r</u> ch	Forward	Halt Se	earch	Σ Summar	i <u>z</u> e
									111

Figure 4-48. NightTrace - Profile Definition Panel with State Defined

- Press the Summarize button to summarize all occurrences of the defined state.

A new tab appears with the results of the summary.

ile View	Daemons S	earch Summar	v Profiles Time	elines Tools <u>H</u> elp			
ie <u>v</u> iew							
0) 🏷 🧔	≠ 🕼 🕅	河 🍳 🕅	🔎 🔎 Σ 🗐	o <mark>⊪ *</mark> + ⊭	H 101	1
ace /tm	p/data_impo	rt Profiles S	tate (0 to 8950)				
	******************		sector sector s	tate (0 to 8950) - 2000000			
linimum sta verage stat otal of state	ate duration: ate duration: e duration: durations: tate gaps fou	0.000001055 0.000193687 0.108851827	2 at offset: 8430 5 at offset: 7376				
laximum sta linimum sta werage stat otal of state	ate gap: e gap:	0.055917770 0.050014477 0.049970130 28.083212838					
linimum sta verage stat otal of state	ate gap: e gap: gaps:	0.050014477 0.049970130 28.083212838	at offset: 6596	Event	CPU Process	s Thread	Time (sec)
inimum sta verage stat otal of state	ate gap: e gap: gaps:	0.050014477 0.049970130	at offset: 6596 Gap (sec)				
inimum sta verage stat otal of state Offset V	e gap: e gap: gaps: End Offset	0.050014477 0.049970130 28.083212838 Duration (sec)	at offset: 6596 Gap (sec) 0.000000000	ENTER_work	appA	N main	7.751195628
inimum sta verage stat otal of state Offset ▼ 498	ate gap: e gap: gaps: End Offset 499	0.050014477 0.049970130 28.083212838 Duration (sec) 0.000149538	at offset: 6596 Gap (sec) 0.000000000 0.050023396	ENTER_work ENTER_work	appA appA	N main N main	7.751195628 7.801368562
inimum sta verage stat otal of state Offset V 498 513	e gap: e gap: gaps: End Offset 499 514	0.050014477 0.049970130 28.083212838 Duration (sec) 0.000149538 0.000344156	at offset: 6596 Gap (sec) 0.000000000 0.050023396	ENTER_work ENTER_work ENTER_work	appA	N main N main N main	7.751195628 7.801368562 7.851738129
offset ▼ 498 513 528	End Offset 499 514 529	0.050014477 0.049970130 28.083212838 Duration (sec) 0.000149538 0.000344156 0.000301928	Gap (sec) 0.00000000 0.050023396 0.050025411 0.050024248	ENTER_work ENTER_work ENTER_work ENTER_work	appA appA appA appA appA	N main N main N main	7.751195628 7.801368562 7.851738129 7.902064306
offset ▼ 498 513 528 543	End Offset 499 514 529 544	0.050014477 0.049970130 28.083212838 Duration (sec) 0.000149538 0.000344156 0.000301928 0.000355484	Gap (sec) 0.00000000 0.050023396 0.050025524	ENTER_work ENTER_work ENTER_work ENTER_work ENTER_work	appA appA appA appA appA appA	N main N main N main N main	7.751195628 7.801368562 7.851738129 7.902064306 7.952445313
offset ▼ 498 513 528 543 558	End Offset 499 499 514 529 544 559	0.050014477 0.049970130 28.083212838 Duration (sec) 0.000149538 0.000344156 0.000301928 0.000355484 0.000308063	Gap (sec) 0.000000000 0.050023396 0.050025411 0.05002524 0.050025524 0.050030661	ENTER_work ENTER_work ENTER_work ENTER_work ENTER_work ENTER_work	appA appA appA appA appA appA appA	N main N main N main N main N main	7.751195628 7.801368562 7.851738129 7.902064306 7.952445313 8.002784038
offset ▼ 498 513 528 543 558 573	End Offset 499 514 529 544 559 574	0.050014477 0.049970130 28.083212838 Duration (sec) 0.000149538 0.000344156 0.000301928 0.000355484 0.000308063 0.000132314	at offset: 6596 Gap (sec) 0.00000000 0.050023396 0.050025411 0.050024248 0.050025524 0.050030661 0.050030646	ENTER_work ENTER_work ENTER_work ENTER_work ENTER_work ENTER_work ENTER_work	appA appA appA appA appA appA appA appA	N main N main N main N main N main N main	7.751195628 7.801368562 7.851738129 7.902064306 7.952445313 8.002784038 8.052946998

Figure 4-49. NightTrace - Tab with Results of Summary

The tab includes summary statistics at the top, including minimum, maximum, and average duration, as well as a table of individual occurrences of each state at the bottom.

You can change the sort criteria of the table to find the longest state duration by clicking the column headers.

- Click the column header that says Duration (sec).

This causes the table to sort in descending order by duration; thus the occurrence of the function work with the longest duration is shown at the top.

<u>V</u> iew <u>D</u>	aemons Sea <u>r</u> c	h S <u>u</u> mmary <u>P</u> rof	files Ti <u>m</u> elines	<u>T</u> ools <u>H</u> elp			
Э 📮	🎏 🛱 =	1		ο 🔢 2 🔍	* <u>+</u> <u>+</u> <u>+</u>	101 abc 🖅	
ce /tmp	/data import	Profiles state (0	to 8950)				
			sononon state (0 t	o 8950)			
		0.000421722 at offs .000001055 at offs					
rage state		000193687	et. 7570				
al of state o		08851827					
mber of sta	te gaps found:	562					
		055917770 at offse	et: 3431				
kimum stat	tegap: 0.						
	2 1	055917770 at offse					
imum state erage state	egap: 0.0 gap: 0.04	050014477 at offse 49970130					
imum state erage state	egap: 0.0 gap: 0.04	050014477 at offse					
ximum stat nimum state erage state al of state g	egap: 0.0 gap: 0.04	050014477 at offse 49970130					
imum state rage state al of state <u>c</u>	e gap: 0.0 gap: 0.0 gaps: 28.0	050014477 at offse 49970130 33212838	t: 6596	Event	CPU Proce	ss Thread	Tim
imum state erage state al of state <u>o</u> Offset	e gap: 0.1 gap: 0.0 gaps: 28.0 End Offset [050014477 at offse 49970130 83212838 Duration (sec)	t: 6596 Gap (sec)	Event			
imum state rage state al of state o Offset 8429	e gap: 0.1 gap: 0.04 gaps: 28.05 End Offset 2 8430	050014477 at offse 49970130 83212838 Duration (sec) ▲ 0.000421722	t: 6596 Gap (sec) 0.050021760	Event ENTER_work	app	Al main	34.2349
imum stati arage state al of state o Offset 8429 5583	e gap: 0.1 gap: 0.0 gaps: 28.0 End Offset 8430 5584	050014477 at offse 49970130 83212838 Puration (sec) ▲ 0.000421722 0.000407610	Gap (sec) 0.050021760 0.050025376	Event ENTER_work ENTER_work	app app	AI main AI main	34.2349 24.7416
Offset 8429 5583 7825	e gap: 0.1 gap: 0.0 gaps: 28.0 End Offset 8430 5584 7826	050014477 at offse 49970130 33212838 0uration (sec) ▲ 0.000421722 0.000407610 0.000394759	t: 6596 Gap (sec) 0.050021760 0.050025376 0.050020842	Event ENTER_work ENTER_work ENTER_work	app app app	AI main AI main AI main	34.2349 24.7416 32.2232
offset 8429 5583	e gap: 0.1 gap: 0.0 gaps: 28.0 End Offset 8430 5584	050014477 at offse 49970130 83212838 Puration (sec) ▲ 0.000421722 0.000407610	Gap (sec) 0.050021760 0.050025376	Event ENTER_work ENTER_work	app app app	AI main AI main AI main	34.2349 24.7416 32.2232 29.1112
Offset 8429 5583 7825	e gap: 0.1 gap: 0.0 gaps: 28.0 End Offset 8430 5584 7826	050014477 at offse 49970130 33212838 0uration (sec) ▲ 0.000421722 0.000407610 0.000394759	t: 6596 Gap (sec) 0.050021760 0.050025376 0.050020842	Event ENTER_work ENTER_work ENTER_work	i app app app app app	AI main AI main AI main AI main	34.2349 24.7416 32.2232 29.1112
Offset 8429 5583 7825 6895	e gap: 0.1 gap: 0.0 gaps: 28.0 End Offset 2 8430 5584 7826 6896	050014477 at offse 49970130 83212838 Duration (sec) ▲ 0.000421722 0.000407610 0.000394759 0.000383091	t: 6596 Gap (sec) 0.050021760 0.050025376 0.050020842 0.050015314	Event ENTER_work ENTER_work ENTER_work ENTER_work	і і арр арр арр арр арр	Al main Al main Al main Al main Al main	34.2349 24.7416 32.2232 29.1112 35.4406
Offset84295583782568958789	e gap: 0.1 gap: 0.0 gaps: 28.0 End Offset 8430 5584 7826 6896 8790	050014477 at offse 49970130 83212838 Puration (sec) ▲ 0.000421722 0.000407610 0.000394759 0.000383091 0.000381447	Gap (sec) 0.050021760 0.050025376 0.050020842 0.050015314 0.050036568	Event ENTER_work ENTER_work ENTER_work ENTER_work ENTER_work	і і арр арр арр арр арр арр	Al main Al main Al main Al main Al main Al main	34.2349 24.7416 32.2232 29.1112 35.4406 22.1781
Offset 0 8429 5583 7825 6895 8789 4814	e gap: 0.1 gap: 0.0 gaps: 28.0 End Offset 8430 5584 7826 6896 8790 4815	0.0004477 at offse 0.000421722 0.000421722 0.000394759 0.000381447 0.000379642	t: 6596 Gap (sec) 0.050021760 0.050020842 0.050015314 0.050036568 0.050020176	Event ENTER_work ENTER_work ENTER_work ENTER_work ENTER_work ENTER_work	і арр арр арр арр арр арр арр	Al main Al main Al main Al main Al main Al main Al main	34.2349 24.7416 32.2232 29.1112 35.4406 22.1781 14.5305

Figure 4-50. NightTrace - Summary Results Sorted By Duration

- Double click on the first row in the table.

This causes the corresponding event to become the current event in the Events panel.

- Return to the Events panel by clicking on the /tmp/data_import tab.

- Hover the mouse cursor over the description of the current event to see the details of the arguments passed to the instance of the work function which consumed the most time.

View Daemons Sear	ch Summary Prof	iles Timelin	es Tools Help	
🖻 📮 🎇 =	• 🗊 🕅 🗊	<u>ر</u> פ פ	🖻 🔎 Σ	▋▋ ◎╏ ¥₽ 単 出
ce /tmp/data_import	Profiles state (0	to 8950)		
			Events proproprio	
Event	Process	Thread	Time (sec)	Tag Description
RETURN_work	appAl	main	34.184911673	returning from work() errno=2
ENTER_semop	appAl	main	34.184912585	calling semop(semid=688146,sops=0xbfb4
RETURN_semop	appAl	main	34.184921201	returning from semop()=0 errno=2
RETURN_semop	appAl	sin	34.184921853	returning from semop()=0 errno=0
ENTER_nanosleep	appAl	main	34.184921943	calling nanosleep(req=0xbfb49784,rem=0x
ENTER_semop	appAl	sin	34.184922828	calling semop(semid=688146,sops=0x407
RETURN_semop	appAl	COS	34.184929093	returning from semop()=0 errno=0
ENTER_semop	appAl	COS	34.184929814	calling semop(semid=688146,sops=0x409
3	app	COS	34.226502679	arg1=0.551937
1	app	sin	34.226515525	arg1=0.829038
2	app	sin	34.226550440	arg1=0.833886
RETURN_nanosleep	appAl	main	34.234931167	returning from nanosleep()=0 errno=2
ENTER_random	appAl	main	34.234931976	calling random() caller=0x8048b12 [m
RETURN_random	appAl	main	34.234932916	returning from random()=1630634994
ENTER_work	appAl	main	34.234933433	tag.1 calling work(control=994) caller=0x
RETURN_work	appAl	main	34.235355154	ret calling work (control=994)
ENTER_semop	appAl	main	34.235356315	cal caller=0x8048b48 [main() at app.c:
RETURN_semop	appAl	main	34.235365921	ret frame=0xbfb497c8
ENTER_nanosleep	appAl	main	34.235366657	cal
RETURN_semop	appAl	sin	34.235384368	ret
ENTER_semop	appAl	sin	34.235385285	cal Raw Arguments: 0x8048b48, 0xbfb497c8,
RETURN_semop	appAl	COS	34.235390606	ret 0x3e2
ENTER_semop	appAl	COS	34.235391463	calling semop(semid=688146,sops=0x409
3	app	COS	34.276620354	arg1=0.544639
1	app	sin	34.276635275	arg1=0.833886
2	app	sin	34.276666025	arg1=0.838671
RETURN_nanosleep	appAl	main	34.285375275	returning from nanosleep()=0 errno=2
ENITED random	ann Al	main	24 205276062	calling random() callor 0v0040h12 [m

Figure 4-51. NightTrace - Description of Longest Instance of work Function

If you look at the code for the work function, its duration is completely dependent on the single parameter control.

Of course the actual value of the argument for your trace run may differ from the figure above, since the value of the argument is generated at random.

Batch Summary of Functions

You can also use **ntrace** in non-GUI mode to obtain summary information for specific functions. Assuming you had captured some trace data for your application; perhaps like this:

```
./appAI &
ntraceud --join /tmp/data
sleep 5
ntraceud --quit-now /tmp/data
killall appAI
```

You could invoke **ntrace** with the following parameters:

ntrace --summary=fs:work appAI /tmp/data

and it would generate output similar to the following:

Summary: work Function entry/	return states
State Summary Results	
Number of states found:	362
Maximum state duration: Minimum state duration: Average state duration: Total of state durations:	0.000134952 at offset: 2444 0.000001614 at offset: 976 0.000058733 0.021261507
Number of state gaps found:	362
Maximum state gap: Minimum state gap: Average state gap: Total of state gaps:	0.032288239 at offset: 58 0.017699007 at offset: 46 0.024923208 9.022201284

Shutting Down

- Select Exit Immediately from the File menu of NightTrace to terminate the NightTrace session.
- Select Exit Immediately from the File menu of **nlight** to terminate the **nlight** session.
- Issue the following command from a terminal shell to kill the **appAI** process which we left running:

killall appAI 2>/dev/null

Conclusion - NightTrace

This concludes the NightTrace portion of the NightStar RT Tutorial.

NightStar RT Tutorial

5 Using NightProbe

NightProbe is a graphical tool for viewing and modifying data from independently executing programs as well as recording data for subsequent analysis.

This chapter assumes you have already built the **app** program and it is running under the control of NightView. If you have not built the program, do so using the instructions in "Building the Program" on page 1-4 and execute the application via the following command before proceeding:

./app &

Invoking NightProbe

Programs to be probed do not need to be instrumented with any special API calls. However, in order for NightProbe to refer to symbolic variable names, the program should be compiled with debug information (typically the -g compilation option).

NightProbe takes advantage of significant performance capabilities of the RedHawk kernel, eliminating intrusion on the process by sampling and modifying variables in other programs using direct memory fetches and stores.

Invoke NightProbe by selecting NightProbe Monitor from the Tools menu of any of the NightStar Tools currently running. You may also invoke NightProbe by using the Night-Probe desktop icon or type the following command:

nprobe &

at a command prompt.

The NightProbe main window is displayed.

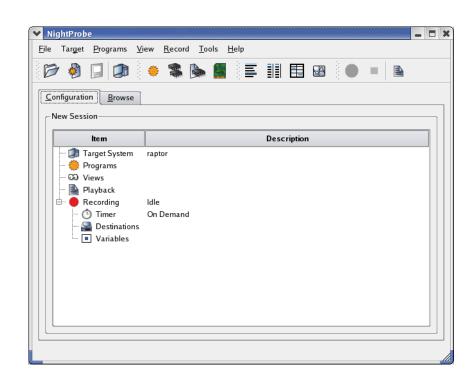


Figure 5-1. NightProbe Main Window

Selecting Processes

NightProbe has the ability to probe several kinds of resources, including programs, shared memory segments, memory mapped entities, and PCI devices.

- Right-click the Programs icon on the Configuration page and select the Program... menu option.

The Program Selection dialog is presented:

♥ Program Select	tion
- Program	
Process Name	Select
PID	Select
Symbol File	Select
	OK Reset Cancel Help

Figure 5-2. Program Selection Dialog

- Press the Select... button to the right of the PID field

The Process Selection dialog will appear.

ſ	nprobe Processes Target: r			
	Filter .*		Filter Clear	Apply To Name 🔻
	PID	Owner 🔻	Name	Command 🛋
	1	root	init	/sbin/init
	1741	root	syslogd	/sbin/syslogd
	1745	root	klogd	/sbin/klogd
	1757	root	portmap	/sbin/portmap
	1777	root	rpc.statd	/sbin/rpc.statd
	1789	root	mdadm	/sbin/mdadm
PI	D		Program	Path
				OK Cancel

Figure 5-3. Process Selection Dialog

- Enter app in the Filter field and press the Enter key.

The list is filtered to only those process whose name includes **app** and an entry should be selected in the table.

- Ensure that a single item appears in the table and press Enter again to close the dialog. If multiple items appear in the table, double-click on the **app** process associated with your user name.

The process ID associated with the **app** program is placed in the PID text field and the Process Name and Symbol File text fields are updated accordingly.

- Press Enter to close the dialog.

The **app** program is added to the list of resources to be probed as is shown under the **Pro-**grams item in the Configuration page.

Viewing Live Data

- Click on the Browse tab in the NightProbe main window.

The Live Browser is displayed.

▼ NightProbe	×
<u>F</u> ile Tar <u>g</u> et <u>P</u> rograms <u>V</u> iew <u>R</u> ecord <u>T</u> ools <u>H</u> elp	
📂 🖗 📮 🐽 🛸 🎆 🚍 📰 🖽 🚇 💻 🖿	
<u>C</u> onfiguration <u>B</u> rowse	
_ Live Browser	1
Filter .* Filter Clear Apply To Variables View All	•
Item Value	
Ē- 🌞 app pid=31229	
Refresh Auto Refresh	.50s
	1.

Figure 5-4. NightProbe Browse Panel

The **Browse** page serves two purposes. It allows you to browse your program to select variables of interest for recording or for viewing with alternative View panels.

It also provides you instant viewing of variables using the tree shown directly within the Browse page.

- Expand the **app** entry in the tree.

The items under a program's icon include all global variables as well as any nested scopes such as Ada packages, or functions that contain static data items.

Each variable item has an icon which indicates whether the variable is a scalar, a pointer, or a composite item such as an array or structure.

The data variable is a composite object and can be expanded.

- Expand the data variable.

ltem		Value
🖻 – 🌞 app	pid=31229	
🕂 💷 data		
🗄 🖬 🖬 data[0]		
- v		
···· 🔳 sema	1048590	
···· 🔳 rate	5000000	
🗄 🖬 💶 ptrs		

Figure 5-5. Expanded Data Item

The downward pointing arrow head is the array subscript expansion icon. By clicking the icon, an additional component of the array is shown.

- Click the array expansion icon so that data[1] is shown
- Expand both structures displayed, data[0] and data[1].

In the Browse page, the current value of all variables shown in the tree is displayed whenever you press the Refresh button at the bottom of the page, whenever an automatic refresh occurs as controlled by the Automatic refresh checkbox, or when the page receives or loses focus.

- Click the Automatic Refresh checkbox.

This causes the display to automatically refresh at the rate shown in the spinbox to the right of the Automatic Refresh checkbox.

Note the values of the count, angle, and value components of each component of the data array changing.

Modifying Variables

The app main program wakes each thread iteratively to do processing. The state variable controls whether this should occur or not.

Note that the current value of the state variable is the enumeration value run.

Double-click the value of the state variable.

ltem	Value
🗄 🕂 🌞 app	pid=31450
🕂 💶 data	
🕂 💶 data[0]	
- 🕨 name	0x08048e4c
🔳 count	23098
🔳 delta	8.726646259971648E-03
🔤 🔳 angle	2.015680753127312E+02
🔤 value	4.848096201641618E-01
🗎 🖬 💷 🗇 🖨	
- 🕨 name	0x08048e50
- count	23098
🔳 delta	8.726646259971648E-03
🔤 🔳 angle	2.015680753127312E+02
🔤 value	8.746197071849462E-01
~ 🗸	
🔳 sema	1081359
···· 🔳 rate	5000000
🕂 💶 ptrs	
state	run

Figure 5-6. Variable Modification in Progress

The cell containing the value is frozen from updates and the current value is selected.

To change the value of a variable, all we need to do is supply a new value and commit the change to the program.

- Type the following in the cell:

hold

- Press the Enter key to commit the value to the program.

The value of the state variable is now hold which prevents the program from waking the threads for computation, as shown in the source code snippet from **app.c**:

```
91 for (;;) {
92 struct timespec delay = { 0, rate };
93 nanosleep(&delay,NULL);
94 work(random() % 1000);
95 if (state != hold) semop(sema.&trigger,1);
96 }
```

- Change the value of the state variable back to run by double clicking the cell and then selecting run from the enumeration list and press Enter.

Selecting Variables for Recording and Alternative Viewing

Each variable has a Mark and a Record attribute. The Mark attribute, when set, indicates that the variable is of particular interest and may be viewed in other panels. The Record attributes specifies that the variable is to be included in recording sessions.

Double-clicking an item causes the color to turn a reddish color and sets its Mark and Record attributes. Alternatively, you can use an item's context menu to individually set its attributes.

- Double-click the count, angle, and value fields from both data[0] and data[1] structures.
- Double-click the rate variable.

The Browse page tree should look similar to the following:

<mark>∕ NightProbe</mark> <u>F</u> ile Target <u>P</u> rograms <u>V</u> ie	ew <u>R</u> ecord <u>T</u> ools <u>H</u> elp
6 🖗 🛛 🗶	* 🛸 🔛 🗏 📰 🖽 📾 🌘 = 🖹
Configuration Browse	
- Live Browser	
Filter .*	Filter Clear Apply To Variables 👻 View All 👻
ltem	Value
🖮 🌞 арр	pid=31450
🕂 💶 data	
🖻 🖬 🖬 data[0]	
- 🕨 🗈 name	0x08048e4c
count	26674
🔳 delta	8.726646259971648E-03
🗖 angle	
🔤 value	2.923717045885066E-01
🖻 🗉 🔳 data[1]	
🕨 name	
🖬 count	
🔳 delta	
- 🖬 angle	
🛄 🖬 value	9.563047560040737E-01
···· 🔳 sema	1081359
rate	5000000
🕂 💷 ptrs	
💷 🔳 state	run
Refresh	Auto Refresh 🕱 1.50s 🖡

Figure 5-7. Mark and Record Attributes Set

Selection of Views

NightProbe provides various methods for viewing data:

- The Browse page
- List View
- Table View
- Spreadsheet View
- Graph View

Additionally, you can stream the output of a recording session to NightTrace or a user application for live analysis, or to a file for subsequent analysis within NightProbe.

Table View

A Table view provides a scrollable table with variables spread across the columns and rows containing the values of the variables, over time.

- Select the Table option from the View menu.

▼ NightProbe	///// _ 🗆 🗙
<u>F</u> ile Target <u>P</u> rograms <u>V</u> iew <u>R</u> ecord <u>T</u> ools <u>H</u> elp	
📂 🏘 📮 🔊 🔅 🛸 🌆 🚍 🏭 🖽 🔴 💻	
Configuration Browse Table Table	······································
Mode: View Live Samples 💌 Select Items	
Sample #: 0	
Refresh Automatic Refresh 1.50s	Clear

Figure 5-8. Table View

Initially, the table is empty. The first step is to select the items we wish to display in the table.

- Press the Select Items... button.

N		Select Items for View	
ſ	Table Vi	ew Item Selection	
	Select i	tems to be shown in Table View.	
Select items from the table below, which is populated with Marked and Recorded variables. Click the Browse button to add more items to the table.			
	Show	ltem	
	×	data[0].count	
	×	data[0].angle	
	×	data[0].value	Show All
	×	data[1].count	SHOW AII
	×	data[1].angle	
	×	data[1].value	Hide All
	×	rate	
L			,
		OK Browse Cancel	Help

Figure 5-9. Item Selection Dialog

This dialog allows you to select items that have the Mark or Record attribute set.

By default, the dialog sets up defaults to display such variables.

- Hide all elements of the data[1] component by clicking their rows in the Hide column.
- Press the OK button.

The table now has five columns, one for the sample number and one for each of the variables we selected in the previous step.

- Check the Automatic Refresh checkbox

At the rate defined in the spinbox to the right of the Automatic Refresh checkbox, new samples are taken of the variables in the table.

<u>C</u> onfigur	<u> </u>	able		
·	View Live Samples	Select Items		·····
Sampl	e 🔻 🛛 data[0].coun	t data[0].angle	data[0].value	rate
4	49614	4.329638275417953E+02	-5.446390353823994E-01	5000000
5	49634	4.331383604669945E+02	-3.907311288927318E-01	5000000
6	49654	4.333128933921936E+02	-2.249510547711851E-01	5000000
7	49674	4.334874263173928E+02	-5.233595668116241E-02	5000000
8	49694	4.336619592425920E+02	1.218693429693408E-01	5000000
9	49714	4.338364921677912E+02	2.923717043025947E-01	5000000
10	49734	4.340110250929903E+02	4.539904993478616E-01	5000000
11	49754	4.341855580181895E+02	6.018150228007619E-01	5000000
12	49774	4.343600909433887E+02	7.313537013190109E-01	5000000
13	49794	4.345346238685879E+02	8.386705677055774E-01	5000000
14	52369	4.570057379879814E+02	-9.953961983217676E-01	5000000

Figure 5-10. Table in Auto Refresh Mode

Values are shown in blue if they have changed since the previous sample.

You can sort by variable value by clicking on a column header.

- Clear the Automatic Refresh checkbox
- Click on the column header for data[0].value and then click again so that the table is sorted from largest to smallest value.

The value shown at the top should be nearly 1.0 if enough samples have been taken (the value of data[0].value is that of a sine wave).

You can modify variables using the Table view in the same manner as described in "Modifying Variables" on page 5-5.

Graph View

The Graph panel presents individual variables as separate lines on a graph.

- Select the Add New Page option from the View menu.
- Select the Graph option from the View menu.

▼ NightProbe	- 🗆 X
<u>F</u> ile Target <u>P</u> rograms <u>V</u> iew <u>R</u> ecord <u>T</u> ools <u>H</u> elp	
D 🖗 💭 🗢 🛸 🌇 🗐 🗄 🖽 🔴 💻 🖿	
<u>Configuration</u> <u>Browse</u> Table Page <u>4</u> Graph	. IV
Mode: View Live Samples Select Items	
800 -	
200 -	
ر 0 0 Most Recent Samples	1000
Sample #: 0	
Refresh Automatic Refresh 🗌 1.505 👻 🚺 🕨 🍙 💌 🔎 🖉 Clea	ar

Figure 5-11. Graph Panel

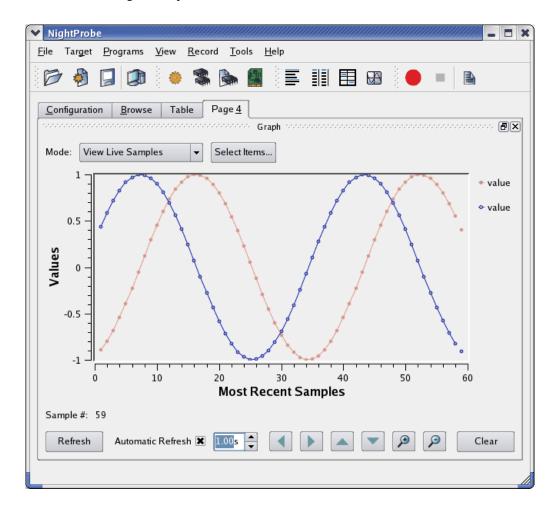
Initially, the graph is empty.

- Press the Select items... button.

Unlike the table view, none of the items in the Select Item dialog are selected to be shown. Typically, only one or very few items are shown on a single graph.

- Mark the data[0].value and data[1].value items as Shown by clicking their respective rows in the Show column.
- Press the OK button.
- Check the Automatic Refresh checkbox.

- Change the refresh rate to 1.0 seconds in the spinbox to the right of the Automatic Refresh checkbox.



Two lines begin to be plotted as shown below.

Figure 5-12. Graph Panel Actively Displaying Values

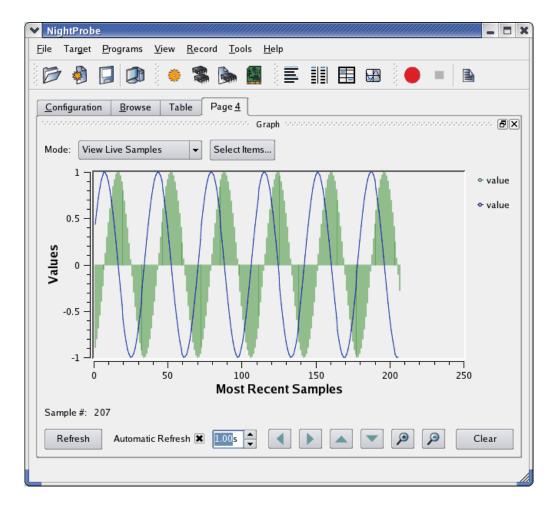
- Select the Edit... option from the context menu of one of the value items in the legend at the right-hand side of the graph panel (right-clicking activates

the context menu).

Edit Curve Attributes
Curve Attributes
Variable value
Style Lines 👻
Symbol Ellipse 👻
Color
OK Cancel Help

Figure 5-13. Edit Curve Attributes Dialog

- Select Sticks from the Style option list.
- Click on the colored block to activate a color selection dialog to change the color.
- Press the OK button to close the color selection dialog.



- Press the OK button to close the Edit Curve Attributes dialog.

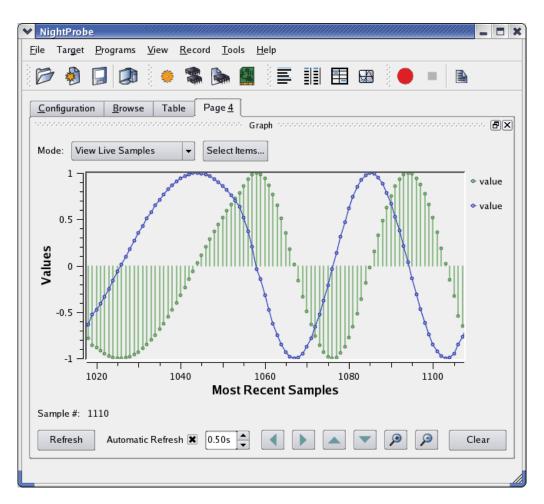
Figure 5-14. Graph Panel with Modified Curves

- Change the refresh rate to 0.5 seconds

The program uses the rate variable to determine the frequency at which the threads are activated to do their calculations.

- Using the **Browse** page or the **Table** panel, change the value of the rate variable from 50000000 to 25000000.

This change effectively doubles the frequency at which the threads operate, so the sine and cosine waves will change shape.



Sending Probed Data to Other Programs

Data values may be recorded to files for subsequent processing, or may be recorded and streamed to NightTrace for live processing.

Similarly, you can send recorded data to any process of choice.

- Raise the Configuration page by clicking on its tab.

ltem	Description
🐊 Target System	narf
🖕 🧓 Programs	•
🛛 🖾 🌞 app	pid=18128
- OD Views	
🖹 Playback	
🗄 🖷 🦲 Recording	ldle
🕐 Timer	On Demand
···· 🞴 Destinations	
🖃 🔲 Variables	
data[0].count	int
🔳 data[0].angle	double
data[0].value	double
🔳 data[1].count	int
data[1].angle	double
🔳 data[1].value	double
🔳 rate	int

Figure 5-15. Recording area of Configuration Page

The Recording portion of the configuration tree indicates the Timing source for recording, the recording Destinations, and the list of variables whose Record attributes are set.

- Right-click on the Timer item in the Recording tree and select the Clock... option.

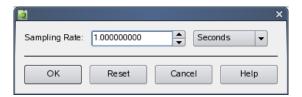


Figure 5-16. Clock Selection Dialog

This dialog controls the rate at which recording samples will be taken.

- Change the units to Milliseconds from the option list Sampling Rate option list.
- Change the Sampling Rate value to 100.0.
- Press the OK button.

The Timer item and description in the tree changes to reflect this activity.

The recording destination will be a user application.

- Right-click the Destinations item and select To Program...

Record To Program	×
General FBS Advanced	
Program Path Selec	t
Program Arguments	
Output File /dev/null Selec	t
Working Directory Select	
X Display 0.0	
Activiation	
Launch From NightProbe Server	
When Stopping Terminate Process	
OK Cancel	Help

Figure 5-17. Record To Program Dialog

- Type **api** into the **Program Path** text field.
- Replace the /dev/null text in the Output File text field with the following.

/tmp/api.out

- Press the OK button.

A simple application which uses the NightProbe API to consume and print the values of recorded samples was copied into the **tutorial** directory in "Creating a Tutorial Directory" on page 1-3.

- Type the following command in your terminal session to build the program:

```
cc -g -o api api.c -lnprobe
```

The Recording area of the Configuration page should look similar to the following.

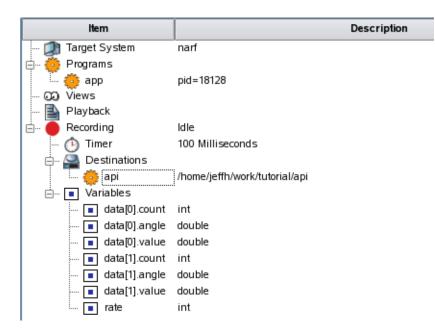


Figure 5-18. Recording Area of Configuration Page w/ Destination

Now that we have selected the variables to record, the recording timing source, and the recording destination, we can proceed to record samples and stream them to the **api** application.

- Press the Record icon on the toolbar:



- View the output of the api program as samples are recorded and passed to it.
- Enter the following command in a terminal session:

tail -f /tmp/api.out

		gnome-terminal	_ = ×]
<u>F</u> ile <u>E</u> dit <u>V</u> iew	<u>T</u> erminal Ta	a <u>b</u> s <u>H</u> elp	
item: count:		29072	
item: angle:		253.701060	
item: value:		0.694658	
item: count:		29072	
item: angle:		253.701060	
item: value:		-0.719340	
item: rate:		5000000	
Sample 479			
item: count:		29074	
item: angle:		253.718513	
item: value:		0.681998	
item: count:		29074	
item: angle:		253.718513	
item: value:		-0.731354	
item: rate:		5000000	
Sample 480			
item: count:		29076	
item: angle:		253.735967	
item: value:		0.669131	
item: count:		29076	
item: angle:		253.735967	
item: value:		-0.743145	
item: rate:		5000000	
			•

The program will generate output similar to the following:

Figure 5-19. Example Output of Graph Program

- Stop the recording process by pressing the Stop icon on the Recording toolbar:



For more information on the NightProbe API, refer to the "NightProbe API" chapter in the *NightProbe User's Guide*.

Using Datamon to Modify Program Variables

The Data Monitoring Application Programming Interface is part of the NightStar RT tool set.

Data Monitoring allows you to specify executable programs that contain Ada, C, or Fortran variables to be monitored, obtain and modify the values of selected variables by specifying their names, and obtain information about the variables such as their addresses, types, and sizes.

NOTE

Ada programs are only supported if compiled with the Concurrent MAXAda compiler which generates proper DWARF debug information.

Data Monitoring is a powerful capability with a rich API. It also allows you to obtain detailed symbolic and attribute information for variables in a program file. However, for our purposes, we will write a very simple program which changes the value of a single variable.

Refer to the *Data Monitoring Reference Manual* for more information about Data Monitoring.

The source code for our **set_rate** program follows:

```
#include <stdlib.h>
#include <stdio.h>
#include <datamon.h>
#define check(x) \setminus
   if((x)) {fprintf(stderr, "%s\n", dm_get_error_string());exit(1);}
main(int argc, char * argv[])
{
   program_descriptor_t pgm;
   object descriptor t obj;
   char buffer[100];
   if (argc != 2) {
      fprintf (stderr, "Usage: set_rate integer-value\n");
      exit(1);
   }
   check(dm_open_program("app",0,&pgm));
   check(dm_get_descriptor("rate",0,pgm,&obj));
   check(dm_get_value(&obj,buffer,sizeof(buffer)));
   check(dm_set_value(&obj,argv[1]));
   printf ("rate: old_value=%s, new_value=%s\n", buffer, argv[1]);
}
```

The dm_open_program function initializes Data Monitoring on the specified process name and PID (in this case zero, which instructs the call to use any process matching the specified name).

The dm_get_descriptor call looks for the specified variable name and returns information about the variable. It also maps the underlying memory page of the variable in the **app** process into the monitoring process.

The dm_get_value and dm_set_value routines return and set the value of the variable using direct memory reads and writes; the **app** process is not affected in any other way than having the value of the rate variable changed.

The **set_rate.c** source file was copied into the current working directory during the activities in "Creating a Tutorial Directory" on page 1-3.

- Compile the program using the following command:

cc -g -o set_rate set_rate.c -ldatamon -lccur_rt

While this portion of the tutorial is in no way dependent on NightProbe itself, we will use NightProbe to see the effect of changing the rate variable using the Datamon API.

- Raise the Graph panel by clicking on the tab labelled Page 4 in Night-Probe.
- Use the Pan Right button in the graph panel to move the viewport to the end of the graph set -- click the button repeatedly until the end of the graph is seen:



- Change the value of the rate variable in the **app** process by issuing the following command:

```
./set_rate 123456789
```

As shown in the source code above, the program prints the previous value of the rate variable and then sets it to the value specified as an argument to **set_rate**.

The sine and cosine waves change shape as shown in the Graph panel.

Conclusion - NightProbe

To terminate NightProbe operations, execute the following steps:

- Select the Exit Immediately option from the File menu

This concludes the NightProbe portion of the NightStar RT Tutorial.

6 Using NightTune

NightTune is a graphical tool for analyzing and adjusting system activities.

This chapter assumes you have already built the **app** program and it is running. If you have not built the program, do so using the instructions in "Building the Program" on page 1-4 and execute the application before proceeding: ./app &

Invoking NightTune

NightTune can be launched with the following command at a command prompt:

ntune &

Or it may be launched by double-clicking on the NightTune desktop icon.

For some aspects of this tutorial, it will be necessary to execute NightTune as the **root** user or to ensure that your user account has appropriate privileges. See the "Setting Up User Privileges" on page 1-2 for more information.

ē 🛛 😣 🛛	🔌 🔂 🤅	d 🗟 🗢 💲 🖇	🍯 🗢 🚰 🕐	reate Panels For: 🛛 🖛 💉 🧩 🛠 🖘
		r Process List: Mice RPri CL	Command	raptor CPU Shiekling and Binding: raptor CPU Shiekling and Binding: raptor CPU Shiekling and Binding: raptor: Intel(R) Xeon(TM) CPU 2.40GHz
			Users Users Japache Japac	



Monitoring a Process

First monitor the running **app** process.

- In the Process List panel, click anywhere within the panel and the type Ctrl+F.
- A Find bar appears at the bottom of the panel. Type **app**, and the process list will be automatically expanded and the first process whose process name includes the word app will be selected.

PID	State	Parent	Size	%CPU	CPU Time	CPU	Affinity	Nice	RPri	CL	L Command 🔺
											🗄 🖳 🔍 Users
											🕂 🗊 apache
											🖶 🗊 bin
											🕂 🗊 danr
											🗄 🖉 gdm
											🕂 🗊 jeffh
4534	Waiting	4527	1672	0.0	0.00	2	all	0	0	OT)T 💮 🥮 ack_vpn
1501		1	137	0.0	16.47						— 🧑 арр
1513	Waiting	1	3 03	0.0	18.26	2	all	0	0	OT)T esistant /app
4633	Waiting	4628	4376	0.0	0.10	2	all	0	0	OT)T 💮 🧶 bash
7044	Waiting	7036	4372	0.0	0.15	1	all	0	0	OT	π 🖶 🌞 bash
7921	5	7916		0.0	0.09	3	all	0	0	OT	
8978	Waiting	8973	4376	0.0	0.05	0	all	0	0	OT	
2024	Waiting			0.0	0.05	3	all	0	0	OT	
5979	5				0.12	0	all	0		OT	
4991	5				0.35	3	all	0	-	OT	
	Waiting		5304	0.0	0.14	2	all	0	0	OT	
1886		1	120	0.0	25.66						- 🔞 ntrace
	Waiting			0.0	0.15	3	all	0	50	FF	
2279	Waiting	4041	104	0.0	13.24	1	all	0	0	OT	
											🖶 🗊 jimbo

Figure 6-2. Expanded Process List

If the selected process is not our **app** process, press the Next icon in the Find bar until the correct process is selected.

Notice that the icon associated with the **app** process has a small gray gear superimposed on the orange process icon. This indicates that process is multi-threaded.

Ö

PID	State	Parent	Size	%CPU	CPU Time	CPU	Affinity	Nice	RPri	CL	. Command 🔺
											🗄 🖳 🚅 Users
											🗄 🗊 apache
											🕂 🗊 bin
											🕂 🗊 danr
											🕀 🐨 🚺 gdm
											🖃 🗊 jeffh
	Waiting	4527		0.0	0.00	2	all	0	0	OT	
1501		1	137		17.15						🕂 🧑 арр
1501	Waiting			0.4	14.77	0	all	0		OT	
1521	5			0.0	1.21	2	all	0	-	OT	
11522	5			0.0	1.17	1	all	0		OT	
	Waiting			0.0	0.00	3	all	0	-	OT	
11513	Waiting		3 03	0.0	18.26	2	all	0	-	OT	
4633	Waiting	4628	4376	0.0	0.10	2	all	0	0	OT	
7044	5			0.0	0.15	2	all	0	0		
7921	Waiting	7916	4380	0.0	0.09	3	all	0	0	OT	
8978	Waiting	8973	4376	0.0	0.05	0	all	0	0	OT	Г 🧶 bash
12024	Waiting	12019	4380	0.0	0.05	3	all	0	0	OT	Г — 🔶 bash
15979	-	15974			0.12	0	all	0	0	OT	
24991	Waiting	24986	4376	0.0	0.35	3	all	0	0	OT	Г 🧼 bash
11886		1	126	9.7	26.11						🔤 🧑 ntrace

- Select the Show Threads option from the context menu associated with the app process.

Figure 6-3. Process List with Threads

The panel shows characteristics of each thread and of the entire process. In particular, they include:

- the virtual memory size of the process
- the percentage and amount of CPU time used by each thread and by the whole process.
- CPU on which each thread ran most recently
- CPU affinity for each thread (the set of CPUs on which the thread is allowed to run)
- scheduling characteristics of each thread

The set of columns displayed can be modified by clicking the Display Fields option of the context menu for the panel, and then choosing individual fields by checking or unchecking their menu items.

Tracing System Calls

NightTune provides a handy interface for tracing system calls made by a process. This is essentially the same as using the strace(1) command, except that NightTune provides the output in a dialog which can be searched and controlled.

- Select the Trace System Calls... option from the context menu associated with the second thread in the **app** program and press the run button.

18	strace of PID 6670:	
semop(98304, 0xb79ab3ae, 1)	= 0	
semop(98304, 0xb79ab3ae, 1)	= 0	
semop(98304, 0xb79ab3ae, 1)	= 0	
semop(98304, 0xb79ab3ae, 1)	= 0	
semop(98304, 0xb79ab3ae, 1)	= 0	
semop(98304, 0xb79ab3ae, 1)	= 0	
semop(98304, 0xb79ab3ae, 1)	= 0	
semop(98304, 0xb79ab3ae, 1)	= 0	
semop(98304, 0xb79ab3ae, 1)	= 0	
semop(98304, 0xb79ab3ae, 1)	= 0	
semop(98304, 0xb79ab3ae, 1)	= 0	
semop(98304, 0xb79ab3ae, 1)	= 0	
semop(98304, 0xb79ab3ae, 1)	= 0	
semop(98304, 0xb79ab3ae, 1)	= 0	
semop(98304, 0xb79ab3ae, 1	\$	
Strace Options		Close Help

Figure 6-4. Strace Output of Thread

As shown in the figure above, the selected thread makes no system calls other than **semop(2)** which is associated with the line 46 of **api.c**, as shown in this code segment:

```
36 void *
37 sine_thread (void * ptr)
38 {
39
      control_t * data = (control_t *)ptr;
40
      struct sembuf wait = \{0, -1, 0\};
41
      work(1);
42
43
      trace_set_thread_name (data->name);
44
45
      for (;;) {
         semop(sema, &wait, 1);
46
47
         data->count++;
48
         data->angle += data->delta;
49
         data->value = sin(data->angle);
50
      }
51 }
```

- Press the Close button to stop the system call trace and close the dialog.

Process Details

NightTune provides detailed analysis of process attributes.

- Select the Process Details... option from the context menu of any thread in the **app** program.

Memory <u>U</u> sage	<u>M</u> emory	<u>F</u> ile Descripto	rs	<u>S</u> ignals <u>C</u>	apabilities <u>E</u> nviror	nment	
	kB			Usage			
Total	12084			Shared			
Reserved	0			Residency			
Text	3				0		12084
Library	11933			Usage:	Reserved Text	.ibrary Data/Heap St	ack
Data/Heap	136			Shared:			
Stack	12						
Shared	5752		;	Residency	Resident & Locked	Resident & Unlocked	Non-resident
Non-Shared	6332						
Resident	652						
Locked	0						
Unlocked	652						
Non-resident	11432						
				•			••
						Update Close	Help

Figure 6-5. Process Details Dialog

All information displayed in this dialog is read-only in nature. You cannot make changes to process attributes using this dialog.

Six tabbed pages provide detailed information about the process, including:

- Memory Usage
- Memory details

- File Descriptors
- Signals
- Capabilities
- Environment

The Memory Usage page provides summary information of the virtual and resident usage of memory in both textual and graphical panes.

Process Details - Memory Details

- Click on the Memory tab to raise that page.

Memory <u>U</u> sage	<u>M</u> emory	<u>F</u> ile Descriptors	<u>S</u> ignals	<u>C</u> apabilities	<u>E</u> nvironme	ent			
NUMA Node									
Local NUMA Nod									
Process bias is no L ocked/Resident		to a single node							
Locked/Resident									
Exists									
					_	_	_	_	
0x00002aaaaa	-								
a0 a2	a4 a	6 a8 aa		ae b0	, b2	b4 b	6 b8	ba bc	be
0x_00000000000									
0 .1	2 2	4 5	6	7 0	0	h	-	d 0	4
	<u>f</u> í			<u> </u>					
Current Page: 0x	00002aaaa	aae7000					Zoom Max		
Status: Re	sident			Previous	Region	evious Page	Zoom Out	Next Page	Next Region
NUMA Node: 0]			
				Shift	Min	Shift Left	Zoom In	Shift Right	Shift Max
							Zoom Min		
-Memory Region	Information								1
File Mapping: /	SYSV01050	025, offset 0x7000							
Addresses: 0)x00002aaa	aaae0000 - 0x00002	2aaaaaeeaff	f Ac	tive: 0		Sh	ared: 0	
Permissions:	K Read 🗙	Write Execute		Inac	tive: 1844674	44073709551	1615 Shared C	lean: 0	
Shared: S		_		Backed by Sv	ap: 1844674	44073709551	1615 Shared	Dirty: 0	
	239360							ivate: 40960	
Resident: 4							Private C		
NUMA Policy: [Dirty: 40960	

Figure 6-6. Process Memory Details Page

This dialog provides controls to allow you to get detailed memory information for any segment or page within the address space.

The controls in the graphical rows are similar to NightTrace in nature.

- Click anywhere on or above the rulers.
- Press Alt+UpArrow to zoom out completely.

The process's entire address space is now displayed. Each segment of the memory address space that is associated with pages in your process is indicated by at least a single vertical black line in the Exists row.

- Click on one of these lines
- Use the mouse wheel or the Zoom In button to zoom in until sufficient detail is available.

In the figure above, memory segments are shown as gray areas in the Exists row. The boundaries of memory segments are shown as vertical black lines. If the zoom factor is large enough, a memory segment may be portrayed as merely one or two vertical black lines.

Details about the memory segment are shown in the textual area in the bottom portion of the page.

The other rows show per-page information, including NUMA pools, and Locked and Resident attributes of the page.

NOTE

Locked and Resident information may not be available on all operating system versions. NUMA information is only applicable to systems supporting a Non-Uniform Memory Architecture and the information is only provided by some operating systems.

Alternatively, you can select a specific address by typing it into the Current Page text field.

See the NightTune User's Guide for more information on the Memory page.

Process Details - File Descriptors

The File Descriptors page lists all open file descriptors associated with the process, and provides a description of each.

Ni	ghtTune Proc	ess Details	:: raptor 8344: ntu	ne						//////// ×
M	emory <u>U</u> sage	<u>M</u> emory	<u>F</u> ile Descriptors	<u>S</u> ignals	<u>C</u> apabilities	<u>E</u> nvironme	nt			
			Pathi	name/Desc	ription					
0	/dev/pts/0									
1	/dev/pts/0									
2	/dev/pts/0									
3	pipe:[121803]	(pid 8344/nt	une fd 4)							
4	pipe:[121803]	(pid 8344/nt	une fd 3)							
5	pipe:[121806]	(pid 8344/nt	une fd 6)							
6	pipe:[121806]	(pid 8344/nt	une fd 5)							
7	socket:[12180]	7]: unix/strea	m: state=CONNECT	ED						
8	/usr/lib/NightTu	une/lib/ntune	.msg							
9	socket:[12182	9]: tcp: local=	raptor:42017 remot	e=raptor:25	517 state=ESTA	BLISHED (pid	2320/nslr	n fd 5)		
10	/proc									
11	/proc/shield/irc	ls								
12	/proc/shield/ltn	nrs	File or de filenam		ated with the file	descriptor in	one of thes	se forma	ts:	
13	/proc/shield/pr	ocs		e (deleted)						
14	/proc/ccur/swit	ches		ode] (other- [inode]: tcp/(pid) Jdp/raw: local=ip	:port remote=	ip:port stat	te=s (oth	er-pid)	
15	/proc/stat				: name=associat	ed-filename s	state=s			
16	/proc/meminfo			[inode]: pacl e either TCF	ket P states like EST/	ABLISHED, LI	ISTEN, FIN	LWAIT	L,	
17	/proc/vmstat				es like LISTENIN processes on thi					
18	/proc/diskstats				cted to the other		-	e same		
19	/proc/interrupts	5	-							
20	/proc/net/dev									
						L	Jpdate	Clo	se	Help

The figure below shows the file descriptors in use by an **ntune** process.

Figure 6-7. File Descriptors Page

The description includes the file name associated with a file descriptor (when relevant), connection information for a socket, and even identifies other processes using a pipe or socket when those processes are on the same system.

Process Details - Signals

Memory <u>U</u> sage	e <u>M</u> emory	<u>F</u> ile De	escriptors	<u>S</u> ignals	<u>C</u> a	pabilities	<u>E</u> nviro	nment	
Number 🔻	Name	Pending	Shared Per	nding B	ocked	lgnored	Handled	Restart	Description
1	SIGHUP								Hangup
2	SIGINT								Interrupt
3	SIGQUIT								Quit
4	SIGILL								Illegal instruction
5	SIGTRAP								Trace/breakpoint trap
6	SIGABRT								Aborted
7	SIGBUS								Bus error
8	SIGFPE								Floating point exception
9	SIGKILL								Killed
10	SIGUSR1								User defined signal 1
11	SIGSEGV								Segmentation fault
12	SIGUSR2						×		User defined signal 2
13	SIGPIPE								Broken pipe
14	SIGALRM								Alarm clock
15	SIGTERM								Terminated
16	SIGSTKFLT								Stack fault
17	SIGCHLD								Child exited
18	SIGCONT								Continued
19	SIGSTOP								Stopped (signal)
20	SIGTSTP								Stopped
21	SIGTTIN								Stopped (tty input)
22	SIGTTOU								Stopped (tty output)
23	SIGURG								Urgent I/O condition
	ciovenu.	_	_			_	_	_	

The Signals table displays attributes of signals.

Figure 6-8. Signals Page

The information shown includes indicators of signals currently pending or blocked by the application, as well as whether the application has a handler installed for a signal.

In the figure above, the application has a handler registered for SIGUSR2.

Changing Process Scheduling Parameters

It may be desirable to change the scheduling properties of a thread or process while it is running to see how that changes the behavior of an application. For instance, perhaps one thread is being starved of CPU time by other threads. You may wish to change its scheduling class to a real-time class and/or its priority to a higher priority.

- Select the Process Scheduler... option of the context menu associated with a thread in the **app** process.

😪 raptor: NightTune - Process Scheduler (on raptor) 🛛 🗙								
-11521 (app)								
	Current System Values							
Priority	Scheduling Class: Other							
Scheduling Class: Other 🗸	Nice Value: 0							
Nice Value:	Real-Time Priority: 0							
Real-Time Priority:	Time Quantun: 99 msecs							
Time Quantum: 99 ms	CPU Affinity: all							
CPU Affinity								
0 兽 🕱 1 兽 🕱 2 兽 🕱 3 兽 🕱	CPU Shielding Legend							
	🦁 Unshielded From Processes							
Set All Clear All	🌼 Shielded From Processes							
	CPU Down							
ОК Аррју	Reset Cancel Help							

Figure 6-9. Process Scheduler Dialog

In this dialog, it is possible to change the Scheduling Class, Nice Value, Real-time Priority, and/or Time Quantum. On multi-processor systems, it also is possible to change the CPU Affinity. For each CPU on which the process or thread is allowed to run, the checkbox with the number of that CPU should be checked. See "Setting Process CPU Affinity" on page 6-11 for more on this topic.

- Change the Scheduling Class to Round Robin by selecting that from a drop down list.
- Change the Real-time Priority to 3.
- Press the OK button.

NOTE

To change the Scheduling Class to Round Robin and change the Real-time Priority, it is necessary that NightTune be run by the **root** user or that your user account has appropriate privileges as described in "Setting Up User Privileges" on page 1-2.

The Process List panel now reflects these changes to the thread.

PID	State	Parent	Size	%CPU	CPU Time	CPU	Affinity	Nice	RPri	CL	Command 🔺
					-						🗄 👷 Users
											🕂 🗊 apache
											🖶 🗊 bin
											🕂 🗊 🕼 danr
											🖶 🗊 gdm
											🖻 🗊 jeffh
	Waiting	4527	1672	0.0	0.00	2	all	0	0	OT	
11501		1	137	1.3	18.18						🖨 🧔 app
11501	Waiting			0.9	15.67	2	all	0	0	OT	···· 🕸
11521	Waiting			0.4	1.28	0	all	0	3	RR	
11522	Waiting			0.0	1.23	0	all	0	0	OT	···· @
11523	Waiting			0.0	0.00	1	all	0	0	OT	· · · · · · · · · · · · · · · · · · ·
11513	Waiting	1	303	0.0	18.27	2	all	0	0	OT	🗝 🌻 assistant
4633	Waiting	4628	4376	0.0	0.10	2	all	0	0	OT	🧶 bash
7044	Waiting	7036	4372	0.0	0.16	2	all	0	0	OT	
7921	Waiting	7916	4380	0.0	0.09	3	all	0	0	OT	
8978	Waiting	8973	4376	0.0	0.05	0	all	0	0	OT	🗝 🌞 bash
12024	Waiting	12019	4380	0.0	0.05	3	all	0	0	OT	🗄 🥮 bash
15979	Waiting	15974	4380	0.0	0.12	0	all	0	0	OT	🜞 bash
24991	Waiting	24986	4376	0.0	0.35	3	all	0	0	OT	— 🔴 bash
11886		1	131	0.0	26.50						- 🔞 ntrace

Figure 6-10. NightTune Process List with modified thread

For the modified thread, the CL (Scheduling Class) field displays the value RR (Round Robin), and the RPri (Real-time Priority) field displays the value 3.

Setting Process CPU Affinity

This section only is applicable if the system running NightTune is a multi-processor system. If not, skip to "Conclusion - NightTune" on page 6-17.

The CPU Shielding and Binding panel shows the CPU hierarchy, shielding status (on Concurrent RedHawk Linux only), CPU usage, and process and IRQ bindings.

proprocessory raptor CPU Shielding and Binding: proprocessory 🗗 🗙									
raptor: Intel(R) Xeon(TM) CPU 2.40GHz									
System									
🖻 Chip 0									
🖶 🗄 🖉 🖉 🖶 🗄 🕮 C	PU 0	[13% Usage]							
🗄 🕁 🖉 🔅 🛄 c	PU 2	[7% Usage]							
Ė Chip 3									
📄 🗄 🖞 🖉 👶 🚥 c									
📄 🗄 🖞 🖉 👶 🚥 C	PU 3	[5% Usage]							

Figure 6-11. CPU Shielding and Binding Panel

The hierarchy is useful in visualizing the relationship of logical CPUs, especially in the presence of hyper-threaded and multi-core chips.

In the figure above, two chips each contain two local CPUs which are hyper-threaded siblings of each other. Hyper-threaded CPUs share some physical resources between them, yet operate in all user-visible ways as independent processors. Multi-core CPUs also share physical resources between their siblings, but much less so than with the hyper-threaded technology.

A process or thread has a CPU affinity, which determines the set of CPUs on which it may execute. It may even be restricted such that it may run on only a single CPU. Often this is called *binding* the process or thread. "Changing Process Scheduling Parameters" on page 6-10 described one way to change the CPU affinity. In addition, the CPU Status panel can be used to bind a process or thread quickly.

- Select Expand All from the context menu associated with the System item in the panel

The tree expands with leaves for bound processes and interrupts for each CPU.

- While the cursor is positioned over one of the threads in the **app** process, press and hold the *left* mouse button, then drag the thread to one of the CPUs in the CPU Shielding and Binding panel and release the mouse button.

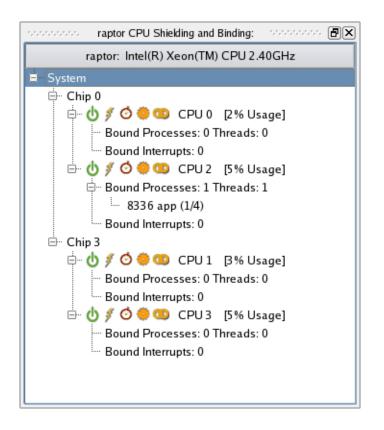


Figure 6-12. CPU Shielding and Binding Panel with Bound Thread

This action binds the selected thread to the particular CPU. That is, its CPU affinity is set to include only that single CPU. When a process' or thread's CPU affinity contains only a single CPU, that process or thread is listed in the CPU Status panel under the particular CPU's Processes tab. In this case, there is one entry under CPU 1. Because only one thread was bound to CPU 1 in this example, the entry includes the suffix (1/4), indicating that only 1 of the 4 threads is bound to that CPU.

The thread's new CPU affinity also is reflected in the Affinity field of the **Process** Monitor panel. That field displays a bit mask in hexadecimal, where the low order bit represents CPU 0, the next bit represents CPU 1, etc. In this case, the value 0x1 has only the lowest bit turned on, indicating CPU 0.

NightTune also can unbind a process quickly.

- While the cursor is over the thread entry in the CPU Status panel, press and hold the *left* mouse button, then drag the item to the Unbind icon at the upper right of the window (resembling a broken chain link) and release the mouse button.



The Process List panel will reflect that the thread is unbound once again.

Setting Interrupt CPU Affinity

The functionality described in this section only is available if NightTune was executed by the **root** user or your user account has appropriate privileges as described in "Setting Up User Privileges" on page 1-2. If this is not the case, skip to "Conclusion - NightTune" on page 6-17.

In addition to being able to set the CPU affinity of a process, NightTune can control the CPU affinity of an interrupt.

It may be desirable to change the CPU affinity of an interrupt. For instance, an interrupt may be occurring frequently and, depending on the CPU which handles it, may be interfering with an application running on that same CPU.

- Close the **Process List** panel by clicking on the right-hand most box in its title bar.
- In its place, open the Interrupt Activity panel by selecting the Interrupt Activity option from the Monitor menu and then the Text Pane option from its sub-menu.

) 🛛 🚷 🕴 🛛 🖗 🤹 📽 💲	\$ 3	M 🔗	• 👪 🕴	Create Pane	ls For: rap	tor 🔻 🔊 » 🔀 📲
raptor CPU Shielding and Binding:					• • •	
raptor: Intel(R) Xeon(TM) CPU 2.40GHz		-	🕴 CPU 1	🕴 CPU 2	🕴 CPU 3	Description
System ⊡ Chip 0	0	1000	0	0	0	timer
	3	0	0	0	0	KGDB-stub
Bound Processes: 0 Threads: 0	4	0	0	0	0	serial
Bound Interrupts: 0	9	0	0	0	0	acpi
🖨 🐨 Bound Processes: 1 Threads: 1	14	0	0	1	0	ide0
8336 app (1/4)	177	39	0	0	0	uhci_hcd
└── Bound Interrupts: 0 ⊟── Chip 3	185	0	0	0	0	uhci_hcd
in (1) ∮ ổ 🜻 💶 CPU 1 [22% Usage]	193	0	0	0	0	uhci_hcd
- Bound Processes: 0 Threads: 0	201	0	0	0	0	ehci_hcd
Bound Interrupts: 0	209	0	0	0	0	Intel 82801DB-ICH4
Bound Processes: 0 Threads: 0	217	0	0	0	0	rcim
Bound Interrupts: 0	225	0	0	82	0	eth0
	NMI	0	0	0	0	Non-maskable interrupts
raptor CPU Usage: https://www.commons.com/	LOC	1007	1000	1012	999	Local interrupts
	RES	240	230	234	208	Rescheduling interrupts
CPU 1	CAL	0	0	0	0	function call interrupts
CPU 2	TLB	0	0	0	0	TLB shootdowns
PU 3	TRM	0	0	0	0	Thermal event interrupts
0 100	SPU	0	0	0	0	Spurious interrupts
Legend: User System Wait Idle n/a	ERR	0	0	0	0	Error interrupts
	MIS	0	0	0	0	APIC errata fixups
][]			tive 🕢 Bound

Figure 6-13. NightTune with Interrupt Activity Panel

The panel shows the number of interrupts per second for each interrupt as handled on each CPU (if on a multi-processor system).

The chain link icon in the Interrupt Activity panel indicates that an interrupt may be handled by that particular CPU. However, if an interrupt may be handled by all CPUs, then no icon appears for that interrupt. The same information is displayed in the Bound Interrupts items for each CPU in the CPU Shielding and Binding panel.

Some systems may employ IRQ balancing which automatically changes IRQ affinities over time. This interferes with attempts to control interrupt affinity manually. For purposes of this tutorial, ensure that IRQ balancing is currently disabled by executing the following command as the root user:

/sbin/service irqbalance stop

To bind an interrupt to a single CPU, it may be dragged in much the same way as a process.

While the cursor is over an interrupt in the Interrupt Activity panel, you may press and hold the *left* mouse button, then drag the interrupt to the particular CPU in the CPU Shielding and Binding panel. Similarly, while the cursor is over an interrupt in the Bound Interrupts list of a CPU in the CPU Shielding and Binding panel, you may press and hold the middle mouse button, then drag the interrupt to a different CPU in the CPU Shielding and Binding panel.

To change an interrupt's affinity to allow multiple CPUs, but possibly exclude one or more, select the Set CPU Affinity... option from the context menu of any interrupt row in the panel.

NOTE

If you are not running as the root user or your user lacks appropriate privileges, the Set CPU Affinity... option will not be present in the context menu.

📸 💦 raptor: NightTune - Interi	rupt Affinity (on raptor) 🛛 🗙
217 IO-APIC-level rcim ▼ Interrupt Affinity 0 % 第 1 % 第 2 % 第 3 % 第 Set All Clear All OK Apply	Current Affinity all CPU Shielding Legend Unshielded From Interrupts Shielded From Interrupts CPU Down Reset Cancel Help

Figure 6-14. Interrupt Affinity Dialog

For each CPU on which the interrupt is allowed to be handled, the checkbox with the number of that CPU should be checked. The changes take effect when the OK or Apply button is pressed.

NOTE

For certain interrupts, such as NMI, it is impossible to control their CPU affinity.

Shielding CPUs for Maximum Determinism and Performance

NightTune allows you to easily shield specific CPUs from processes, interrupts, and shared resource interference from other CPUs.

This is demonstrated as part of the NightSim section in this tutorial. See "Overrun Detection and System Tuning" on page 7-9 for more information.

Conclusion - NightTune

The remaining portion of the tutorial is unrelated to the execution of the **app** program. Terminate the program by executing the following steps:

- Drag the **app** process from the **Process** List panel using the left mouse button to the Kill icon on the toolbar and release.



- Terminate NightTune by selecting Exit from the File menu.

This concludes the NightTune portion of the NightStar RT Tutorial.

NightStar RT Tutorial

7 Using NightSim

NightSim is a graphical tool for scheduling multiple processes in a synchronized manner and monitoring their execution.

NightSim provides a graphical interface to the Frequency Based Scheduler utilities.

If you don't have the Frequency Based Scheduler installed on your system, this portion of the tutorial isn't applicable to you. Use the following command to see if the Frequency Based Scheduler is installed:

rpm -q ccur-fbsched

This chapter of the tutorial also uses a real-time clock interrupt source from the Real-Time Clock and Interrupt Module (RCIM) which is standard equipment on all Concurrent iHawk systems. If your system does not include an RCIM device, this portion of the tutorial isn't applicable to you. Use the following command to see if an RCIM is installed:

cat /proc/driver/rcim/status

If the file shown above does not exist, an RCIM does not exist on your system or your kernel has had the RCIM support removed.

For some aspects of this section, it will be necessary to execute NightSim and NightTune as the **root** user or to ensure that your user account has appropriate privileges. See the "Setting Up User Privileges" on page 1-2 for more information.

Creating FBS Applications

It is trivial to modify cyclic applications so that they may be scheduled via NightSim.

A single API call is required.

The source code for our simplistic work application follows:

```
#include <fbsched.h>
int workload = 1000;
main()
{
    int data = 0;
    int i;
    volatile double d = 1.0;
    while (fbswait()==0) {
        data = !data;
        for (i=0; i<workload; ++i) d = d/d;
    }
}</pre>
```

The call to fbswait() causes the process to block until its next scheduled cycle at which point it returns. The process then performs its workload and then loops to block in fbswait() until its next scheduled cycle.

The **work.c** source file was copied from /**usr/lib/NightStar/tutorial** into the current working directory in an earlier portion of this tutorial.

Compile and link the application using the following command:

cc -g -o work work.c -lccur_fbsched -lccur_rt

Invoking NightSim

A NightSim configuration file has been prepared for this tutorial and should have been copied to your current working directory during the activities in the section entitled "Creating a Tutorial Directory" on page 1-3.

Launch NightSim specifying the configuration file, as show below:

```
nsim -f nsim.config -offline &
```

🛃 NightSim - /zoey/stuff/jeffh/test/tut/nsim.config 📃 🗐 🗌	×
<u>File T</u> arget <u>V</u> iew <u>S</u> cheduler <u>P</u> rocess <u>M</u> onitor <u>T</u> ools <u>H</u> elp	
Server Offline	
Sche <u>d</u> uler Monitor	_
Scheduler	×
Target System: localhost	
Control Definition	
Set up Remove Automatic Configuration Interrupt Source Real-time clock rtc2	
Inactive Scheduler Key 2778 Interrupt Device /dev/rcim/rtc2	
Start Pause Resume Cycles per Frame 4 Clock Period/Freq 0.1 mSec V	
Tasks per Cycle 4 📥 Valid Range (0.001 - 655350.0) mSec	
Refresh Rate 1.00 Hz Permissions 600 (rw)	
Processes transmission and the second s	×
FPID Program CPU IDs Policy Prio Param Limit Ov Halt Deadline DI Halt Origin Schedule	
work 0 FIFO 1 0 false 0 1 2 3	
idle 0 0 1 2 3	i
	1
Key: Idle Scheduled Deadline Unscheduled Refresh Add Copy Edit Remove	
Configuration File: nsim.config	

Figure 7-1. NightSim Initial Window

Creating a Scheduler

NightSim allows you to define the scheduling of multiple processes, using the following parameters:

- The scheduling source (usually an external interrupt)
- The rate at which the interrupts occur (for clock-based interrupts)
- The period at which a process is scheduled
- The CPU affinity, scheduling policy and priority of scheduled processes

Collectively, these parameters define a scheduler.

A cycle is defined as the time between the scheduling sources (interrupts).

A frame is defined by a fixed number of cycles. Frames are useful concepts in many cyclic applications where a series of discrete steps (cycles) must be executed in order before the entire algorithm (frame) repeats.

The scheduler configured by the **nsim.config** file specified on the command line in the previous section defined a scheduler with the following attributes visible on the main window:

- Cycles Per Frame -- four cycles per frame
- Timing Source an interrupt source using RTC2 of the Real-time Clock and Interrupt Module device (RCIM)
- Clock Period -- a cycle time of 100 microseconds
- **Processes** -- a single process, **work**, schedule to run on every cycle of the frame

To view the details of the attributes of the scheduled process, select the **./work** process in the process area at the bottom portion of the **Processes** panel and then press the **Edit**... button in the lower-right portion of the panel.

The Edit Process	dialog	is	disp	layed.
------------------	--------	----	------	--------

Edit Process
Target: localhost
Process <u>C</u> ommand Line
,/work Select
EBS Schedule <u>R</u> untime Properties I/O and <u>D</u> ebug
Starting Cycle 0 1 Period Within Frame
Parameter -1 (Optional)
Soft Overrun Limit 0 Halt FBS if Overrun Limit is Exceeded.
Apply Deadline
Deadline Time mSec -
Deadline Origin Cycle-Relative 💌
Halt on Deadline Violation
OK Reset Cancel Help

Figure 7-2. NightSim Edit Process Dialog

The FBS Schedule tab shows the starting cycle and period of the **work** process. The Staring Cycle defines the cycle within the frame where the process will begin its execution. The Period defines the frequency of execution, in cycles. A period value of 1 causes the application to execute every cycle in the frame.

Click on the Runtime Properties tab in the dialog.

Edit Process
Target: localhost
Process <u>C</u> ommand Line
,/work Select
<u>FBS Schedule</u> <u>R</u> untime Properties I/O and <u>D</u> ebug
Working Directory: . Select
Scheduling Class: First In First Out 💌
Priority: 1 (Range: 1-99)
Available CPUS: Unknown (nsim is offline)
CPU List: 0 (mask=0x1)
OK Reset Cancel Help

Figure 7-3. Runtime Properties Tab

NOTE

The CPU Bias description area of the Runtime Properites tab may vary depending on the number of CPUs on your system.

The Runtime Properties tab allows you to chose the CPU on which execution is allowed, the scheduling policy, and the scheduling priority of the process.

Close the Edit Process window by pressing the Cancel button.

Notice that in addition to the **work** process, the /idle process is listed in the scheduling area of the NightSim window. We have registered the /idle process so that we may subsequently monitor the amount of idle time available for each cycle. The /idle process is not a process that is scheduled, but rather it is a placeholder used to represent idle cycles.

Running the Scheduler

To start the scheduling of the process, press the Setup button followed by the Start button in the Control area.

🖉 NightS	im - /zoey/stuff/jeffh/test/tut/nsim.config	
<u>File Target View Scheduler Process Monitor</u>	<u>T</u> ools <u>H</u> elp	
	Server Online	
Sche <u>d</u> uler <u>Monitor</u>		
	seedeesteere Scheduler menseedeesteere seedeesteere seedeesteere seedeesteere seedeesteere seedeesteere seedee	
Target System: localhost	2	
Control	Definition	
Set up Remove	Automatic Configuration Interrupt Source Re	eal-time clock rtc2 🔍
Running	Scheduler Key 2778 Interrupt Device /dev	v/rcim/rtc2
Start over Pause Resume	Cycles per Frame 4 Clock Period/Freq 0.1	1 mSec 🔻
Frame: 27765	Tasks per Cycle 4 Valid Range (0.0	001 - 655350.0) mSec
Cycle: 2	Tasks per Frame 4	
Refresh Rate 1.00 Hz	Permissions 600 (nv)	
	Processes	·····
FPID Program CPU IDs Policy P	Prio Param Limit Ov Halt Deadline DI Halt Origin	Schedule
0 work 0 FIFQ	1 0 false 0	1 2 3
1 idle 0		1 2 3
Key: Idle Scheduled Deadline	Unscheduled <u>R</u> efresh <u>A</u> dd <u>C</u> op	py <u>E</u> dit <u>R</u> emove
Configuration File: nsim.config		

Figure 7-4. Scheduling Started

Note the Frame count begins to increase under the Control area as the Cycle oscillates between 0 and 3.

To monitor the execution of the process, click on the Monitor tab near the top of the window.

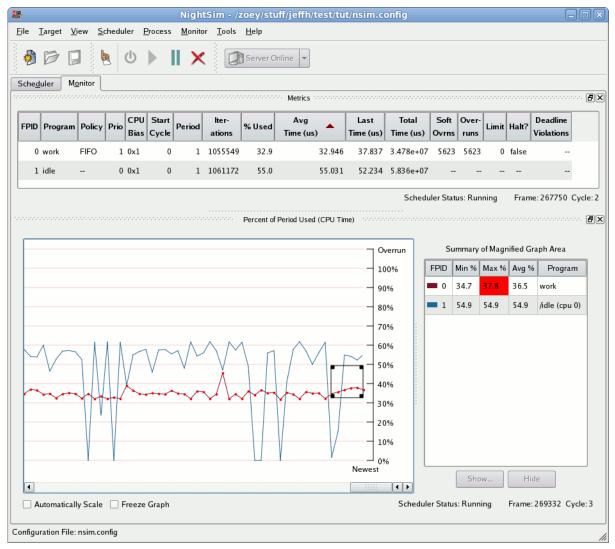


Figure 7-5. NightSim Monitor Page

The NightSim Monitor page provides statistics about each individual process on the scheduler. It includes the PID, program name, CPU bias, number of cycles executed, the CPU times related to per cycle execution, counts of overruns, and the average percentage of the frame used by each process. Additional statistics can be selected for display via the Display Monitor Columns option item of the Monitor menu.

The lower half of the page shows the Percent of Period Used (CPU Time) graph. There is a line for each process on the scheduler; the percent of time used (CPU time) during the last cycle is plotted over time. If an application overruns its timeslot, a red dot is shown on the graph. Points that fall within the square magnifier are detailed in the table to the right. (Note that a process can overrun its deadline even if it doesn't use more than 100% of its alloted CPU time -- other processes could be interfering with it or it may be waiting on I/O, etc.).

Watch the Last Time column. The values displayed are the CPU time used by each process for their last cycle's execution in microseconds. The values attributed to the /idle process indicate the remaining CPU time available within the cycle.

We will adjust the workload of the **work** process and see the effects shown in the Night-Sim Monitor window.

Using Datamon to Modify Program Variables

The Data Monitoring Application Programming Interface is part of the NightStar RT tool set.

Data monitoring allows you to specify executable programs that contain Ada, C, or Fortran variables to be monitored, obtain and modify the values of selected variables by specifying their names, and obtain information about the variables such as their addresses, types, and sizes.

Data monitoring is a powerful capability with a rich API. However, for our purposes, we will write a very simple program which changes the value of a single variable.

Refer to the *Data Monitoring Reference Manual* for more information about data monitoring.

The source code for our **set_workload** program follows:

```
#include <stdlib.h>
#include <stdio.h>
#include <datamon.h>
#define check(x) \setminus
  if((x)) {fprintf(stderr, "%s\n", dm_get_error_string());exit(1);}
main(int argc, char * argv[])
  program_descriptor_t pgm;
  object_descriptor_t
                        obj;
  char buffer[100];
   if (argc != 2) {
      fprintf (stderr, "Usage: set_workload integer-value\n");
      exit(1);
   }
   check(dm_open_program("work",0,&pgm));
   check(dm_get_descriptor("workload",0,pgm,&obj));
   check(dm_get_value(&obj,buffer,sizeof(buffer)));
   check(dm_set_value(&obj,argv[1]));
   printf ("workload: old_value=%s, new_value=%s\n", buffer, argv[1]);
}
```

The dm_open_program function initializes Data Monitoring on the specified process name and PID (in this case zero, which instructs the call to use any process matching the specified name).

The dm_get_descriptor call looks for the specified variable name and returns information about the variable. It also maps the underlying memory page of the variable in the **work** process into the monitoring process.

The dm_get_value and dm_set_value routines return and set the value of the variable using direct memory reads and writes; the **work** process is not affected in any other way than having the value of the workload variable changed.

The **set_workload.c** source file was copied into the current working directory during the activities in "Creating a Tutorial Directory" on page 1-3.

Compile the program using the following command:

```
cc -g -o set_workload set_workload.c -ldatamon -lccur_rt
```

Change the value of the workload variable in the **work** process by issuing the following command:

```
./set_workload 0
```

As shown in the source code above, the program prints the previous value of the workload variable and then sets it to the value specified as an argument to **set_workload**.

The Last Time field for **.**/work is affected by the reduced workload as shown in the NightSim Monitor window.

Experiment with various values of workload using the **set_workload** program until the average Last Cycle time for ./work is approximately 50 microseconds.

Overrun Detection and System Tuning

A scheduling *overrun* occurs when a process's next cycle begins but it has not yet finished execution of its previous cycle.

The NightSim Monitor window includes overrun counts for each process.

It is likely that several overruns have occurred for the work process.

NOTE

If overruns have not yet occurred, place some additional load on the system. Running the following command in a separate terminal session should have the desired effect:

find / -print

The NightTrace tool, as described in a previous chapter, is well suited for determining the specific cause of process overruns. NightTrace kernel tracing provides a detailed view of system activity on all CPUs, including process context switches, interrupts, system calls, and machine exceptions.

For brevity, we will assume that the cause of the overruns is due to additional activities unrelated to the scheduler are occurring on the CPU where **work** executes.

We will use NightTune to shield the CPU associated with our scheduler from other activities.

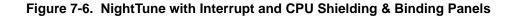
NOTE

If your system only has a single CPU, the remaining portion of this section is inapplicable. Skip to "Shutting Down the Scheduler" on page 7-14 in this case.

Launch NightTune using the **ntune.config** file that was copied into the current working directory during the activities in "Creating a Tutorial Directory" on page 1-3:

6					NightTune	
<u>F</u> ile	<u>V</u> iew <u>M</u> o	nitor	<u>T</u> ools	<u>H</u> elp		
B)		5	💿 🗟 🗢 📚	🖇 🍈 🧇 🕵 Create Panels For: 🛛 » 🗙
raptor Interrupt Activity (Interrupts/Second): CPUS						raptor CPU Shielding and Binding: 000000000 🗐
CPUS						raptor: Intel(R) Xeon(TM) CPU 2.40GHz
0	1000	0	0	/ -	timer	⊨. System ⊜. Chip 0
3	0	0	0		KGDB-stub	🗄 🖞 💋 🧔 CPU 0 [49% Usage]
4	0	0	0		serial	i i i i i i i i i i i i i i i i i i
9	0	0	0		асрі	🗄 🖞 💋 🧔 CPU 1 🛛 [44% Usage]
14	10	0	0		ide0	🗄 🗄 🖞 💋 🧔 CPU 3 [41% Usage]
177	0	0	0		uhci hcd	
185	0	0	0		uhci hcd	
193	0	0	0		uhci hcd	
201	0	0	0		ehci hcd	
209	0	0	0		Intel 82801DB-ICH4	
217	9999	0	0		rcim	
225	289	0	0		eth0	
238	9999	0	0		fbsched	
NMI	0	0	0	0	Non-maskable interrupts	raptor CPU Usage: 1000000000000000000000000000000000000
LOC	999	999	999		Local interrupts	
RES	102	492			Rescheduling interrupts	
CAL	0	0	0		function call interrupts	CPU 2
TLB	2	1	1		TLB shootdowns	
TRM	0	0	0	0	Thermal event interrupts	0 100
SPU	0	0	0		Spurious interrupts	Legend: User System Wait Idle n/a
ERR	0	0	0		Error interrupts	
MIS	0	0	0		APIC errata fixups	
					i	
Legen	id: 🝠 Un	shield	ed 🔰	Shiel	ded 🕑 Inactive 🚭 Bound	

ntune -c ./ntune.config &



A NightTune window appears which displays interrupt activity and the shielding and bound status of all CPUs.

Right-click on the System icon in the CPU Shielding & Binding panel and select Expand All from the context menu.

Note that work process is listed in the Bound Processes list of CPU 0.

Take the following actions to bind the RCIM interrupt to CPU 0 and shield CPU 0 from all other activities:

- While the cursor is positioned in the Interrupt panel over the cell in the Description column which contains the word **rcim**, press and hold the left mouse button, then drag the interrupt onto the CPU 0 row in the CPU Shielding and Binding panel, and release the mouse button. The **rcim** interupt is now bound to CPU 0.

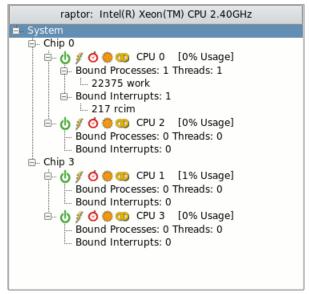


Figure 7-7. Process and Interrupt Bound to CPU 0

- Right-click the row corresponding to CPU 0 in the CPU Shielding and Binding panel and select the Change Shielding... option from the context menu.

💌 raptor: CPU - Shielding
Property CPU - Shielding Click on icons to change CPU shielding Click on icons to change CPU shielding Chip 0 CPU 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
OK Apply Reset Cancel Help

Figure 7-8. Change Shielding Dialog

- Click the Maximize Shielding icon in the CPU 0 line (the maximize shielding icon is the right-most icon with three overlapping shield figures).

💙 raptor: CPU - Shielding	×							
Click on icons to change CPU shielding								
Setting differ from system								
🗄 System								
Chip 0								
- CPU 0 👌 💞 🦁 🦁 🗿 - CPU 2 👌 🖇 🗇 🚳 🖏								
⊡ CP0 2 0 9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0								
- CPU 1 (b) ∮ ♂ ♦ 00 (5) - CPU 3 (b) ∮ ♂ ♦ 00 (5)								
CPU 3 👌 🖋 🧿 🎒								
OK Apply Reset Cancel Help]							
	-							

Figure 7-9. Shielding Changes Pending

The CPU 0 line changes its display to indicate that all processes and interrupts other than **work** and **rcim** will be shielded from CPU 0. Additionally, the sibling hyper-threaded CPU (in this case CPU 2 as shown below CPU 0) is marked down so that hyper-threaded execution on CPU 2 does not interfere with CPU 0.

NOTE

The hyperthreaded sibling of CPU 0 may be a logical CPU number other than CPU 2.

NOTE:

Your system may not support hyper-threading or it may not have hyper-threading enabled in which case the CPUs are not displayed in hyper-threaded groups.

- Press the OK button to activate the shielding changes.

Return to the NightSim Monitor window and watch the Overrun column. It is likely that overruns have ceased to occur. Clear the overrun count by selecting the Clear Performance Data option item from the Monitor menu. This action resets all the statistics to zero.

Watch the Overrun column to see if any overruns still occur.

Experiment with the **./set_workload** program to make the workload of the **./work** application such that only 35 microseconds are left for **/idle** processing.

If the system is properly configured, the scheduler should continue to execute without any overruns on the shielded CPU.

Shutting Down the Scheduler

Return to the NightSim window and press the Remove button to terminate the scheduler. Press Yes when presented with the dialog which asks whether to kill the processes associated with the scheduler.

Exit NightSim by selecting the Exit menu item from the File menu. A dialog asking whether or not to save changes to **nsim.config** may appear; if so, press No.

You may also wish to clear the shielding attributes for CPU 0 and return the system to its previous state using NightTune.

Exit NightTune by selecting the Exit from the File menu.

This concludes the NightSim portion of the NightStar RT Tutorial.

A Tutorial Files

The following sections show the source listings for the files used in the *NightStar RT Tutorial*.

- api.c
- app.c
- function.c
- report.c
- set_workload.c
- set_rate.c

api.c

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <fcntl.h>
#include <errno.h>
#include <string.h>
#include <nprobe.h>
int cycles = 0;
int overruns = 0;
char * sample;
// Perform the work of consuming a single Data Recording
sample from NightProbe.
11
int
work (FILE * ofile, np_handle h, np_header * hdr) {
  np item * i;
   int status;
   int which;
   // Read one sample, which may contain data for multiple
processes
  // and variables.
   11
   status = np_read (h, sample);
   if (status <= 0) {
      return status;
   }
   cycles++;
   fprintf (ofile, "Sample %d\n", cycles);
   for (i = hdr->items; i; i = i->link) {
      char buffer [1024];
      sprintf (buffer, "item: %s:", i->name);
      fprintf (ofile, "%-30s", buffer); // Nice formatting :-
)
      // Display the value of each item.
      // For arrays, format each individual item.
      11
      for (which = 1; which <= i->count; ++which) {
         char * image = np_format (h, i, sample, which);
         if (image != NULL) {
            fprintf (ofile, " %s", image);
```

```
} else {
            fprintf (ofile, "\n<error: %s>\n", np_error (h));
            return -1;
         }
        free (image);
      }
      fprintf (ofile, "\n");
   }
  fflush (ofile);
  return 1;
}
int
main (int argc, char *argv[])
{
  np_handle h;
  np_header hdr;
  np_process * p;
  np_item * i;
  int fd;
  int status;
  FILE * ofile = stdout;
  fd = 0; // stdin
  status = np_open (fd, &hdr, &h);
  if (status) {
     fprintf (stderr, "%s\n", np_error (h));
      exit(1);
   }
  sample = (char *) malloc(hdr.sample_size);
  if (sample == NULL) {
      fprintf (stderr, "insufficient memory to allocate
sample buffer\n");
     exit(1);
   }
   for (p = hdr.processes; p; p = p->link) {
      if (p->pid >= 0) {
         fprintf (ofile, "process: %s (%d)\n", p->name, p-
>pid);
      } else {
         fprintf (ofile, "resource: %s (%s)\n", p->name, p-
>label);
      }
   }
  fprintf (ofile, "\n");
```

```
for (i = hdr.items; i; i = i->link) {
      fprintf (ofile, "item: %s (%s), size=%d bits, count=%d,
type=%d\n",
             i->name, i->process->name, i->bit_size, i-
>count, i->type);
   }
  fprintf (ofile, "\n");
   for (;;) {
     status = work (ofile, h, &hdr);
      if (status <= 0) break;</pre>
   }
   fprintf (ofile, "Data Recording done: %d cycles fired, %d
overruns\n",
            cycles, overruns);
   if (ofile != stdout) {
    fclose (ofile);
   }
   if (status < 0) {
      fprintf (stderr, "%s\n", np_error (h));
   }
  np_close (h);
   // At this point, file descriptor 0 remains open, but is
no
  // longer a NightProbe Data File/Stream.
}
```

app.c

```
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <pthread.h>
#include <errno.h>
#include <ntrace.h>
#include <math.h>
#include <sys/ipc.h>
#include <sys/sem.h>
static void * heap thread (void * ptr);
extern void work (int control);
typedef struct {
  char * name;
  int
         count;
  double delta;
  double angle;
  double value;
} control_t;
control_t data[2] = { { "sin", 0, M_PI/360.0, 0.0, 0.0 },
                       { "cos", 0, M_PI/360.0, 0.0, 0.0 } };
enum { run, hold } state;
int rate = 50000000;
int sema;
extern double
FunctionCall(void)
{
   return data[0].value + data[1].value;
}
void *
sine_thread (void * ptr)
{
   control t * data = (control t *)ptr;
   struct sembuf wait = \{0, -1, 0\};
   work(1);
   trace_set_thread_name (data->name);
   for (;;) {
     semop(sema, &wait, 1);
      data->count++;
     data->angle += data->delta;
      data->value = sin(data->angle);
   }
}
```

```
void *
cosine_thread (void * ptr)
{
   control t * data = (control t *)ptr;
   struct sembuf wait = \{0, -1, 0\};
  work(1);
   trace set thread name (data->name);
   for (;;) {
      semop(sema, &wait, 1);
      data->count++;
      data->angle += data->delta;
      data->value = cos(data->angle);
   }
}
int
main (int argc, char * argv[])
{
  pthread_t thread;
  pthread_attr_t attr;
   struct sembuf trigger = \{0, 2, 0\};
   trace_begin ("/tmp/data",NULL);
   sema = semget (IPC_PRIVATE, 1, IPC_CREAT+0666);
  pthread attr init(&attr);
   pthread_create (&thread, &attr, sine_thread, &data[0]);
   pthread_attr_init(&attr);
   pthread_create (&thread, &attr, cosine_thread, &data[1]);
  pthread_attr_init(&attr);
   pthread_create (&thread, &attr, heap_thread, NULL);
   for (;;) {
      struct timespec delay = { 0, rate } ;
      nanosleep(&delay,NULL);
      work (random() % 1000);
      if (state != hold) semop(sema,&trigger,1);
   }
   trace_end () ;
}
void * ptrs[5];
static void *
heap_thread (void * unused)
{
```

```
int i = 5;
   int scenario = -1;
  void * ptr;
  int * * iptr;
  extern void * alloc_ptr (int size, int swtch);
  extern void free_ptr (void * ptr, int swtch);
   for (;;) {
      sleep (5);
      switch (scenario) {
      case 1:
         // Use of freed pointer
         ptr = alloc_ptr(1024,3);
         free_ptr(ptr,2);
         memset (ptr, 47, 64);
         break;
      case 2:
         // Double-free
        ptr = alloc ptr(1024,3);
         free_ptr(ptr,2);
         free(ptr);
         break;
      case 3:
         // Overwriting past end of an allocated block
#define MyString "mystring"
        ptr = alloc ptr(strlen(MyString),2);
         strcpy (ptr,MyString); // oops -- forgot the zero-
byte
         break;
      case 4:
         // Uninitialized use
         iptr = (int * *) alloc_ptr(sizeof(void*),2);
         if (*iptr) **iptr = 2778;
         break;
      case 5:
         // Leak -- all references to block removed
        ptr = alloc ptr(37,1);
        ptr = 0;
         break;
      case 6:
         // Some more allocations we'll check on...
         ptrs[0] = alloc_ptr(1024*1024,3);
         ptrs[1] = alloc_ptr(1024,2);
         ptrs[2] = alloc_ptr(62,1);
         ptrs[3] = alloc_ptr(4564,3);
         ptrs[4] = alloc_ptr(8177,3);
        break;
      }
      (void) malloc(1);
      scenario = 0;
   }
}
```

```
void * func3 (int size, int count)
{
  return malloc(size);
}
void * func2 (int size, int count)
{
  if (--count > 0) return func3(size,count);
  return malloc(size);
}
void * func1 (int size, int count)
{
  if (--count > 0) return func2(size,count);
  return malloc(size);
}
void free3 (void * ptr, int count)
{
  free(ptr);
}
void free2 (void * ptr, int count)
{
   if (--count > 0) {
     free3(ptr,count);
     return;
   }
   free(ptr);
}
void free1 (void * ptr, int count)
{
   if (--count > 0) {
      free2(ptr,count);
     return;
   }
  free(ptr);
}
void * alloc ptr (int size, int count)
{
  return func1(size,count);
}
void free_ptr (void * ptr, int count)
{
  free1(ptr,count);
}
void work (int control)
{
  volatile double calculations[2048];
```

```
volatile double d = 0.0;
   int i;
   for (i=0; i<2048; ++i) {</pre>
      calculations[i] = 3.14159;
   }
   for (i=0; i<control*10; ++i) {</pre>
      d = d*d;
      calculations[i%2048] = d;
   }
}
#include <signal.h>
int nosighup (void)
{
   struct sigaction ignore;
   ignore.sa_flags = 0;
   ignore.sa_handler = SIG_IGN;
   sigemptyset(&ignore.sa_mask);
   sigaction(SIGHUP,&ignore,NULL);
}
```

function.c

```
double
FunctionCall(void)
{
   static double counter;
   return counter++;
}
```

report.c

```
#include <stdio.h>
void report (char * caller, double value)
{
   static int count;
   if (++count % 40) printf ("The value from %s is %f\n",
   caller, value);
}
```

set_workload.c

```
#include <stdlib.h>
#include <stdio.h>
#include <datamon.h>
#define check(x) if((x)) {fprintf(stderr, "%s\n",
dm_get_error_string());exit(1);}
main(int argc, char * argv[])
{
  program_descriptor_t pgm;
  object_descriptor_t obj;
  char buffer[100];
   if (argc != 2) {
      fprintf (stderr, "Usage: set_workload integer-
value\n");
      exit(1);
   }
   check(dm_open_program("work",0,&pgm));
   check(dm_get_descriptor("workload",0,pgm,&obj));
   check(dm_get_value(&obj,buffer,sizeof(buffer)));
   check(dm_set_value(&obj,argv[1]));
   printf ("workload: old_value=%s, new_value=%s\n", buffer,
argv[1]);
}
```

set_rate.c

```
#include <stdlib.h>
#include <stdlib.h>
#include <stdlib.h>
#include <datamon.h>

#define check(x) if((x)) {fprintf(stderr, "%s\n",
dm_get_error_string());exit(1);}
main(int argc, char * argv[])
{
    program_descriptor_t pgm;
    object_descriptor_t pgm;
    object_descriptor_t obj;
    char buffer[100];

    if (argc != 2) {
        fprintf (stderr, "Usage: set_rate: integer-value\n");
        exit(1);
    }
```

```
check(dm_open_program("app",0,&pgm));
check(dm_get_descriptor("rate",0,pgm,&obj));
check(dm_get_value(&obj,buffer,sizeof(buffer)));
check(dm_set_value(&obj,argv[1]));
printf ("rate: old_value=%s, new_value=%s\n", buffer,
argv[1]);
}
```

NightStar RT Tutorial