

NightStar LX Tutorial

Version 4.3



0897009-080 February 2013 Copyright 2013 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.

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 Assistant, a Qt[®] utility. Qt is a registered trademark of Digia Plc and/or its subsidiaries.

NVIDIA[®] CUDATM is a trademark of NVIDIA Corporation.

Preface

General Information

NightStar LXTM allows users running Linux[®] to schedule, monitor, debug and analyze the run-time behavior of their time-critical applications as well as the Linux operating system kernel.

NightStar LX consists of the NightTraceTM event analyzer; the NightProbeTM data monitoring tool, the NightViewTM symbolic debugger, and the NightTuneTM system and application tuner.

Scope of Manual

This manual is a tutorial for NightStar LX.

Structure of Manual

This manual consists of six chapters and an appendix which comprise the tutorial for NightStar LX.

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:

0897395	NightView TM User's Guide	
0897398	NightTrace™ User's Guide	
0897465	NightProbe [™] User's Guide	
0897515	NightTune [™] User's Guide	

Contents

Chapter 1 Overview

Getting Started	. 1-2
Creating a Tutorial Directory	. 1-2
Building the Program	. 1-2

Chapter 2 Panels

Moving Panels	2-2
Fabbed Panels	2-6
Context Menus.	2-8
Futorial 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
Scenario 4: Use of Uninitialized Heap Blocks
Scenario 5: Detection of Leaks
Scenario 6: Allocation Reports
Disabling Heap Debugging 3-21
Debugging Multiple Threads
Traversing Linked Lists
Using Monitorpoints
Using Eventpoint Conditions and Ignore Counts 3-33
Using Patchpoints
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 4-7
Zooming
Moving The Interval 4-9
Using the Events Panel for Textual Analysis 4-10
Customizing Event Descriptions 4-11

Searching the Events List
Halting the Daemon
Using States
Displaying State Duration
Generating Summary Information
Defining a Data Graph
Using the NightTrace Analysis API
Automatically Tracing Your Application
nlight Wizard - Selecting Programs
nlight Wizard - Defining Illuminators
nlight Wizard - Selecting Illuminators
nlight Wizard - Relinking the Program
nlight Wizard - Activating Illuminators
Running the Program
Analyzing Application Illumination Events
Summarizing Workload Performance
Batch Summary of Functions
Shutting Down
Conclusion - NightTrace

Chapter 5 Using NightProbe

Invoking NightProbe	-1
Selecting Processes	-2
Viewing Live Data	-4
Modifying Variables	-5
Selecting Variables for Recording and Alternative Viewing	-7
Selection of Views	-8
Table View	-8
Graph View	11
Sending Probed Data to Other Programs	15
Conclusion - NightProbe 5-1	19

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
Conclusion - NightTune

Appendix A Tutorial Files

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

Illustrations

Figure 2-1. Viewing Page with List & Graph Panels	2-2
Figure 2-2. Panel Detaches from Page	2-3
Figure 2-3. Panel Movement in Progress	2-4
Figure 2-4. Graph Panel on Top of List Panel	2-5
Figure 2-5. Table View added to Page	2-6
Figure 2-6. Panel in Motion Creating Tab	2-7
Figure 3-1. NightView Main Window	3-2
Figure 3-2. app Program Loaded	3-4
Figure 3-3. NightView Debug Heap Dialog	3-7
Figure 3-4. Heap Totals and Configuration	.3-10
Figure 3-5. info memory Command Output	.3-13
Figure 3-6. Heap Error Description	.3-14
Figure 3-7. Heap Leaks Display	.3-18
Figure 3-8. Still Allocated Blocks Display	.3-20
Figure 3-9. Context Panel With Stack Frames Expanded	.3-22
Figure 3-10. Run Mode Selector	.3-23
Figure 3-11. Pointer to Linked List Expanded	.3-25
Figure 3-12. Dialog Selecting Linked List Component	.3-26
Figure 3-13. Pointer Variable Displayed As Linked List	.3-26
Figure 3-14. Filter Dialog	.3-27
Figure 3-15. Filtered Linked List	.3-28
Figure 3-16. Filtered Linked List Expanded	.3-28
Figure 3-17. Monitorpoint Dialog	.3-30
Figure 3-18. NightView Monitor Panel	.3-31
Figure 3-19. Patchpoint Dialog	3-35
Figure 3-20. Result of Patching in Call to Newly Loaded Function	.3-38
Figure 3-21. Tracepoint Dialog	.3-40
Figure 4-1. NightTrace Main Window	4-2
Figure 4-2. Import Daemon Definitions Dialog	4-3
Figure 4-3. Logging Data	4-4
Figure 4-4. app_data Page	4-5
Figure 4-5. NightTrace Timeline	4-7
Figure 4-6. Timeline Interval Panel	4-9
Figure 4-7. Events Panel	.4-10
Figure 4-8. Add Event Description dialog	.4-11
Figure 4-9. Searching using the Profiles Dialog	.4-13
Figure 4-10. Browse Events Dialog	.4-14
Figure 4-11. Events Panel After Search	.4-15
Figure 4-12. Timeline Panel After Search	.4-16
Figure 4-13. Profiles Dialog With Obtuse Profile Selected	.4-17
Figure 4-14. Timeline Editing	.4-18
Figure 4-15. Edit State Graph Profile dialog	.4-19
Figure 4-16. Sine State in Timeline	.4-21
	1 00
Figure 4-17. Summary Results Page	.4-23
Figure 4-17. Summary Results Page	.4-23
Figure 4-17. Summary Results Page Figure 4-18. Summary Graph Figure 4-19. State Durations Graph Modified	.4-23 .4-24 .4-25
Figure 4-17. Summary Results Page Figure 4-18. Summary Graph Figure 4-19. State Durations Graph Modified Figure 4-20. Timeline in Edit Mode	.4-23 .4-24 .4-25 .4-26
Figure 4-17. Summary Results Page Figure 4-18. Summary Graph Figure 4-19. State Durations Graph Modified Figure 4-20. Timeline in Edit Mode Figure 4-21. Adding a Data Graph	.4-23 .4-24 .4-25 .4-26 .4-27

.....

	4 00
Figure 4-23. Display Page with Data Graph	4-29
Figure 4-24. Export Profiles to Night Trace API Source File dialog	4-30
Figure 4-25. nlight Wizard - Select Programs Step	4-33
Figure 4-26. nlight Wizard - Define Illuminators Step	4-35
Figure 4-27. nlight Wizard - Select Illuminators Step	4-37
Figure 4-28. nlight Wizard - Relink Programs Step	4-39
Figure 4-29. nlight Wizard - Activate Illuminators Step	4-41
Figure 4-30. Night Trace - Import File Name	4-43
Figure 4-31. Night Trace - Daemon Ready to Launch	4-44
Figure 4-32. Night Frace - Daemon Collection Events	4-45
Figure 4-33. Night Trace - /tmp/data_import Timeline	4-46
Figure 4-34. Night Trace - Events Panel w/ Tool Tip	4-47
Figure 4-35. NightTrace - Event Panel Search Dialog	4-48
Figure 4-36. NightTrace - Events Panel after Search	4-48
Figure 4-37. NightTrace - Events Panel Context Menu	4-49
Figure 4-38. Night Trace - Launches Editor with Source File at Line Number	4-50
Figure 4-39. NightTrace - Functions Summary Table	4-51
Figure 4-40. Function Details Table for the work function	4-52
Figure 5-1. NightProbe Main Window	. 5-2
Figure 5-2. Program Selection Dialog	. 5-3
Figure 5-3. Process Selection Dialog	. 5-3
Figure 5-4. NightProbe Browse Panel	. 5-4
Figure 5-5. Expanded Data Item	. 5-5
Figure 5-6. Variable Modification in Progress	. 5-6
Figure 5-7. Mark and Record Attributes Set	. 5-7
Figure 5-8. Table View	. 5-9
Figure 5-9. Item Selection Dialog	5-10
Figure 5-10. Table in Automatic Sampling Mode	5-11
Figure 5-11. Graph Panel	5-12
Figure 5-12. Graph Panel Actively Displaying Values	5-13
Figure 5-13. Edit Curve Attributes Dialog	5-13
Figure 5-14. Graph Panel with Modified Curves	5-14
Figure 5-15. Recording area of Configuration Page	5-16
Figure 5-16. Clock Selection Dialog	5-16
Figure 5-17. Record To Program Dialog	5-17
Figure 5-18. Recording Area of Configuration Page w/ Destination	5-18
Figure 5-19. Example Output of Graph Program	5-19
Figure 6-1. Night lune initial panels	. 6-1
Figure 6-2. Expanded Process List	. 6-2
Figure 6-3. Process List with Inreads	. 0-3
Figure 6-4. Strace Output of Inread	. 6-4
Figure 6-5. Process Details Dialog	. 6-5
Figure 6-6. Process Memory Details Page	. 0-0
Figure 6-7. File Descriptors Page	. 6-8
Figure 6-8. Signals Page	. 6-9
Figure 0-9. Process Scheduler Dialog	0-10
Figure 0-10. Night Lune Process List with modified thread	0-11
Figure 0-11. CPU Shielding and Binding Panel	6-12
Figure 0-12. CPU Snielding and Binding Panel with Bound Thread	0-15
Figure 0-15. Night Lune with Interrupt Activity Panel	0-15
Figure 0-14. Interrupt Affinity Dialog	6-16

NightStar LXTM is an integrated set of debugging tools for developing time-critical Linux[®] applications. NightStar LX 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 LX tools consist of:

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

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

NightStar LX operates with the standard Linux kernel. Certain features that are available in the NightStar RT product are not available in NightStar LX because they require kernel features not available in the standard Linux kernel.

Getting Started

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 LX. 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_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 LX 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 LX 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 LX 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
- traversing linked lists
- searching segments of memory

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 Debu	igger 📃 🗆 🔍
<u>File View Shell Process Source Eventpoint Data Tools Help</u>	
	ei 💷 di ∉ 👓 🍂 🗶 🔚 🎋 🔳 🛍
Source	Eventpoints
	Context Locals Messages NightView 7.4(nview-riq11756A), Linked Mon Jan 28 16:49:19 EST 2013 In case of confusion, type "help" /usr/lib/NightView-7.4/ReadyToDebug ~jeffh/.bash_init complete. zippy> /usr/lib/NightView-7.4/ReadyToDebug
Command:	Run All Threads
No process	li li

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. You can disable the welcome screen for subsequent invocations using the checkbox in the lower left corner of that screen. If the screen appears, 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 1-2.

- 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 High			
Common Errors Detection				
Block Overrun Dangling Point	ter Uninitialized Field			
-General Settings		Error Control		
Hardware overrun protection			Stop	Print
Emcify check been freq		free fill modified	×	×
Specify check heap freq		free not at beginning	×	×
Specify retained free blocks	100	free unallocated	×	×
Specify heap size	Unlimited heap size	malloc zero		
Class size		memalign not power of 2		
Stop Size	0	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				
- Malles mare Malles fill h		Den forman fill hades: Ox bf		-
FIII manoc space Manoc III b	iyte: Uxc5	Pre-tence mi byte: 0xbi		-
Fill free space Free fill by	te: 0xc3	Post-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.

tem	Value
🗄 🖞 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
	1

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
I	-1086326084
iptr	0
	0x8102c68
넖 🛶 🕂 (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
🗄 🕞 unused	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.

-1086326084
100002.0001
0
0x8102c68
allocated
0x08102c68 0x08102c6f
8 bytes
1 (as of last heap check)
post-fence modified in block (value=0x8102c68)
0x080485ec in func2() at app.c line 168
3
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.

Data Ministration Contraction			
Item	Value		
🚊 👔 Heap Information	local:29316		
i∰… <u>Σ</u> Totals			
🗄 🖌 Configuration	3		
i⊟… 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			
⊕ φ→ leak set, 16 bytes	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.

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

	seesseese. Data tottottottottottottottottottottottottot
Item	Value
⊕. Leak Sets	local:29316: new at heap operation 10
🗄 🖞 Heap Information	local:29829
Provide the set of	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
🖶 ?🔶 allocated set, 8177 bytes	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
🖶 ? → allocated set, 1024 bytes	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>
	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>
🖶 ?+ allocated set, 62 bytes	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>
🖶 ? → allocated set, 37 bytes	1 block of 37 bytes, 0x08048dc3 in func1() at app.c line 174
	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>
? 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, 0xb8050769 at <xt_thread_setup+29></xt_thread_setup+29>
	2 blocks of 1 bytes, 0x08048d38 in heap_thread() at app.c line 155

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

- Disable all overhead associated with heap debugging, issue the following command:

heapdebug off

- 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

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

Debugging Multiple Threads

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

- Set a breakpoint on line 47 by issuing the following command:

b 47

- Resume the process by clicking on the resume icon or by issuing the following command:

resume

The process will run until one of the threads reaches the breakpoint on line 47.

- 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

Item	Threads for local:9573 app
🚊 🎲 9413	C thread 0x2b65d0e64b20
	" #0 0x00400e1b in main(int argc = 1, char ** argv = 0x7fffda9118f8) at app.c line 93
	C thread 0x2b65d1c7a700 (sine thread)
	#0 0x00400c26 in sine thread(void * ptr = 0x6020c0) at app.c line 47
📄 🕞 data	0x6020c0 struct *
🕴 🗄 🕞 ptr	0x6020c0
🗄 🖬 wait	struct sembuf
i	C thread 0x2b65d1e7b700 (cosine_thread)
🗄 🔅 🥵 9575	C thread 0x2b65d207c700 (heap_thread)

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 parenthesis. 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 NightView using the **set-thread-name** command.

You can switch to the context of other threads by clicking on the thread of interest. When you click on a thread, the source displayed in the NightView main window changes to location where that thread is executing.

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.

NightView provides a Run Mode which specifies how threads are resumed and stopped. By default, the Run Mode is Run All Threads. Thus 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.

If you change the Run Mode to Run One Thread, then when you resume a thread, it is the only one that runs. All stopped threads remain stopped.

On one of the toolbars, you will see a option list which represents the current run mode. By default, this item is at the bottom of the screen to the right of the **Command** area.

Run All Threads 🛛 🔫

Figure 3-10. Run Mode Selector

- Change the mode to Run One Thread by clicking on the list and selecting that mode.
- Click the Next icon stops on line 46, the call to semop().
- Now click the Next icon one more time.

Notice that the Next operation does not complete. This is because we were only allowing a single thread to execute, and the thread is blocked in the semop() call, waiting for another thread to unblock it (the main thread).

- Press the Interrupt icon

to cancel the Next operation.

NOTE

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

Also, some systems may have debug versions of glibc installed, in which case NightView may show you source code inside semop(), or routines it calls. Regardless, you will likely be presented with a gray triangular arrow, as described below, unless you are stopped at the lowest level, a system call.

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

clear app.c:47

- Change the Run Mode to Run All Threads.
- 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!

Traversing Linked Lists

NightView's data display panels allow you to view variables, indirect through pointers, and expand or collapse levels of detail. Variables are presented in a tree to facilitate view-ing.

NightView provides two features which make viewing complex data structures easier: linked lists and filtering.

Our application has created a list of structures which are linked via a member of each structure. The variable head represents the start of this linked list.

For simplicity, we will remove the existing data panel before proceeding with this section.

- Raise the existing data panel and then close it by clicking the close icon in the upper-right of the panel's control area.
- Add the variable head to a new data panel by typing the following command:

data head

A new data panel now appears and contains the pointer variable head.

- Expand the pointer variable and the link pointer member of it, and several of its children.

nonnennennennennennennen Data herrennennennennennennen 🗐 🛛)
Item	Value		1
🖻 🕞 head	0x804c268 struct node_t *		1
- 💽 value	[~] 6		1
🗄 🕞 link	0x804c278 struct node_t *		1
···· 🔳 value	7		1
📥 🕞 link	0x804c288 struct node_t *		1
🔳 value	8		1
	0x804c298 struct node_t *		
			J

Figure 3-11. Pointer to Linked List Expanded

As shown in the figure above, each node in the linked list is nested under the previous node in the list. While this is a fine representation, it becomes cumbersome once you display more than just a few nodes.

As an alternative, you can tell NightView that the pointer is a member of a linked list.

- Right-click on the head variable and select Treat As Pointer To Linked List...

A small dialog is presented which allows you to specify the member of the structure which defines the next element in the list.

Enter an expression that, given an element, produces the address of the next element. \$p is a temporary convenience variable with the address of each element in turn.		
Linked List Expression:		
\$p->link		
OK Cancel Help		

Figure 3-12. Dialog Selecting Linked List Component

NightView automatically populates a drop-down list with all members which have types appropriate for indicating a link in a list. In our case, it has correctly chosen the member which identifies the next node in the list.

- Press OK

The head variable in the data panel is now displayed using an alternative method.

nanananananananananananan Data Manananananan BX		
Item	Value	
📴 🕞 head	0x804c268 struct node_t * {linked list: \$p->link [0:3]}	
iģ… 💽 -> [0th]	0x804c268 struct node_t *	
🕂 🕞 -> [1st]	0x804c278 struct node_t *	
i	0x804c288 struct node_t *	
🕂 🕞 -> [3rd]	0x804c298 struct node_t *	
···· 🔳 value	9	
🗄 🕞 link	0x804c2a8 struct node_t *	
L 🛛	5	

Figure 3-13. Pointer Variable Displayed As Linked List

In the figure above, the various nodes in the linked list are displayed at the same level and are numbered, starting from 0.

- Click on the guard symbol (blue triangle) several times until the "3rd" node is shown and then expand it to match the figure above.

NightView will allow you to expand the list as long as the member that you selected above that defines the next item in the list is not NULL. You can also use the context-menu to tell NightView how many nodes in the list to display (as opposed to continuing to click on the guard symbol to extend the list).

Often when viewing a linked list you may want to identify a particular node in the list. We will use NightView's filtering capability to do this.

- Right-click on the head variable and select Filter Elements with a Condition...

The following dialog appears which allows you to type in an expression which defines the nodes in the list to be shown.

Enter a condition exp the condition will be s variables are set as th in turn:	ression. Only the elements that match hown. Some temporary convenience e condition is evaluated for each element	
\$i has the index of the element. Example: my_array[\$i] < 5 \$p has the address of the element. Example: *\$p < 5 \$v refers to the element. Example: \$v < 5		
Clear the text field to remove the filter.		
Condition Filter Expres	ssion:	
\$p->value % 7 == 0	▼ Clear	
When looking for each underlying elements	n filtered element, how many of the should the filter check?	
Search Limit:	1024	
	OK Cancel Help	

Figure 3-14. Filter Dialog

The expression can include several special built-in variables which aid you in specifying the filter. The text in the dialog explains these variables: \$i, \$p, and \$v.

- Type in the following text in the Condition Filter Expression field, as shown in the figure above, and the press OK.

\$p->value % 7 == 0

nan an		
Item	Value	
🖃 🕞 head	0x804c268 struct node_t * {linked list: \$p->link filtered: \$p->	
⊕ [1st]	0x804c278 struct node_t *	
Click to search Search at 4 for up to 1024 elements.		

We have told NightView to only show us nodes in the list whose member value is a multiple of seven.

Figure 3-15. Filtered Linked List

Initially, the first node in the list matching the filter condition is shown -- it is node # 1, the second node in the list (node numbering starts at 0).

- Expand the filtered list by clicking the guard symbol two times, and then expand all three filtered elements to match the figure below:

	nanananan Data manananananananananan 🗗 🗙
Item	Value
🖃 🕞 head	0x804c268 struct node_t * {linked list: \$p->link filtered: \$p->
⊡… 💽 -> [1st]	0x804c278 struct node_t *
- alue	7
🗄 🗤 🕟 link	0x804c288 struct node_t *
[8th]	0x804c2e8 struct node_t *
🔳 value	14
🖽 🕞 link	0x804c2f8 struct node_t *
🖕 🕞 -> [15th]	0x804c358 struct node_t *
🔳 value	21
🗄 🕁 link	0x804c368 struct node_t *
🗆 🔝 Click to search	Search at 16 for up to 1024 elements.
•	

Figure 3-16. Filtered Linked List Expanded

See that all nodes shown in the list have a value member which is a multiple of seven, which satisfies the filter expression we specified above.

Notice that an ellipsis follows each node number when the next node in the list is not consecutive, indicating that there are gaps in the displayed list due to filtering. The description field of the head of the linked list also indicates filtering is active.

You can use NightView's filtering capability on arrays as well as linked lists. In fact, you can use it to search through memory for a particular value. Just add a pointer value to the

data panel, tell NightView to treat it as an array using the context menu, and then apply a filter expression.

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	lonitorpoint	×
Location: app.c:46		
	Options:	
	Enable	
	🔿 Enable, disable a	after next hit
	 Disable 	
Condition If:	~	
Ignore Count: 0		
Name:		
Expressions		
New		
Expression	Format	Label
	default	
	OK Cancel	Help

Figure 3-17. 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.

nonconcentration Monitor Monitor Alexandration	
Item	Value (1000 ms between samples)
sine count	✓ 3344
sine value	-0.788010753607640
cosine count	✓ 3344
cosine value	-0.615661475324483
1	

Figure 3-18. 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 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:

```
1 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 🔲 🕅	3
≡→ Location: app.c:48	
Options	
Enable	
\bigcirc Enable, disable after next hit	
O Disable	
Condition If:)
Ignore Count: 0	
Name:]
Action	1
Insert an expression at this location	
 Branch to a different location 	
○ Set thread local tag values	
Evaluate:	
OK Cancel Help	j

Figure 3-19. 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
```

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	
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		
		•

Figure 3-20. 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.

8	Set New Tracepoint	
🕑 Location: 🛛	app.c:48	
	_ Options	
	Enable	
	C Enable, disable after next hit	
	O Disable	
Condition If:		
Ignore Count:	0	
Name:		
-NightTrace Ev	/ent	
ID:		•
Value:		
	OK Cancel Help	

Figure 3-21. 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. So simply iconfiy the NightView window and do not exit from NightView.

Conclusion - NightView

This concludes the NightView portion of the NightStar LX Tutorial.

NightStar LX 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 RT can also log kernel events such as individual system calls, context switches, machine exceptions, page faults and interrupts, but no kernel support exists for this in standard Linux. 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 3-39 before beginning this section of the tutorial (and resume execution of the pro-

cess).

<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> rofiles Ti <u>m</u> elines <u>T</u> ools <u>H</u> elp		
🗁 📮 😤 🗢 📨 🐨 🔊 🔎 🔎 🗩 🗴 🔠 💽 🐾	н н 🔛 🚥	
home because and the second seco		aaaaaaa B x
Type Daemon Target Logged Lost S	itate Attached	Buffer
—K— kernel_trace_to_gui raptor H	alted	
▲ Launch ▲ Esume ▲ Pause ▲ Halt Elush Display Trigge	rs <u>E</u> nable Eve	nts Delete
Trace Segments		
Type 🔻 Trace Segment Target Logged Lost Duration (sec) Un	aved	
ſ	Save Trace Data	lose Trace Data
l	Save mace Data	ose made 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.

NOTE

The number of events shown in the **Trace Segments** panel will normally differ from the number of events shown in the **Daemons** 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.

		Import Daemon Definitions		×
Target narf	Refresh List			
Program ID 🔺	Program	User	Key File	
32498	арр	jeffh	/tmp/data	
		Im	port 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 3-39.

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

			oppo Daemons				8(
Туре	Daemon	Target	Logged	Lost	State	Attached	Buffer
—U—	app_data	zoey	0	0	Logging	1	4171
							Å
U Li	aunch	Pause <u>H</u> alt	<u>F</u> lush	<u>D</u> isplay	<u>T</u> riggers	Enable Eve	nts

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 that contains the Events panel and the timeline associated with those events.

%		NightTrace	e - New Sessio	n(Unsaved)			-0
<u>File View Daemon</u>	ns Sea <u>r</u> ch S <u>u</u>	mmary <u>P</u> rofiles 1	Ti <u>m</u> elines <u>T</u> ools	<u>H</u> elp			
D 🛛 🏷	💐 = 🕼	🕲 🍘 🎾) 🎾 🎾 🤉	2 1 0 1		oo ≬≞	н
Trace app_data			anno Events an				
Offset	Event	Process	Thread Tag	J Time (see	:) Description		
0	NT_TIMER	0	0	0.000_000_00	0 arg1=0x0		
1	NT_TIMER	0	0	0.010_000_00	0 arg1=0x0		
			app data of				
			upp_unu				
L							
User Events:							
ober Evenes.							
		0.000001s					
		0.0001s	ρ.0	031s	0.0061s	1	p.0091s
Current Time	0.000 000	000 Hover time fr	om current timeli	ne = 0.000 000 0	00		
Start Time	0.000 000	000					
End Time	0.000 000	000					
Span	0.000 000	000 Current offse 00 from curr arg1=0x0	t=0_id=NT_TIMEI ent time)	R proc=0 thr=0 t	time(sec)=0.000	0_000_000 (0.000_000_0
•							

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 Up to zoom out
- Press Alt-Up to zoom out completely.

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

- Click in the middle of the lower panel.

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

			Night	Trace -	New S	Sessio	on(Un	isave	d)							Ŀ	
e <u>V</u> iew <u>D</u> ae	mons Sea <u>r</u> ch	S <u>u</u> mmar	y <u>P</u> rofile	es Ti <u>m</u>	elines	Tools	<u>H</u> elp										
) 🛛 🕻	= 🛱 🕯	F 1	s 🔊)	ø	9	Σ		© <mark>⊪</mark>	* ₩	101 abc	Œ	620		. н		
ace app_da	ita																
					eesse Eve	ents 🔗											E
Offset	Even	(P	rocess		Thread	Ta	g		Time	(sec)	Des	cript	ion				
3821	cycle_en	1	арр		sin	ו		66.70	6_33	1_287	arg1	=0.6	15661	L			
3822		3	арр		COS	5		66.75	6_34	3_175	arg1	=0.7	82608	3			
3823	cycle_star	t	app		sin	ו		66.75	6_34	4_480							
3824	cycle_en	1	app		sin	1		66.75	6_34	5_954	arg	1=0.6	52251	5			
3825		\$	app		COS	5		66.80	6 36	0 420	arg1	=0.7	77146	5			
Thread: cos(33	354)																
Thread: cos(33 Thread: sin(33 User Events:	354) 53)																
Thread: cos(33 Thread: sin(33 User Events:	354) 53)		0. 2 s	1 1	,			60	0.15						1	20.1s	
Thread: cos(33 Thread: sin(33 User Events:	354) 53)		0.1s	1 1	1			00 00).15				1 1	-1	ئے ا	20.1s 1 20.1s	
Thread: cos(33 Thread: sin(33 User Events: Current Time Start Time End Time	66.772 0.000 134.992	331 556 200 000 206 034	0.1s Hover ti	i ime fron	ı ı	ı ı	line = 4	48.452).15 .15 .12 .487	· · · · · · · · · · · · · · · · · · ·			11		, l	20.1s	
Thread: cos(33 Thread: sin(33 User Events: Current Time Start Time End Time Span	66.772 0.000 134.992 134.992	831 556 200 000 206 034 206 034	0.1s 	ime fron . offset= 602 from .62251:	n curren 3824 icom curre	ht time	line = d e_end ie)	60 60 48.452 proc=).15 .15 2_487	/ /_880 thr=si	n tin	 ne(sec	<u> </u>	.756_	1 1 345_9	20.1s 20.1s	.0
Thread: cos(33 Thread: sin(33 Jser Events: Jser Events: Current Time Start Time Start Time	66.772 0.000 134.992 134.992	831 556 000 000 006 034 006 034	0.15 1 Hover ti Current 16_485 arg1=0	ime fron offset= _602 fro .62251	n curren 3824 ico om curres	ht time d=cycle	line = - e_end ie)	60 60 48.452 proc=).15).15 2_487	1 7_880 thr=si	n tirr	 ne(sec	L L L L L L L L L L L L L L L L L L L	.756_:	1 1 1 345_9:	20.1s 20.1s 54 (0	.0

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.

Our application contains four threads: heap, sin, cos, and main. Rows for individual threads show only events logged by that thread. In addition, there is a user events graph near the bottom that shows events for all threads.

NOTE

You will see blank labels and graphs in your timeline. These are the labels and graphs for the main and heap threads which are not logging any events. The contents of the label are not shown until at least one event is logged by the main thread. If you see all blank labels, you likely didn't click in the middle of the timeline as instructed in the preceding step.

In "Using Tracepoints" on page 3-39 in the Using NightView section, we inserted tracepoints into the sine and cosine threads, which registered themselves as "sin" and "cos", respectively.

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 click anywhere within the timeline
- Press the Down key repeatedly until you can see individual lines in the graph
- Press the Up 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 Right 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 Left key causes the current timeline to go to the previous event.

- Press Ctrl+Right

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+Left 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

				ts ananan			
Offset	Event	Process	Thread	Tag	Time (sec)	Description	▲
941	2	app	sin		18.639_737_052	arg1=0.707107	
942	3	app	COS		18.689_830_607	arg1=0.713250	
943	1	app	sin		18.689_830_614		
944	2	app	sin		18.689_833_985	arg1=0.713250	
945	1	app	sin		18.739_897_536		
946	3	app	COS		18.739_897_556	arg1=0.707107	
947	2	арр	sin		18.739_900_519	arg1=0.719340	

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 Down key to advance to the next event.
- Press the Up 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 1		
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.

			sososososos <mark>Even</mark>	ts nonnon		
Offset	Event	Process	Thread	Tag	Time (sec)	Description
938	cycle_end	app	sin		18.589_677_378	arg1=0.700909
939	cycle_start	app	sin		18.639_734_687	
940	3	app	COS		18.639_736_625	arg1=0.719340
941	cycle_end	app	sin		18.639_737_052	arg1=0.707107
942	3	app	COS		18.689_830_607	arg1=0.713250
943	cycle_start	app	sin		18.689_830_614	
944	cycle_end	app	sin		18.689_833_985	arg1=0.713250
945	cycle_start	арр	sin		18.739_897_536	
946	3	app	COS		18.739_897_556	arg1=0.707107
947	cycle_end	app	sin		18.739 900 519	arg1=0.719340
948	cycle_start	app	sin		18.789_970_202	
949	3	app	COS		18.789_972_729	arg1=0.700909
950	cycle_end	арр	sin		18.789_972_991	arg1=0.725374

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 dialog appears containing the Profile	Status	List	and Profile	Definition	ſ
areas:					

1		Pr	ofiles			×
Type Name	Status	Count	Last	Offset		
]
Key / Value	Condition	▼ Res	set		Choose Profile	
Events (ALL					Browse
Exclude Events	NONE					Browse
Condition (TRUE					
Processes	ALL					Browse
Threads (ALL					Browse
Output Script (/usr/lib/NightTrace/bin	/event-summary	۸.sh			Browse
CPUs	all	mask=all)				
Name	cond					
Close dialog on sum	nmary or successful se	arch				
Add Apply 🎘 S	earch Bac <u>k</u> ward	Search Forward	Halt Sear	rch <u></u> Su	mmari <u>z</u> e Clos	e Help

Figure 4-9. Searching using the Profiles Dialog

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

🕤 Select E	vents (on	raptor)	×
NONE			
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:	🖊 Next	🕇 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 dialog.

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.

			oppoppoppop Even	ts receive			······································
Offset	Event	Process	Thread	Tag	Time (sec)	Description	
986	cycle_end	app	sin		19.390_874_186	arg1=0.793353	
987	3	app	COS		19.440_928_124	arg1=0.615661	
988	cycle_start	app	sin		19.440_929_571		
989	cycle_end	app	sin		19.440_931_735	arg1=0.798636	_
990	3	app	COS		19.490_987_163	arg1=0.608761	
991	cycle_start	app	sin		19.490_987_360		
992	cycle_end	app	sin		19.490_989_483	arg1=0.803857	
993	3	app	COS		19.541_048_819	arg1=0.601815	
994	cycle_start	app	sin		19.541_048_848		
995	cycle_end	app	sin		19.541_051_516	arg1=0.809017	
996	cycle start	app	sin		19.591 126 040		
997	3	app	COS		19.591_126_446	arg1=0.594823	
998	cycle_end	арр	sin		19.591_128_256	arg1=0.814116	<u> </u>

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.

- Move the mouse cursor to the event description box at the bottom of the panel and leave it there without moving it



Figure 4-12. Timeline Panel After Search

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.

On the **Trace** tab, 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.

Using States

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

- Click either of the **Profiles** icons on the toolbar

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

H

1			Profiles	5				X
Type Name	Sta	tus C	ount	Last O	ffset			٦
obtuse	Tri	Je	416	5672				
Key (Malua	Condition		Deset			Chasse Drafile		
Key / Value	Condition	▼	Reset	ļ		Choose Profile.	·	
Events	cycle_end						Browse	
Exclude Events (NONE						Browse	
Condition (arg_dbl > 0.8							
Processes	ALL						Browse	
Threads (ALL						Browse	
Output Script (/usr/lib/NightTrace	e/bin/event-su	immary.sh				Browse	
CPUs (all	(mask=all	I)					
Name (obtuse							
X Close dialog on sum	nmary or successfi iearch Bac <u>k</u> ward	ul search	Forward H	alt Search	h Σ Su	mmari <u>z</u> e	Close Help	

Figure 4-13. Profiles Dialog 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.
- Close the dialog.

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".

1	Edit State Graph Profile	(ox)
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)		
	Сал	cel Help	>

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.

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

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

- 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 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 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 table.

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.

A new	page is c	created of	display	ying th	ne results	of the	summary.
				(<u>(</u>)			

			ooooooo sine (0 to	38832) 0000000				ana Px
State Summa	ry Results	==						
Number of sta	ates found: 129	943						
Maximum sta Minimum stat Average state Total of state o	te duration: 0.0 e duration: 0.00 duration: 0.00 durations: 0.024	000_022_489 at offs 000_001_270 at offs 00_001_916 4_796_645	set: 26159 et: 27196					
Number of sta	ate gaps found: 1	2942						
Maximum sta Minimum stat Average state Total of state	te gap: 0.09 e gap: 0.04 gap: 0.050 gaps: 647.95	50_318_079 at offse 49_807_647 at offse 0_066_151 66_131_243	t: 30125 t: 30128					•
								•
Offset 🔻	End Offset	Duration (sec)	Gap (sec)	Event	CPU Process	Thread	Time (sec)	▼ Tag ▲
Offset 🔻	End Offset	Duration (sec)	Gap (sec)	Event cycle_start	CPU Process app	Thread sin	Time (sec)	Tag 📤
Offset	End Offset 5	Duration (sec) 0.000_003_417 0.000_003_168	Gap (sec) 0.000_000_000 0.050_094_051	Event cycle_start cycle_start	CPU Process app app	Thread sin sin	Time (sec) 3.018_492_735 3.068_590_204	Tag ▲
Offset	End Offset 5 8 11	Duration (sec) 0.000_003_417 0.000_003_168 0.000_002_693	Gap (sec) 0.000_000_000 0.050_094_051 0.050_081_071	Event cycle_start cycle_start cycle_start	CPU Process app app app	Thread sin sin sin	Time (sec) 3.018_492_735 3.068_590_204 3.118_674_444	Tag 🔺
Offset ▼ 3 6 10 13	End Offset 5 8 11 14	Duration (sec) 0.000_003_417 0.000_003_168 0.000_002_693 0.000_001_877	Gap (sec) 0.000_000_000 0.050_094_051 0.050_081_071 0.050_034_599	Event cycle_start cycle_start cycle_start cycle_start	CPU Process app app app app	Thread sin sin sin	Time (sec) 3.018_492_735 3.068_590_204 3.118_674_444 3.168_711_736	Tag 🔺
Offset ▼ 3 6 10 13 16	End Offset 5 8 11 14 17	Duration (sec) 0.000_003_417 0.000_003_168 0.000_002_693 0.000_001_877 0.000_002_337	Gap (sec) 0.000_000_000 0.050_094_051 0.050_081_071 0.050_034_599 0.050_085_178	Event cycle_start cycle_start cycle_start cycle_start cycle_start	CPU Process app app app app app	Thread sin sin sin sin	Time (sec) 3.018_492_735 3.068_590_204 3.118_674_444 3.168_711_736 3.218_798_791	Tag 🔺
Offset ▼ 3 6 10 13 16 19	End Offset 5 8 11 14 17 20	Duration (sec) 0.000_003_417 0.000_003_168 0.000_002_693 0.000_001_877 0.000_002_337 0.000_003_146	Gap (sec) 0.000_000_000 0.050_094_051 0.050_081_071 0.050_034_599 0.050_085_178 0.050_080_877	Event cycle_start cycle_start cycle_start cycle_start cycle_start cycle_start	CPU Process app app app app app app app	Thread sin sin sin sin sin	Time (sec) 3.018_492_735 3.068_590_204 3.118_674_444 3.168_711_736 3.218_798_791 3.268_882_005	Tag A
Offset ▼ 3 6 10 13 16 19 22	End Offset 5 8 11 14 17 20 23	Duration (sec) 0.000_003_417 0.000_003_168 0.000_002_693 0.000_001_877 0.000_002_337 0.000_003_146 0.000_002_545	Gap (sec) 0.000_000_000 0.050_094_051 0.050_081_071 0.050_034_599 0.050_085_178 0.050_080_877 0.050_062_966	Event cycle_start cycle_start cycle_start cycle_start cycle_start cycle_start cycle_start cycle_start cycle_start	CPU Process app app app app app app app	Thread sin sin sin sin sin sin sin	Time (sec) 3.018_492_735 3.068_590_204 3.118_674_444 3.168_711_736 3.218_798_791 3.268_882_005 3.318_948_118	Tag A
Offset ▼ 3 6 10 13 16 19 22 25	End Offset 5 8 11 14 17 20 23 26	Duration (sec) 0.000_003_417 0.000_003_168 0.000_002_693 0.000_001_877 0.000_002_337 0.000_003_146 0.000_002_545 0.000_002_579	Gap (sec) 0.000_000_000 0.050_094_051 0.050_081_071 0.050_085_178 0.050_080_877 0.050_062_966 0.050_055_387	Event cycle_start cycle_start cycle_start cycle_start cycle_start cycle_start cycle_start cycle_start	CPU Process app app app app app app app app	Thread sin sin sin sin sin sin sin	Time (sec) 3.018_492_735 3.068_590_204 3.118_674_444 3.168_711_736 3.218_798_791 3.268_882_005 3.318_948_118 3.369_006_049	Tag A
Offset ▼ 3 3 6 10 13 16 19 22 25 28	End Offset 5 8 11 14 17 20 23 26 29	Duration (sec) 0.000_003_417 0.000_003_168 0.000_002_693 0.000_001_877 0.000_002_337 0.000_003_146 0.000_002_545 0.000_002_579 0.000_002_003	Gap (sec) 0.000_000_000 0.050_094_051 0.050_081_071 0.050_085_178 0.050_080_877 0.050_062_966 0.050_055_387 0.050_056_031	Event cycle_start cycle_start cycle_start cycle_start cycle_start cycle_start cycle_start cycle_start cycle_start cycle_start	CPU Process app app app app app app app app app a	Thread sin sin sin sin sin sin sin sin	Time (sec) 3.018_492_735 3.068_590_204 3.118_674_444 3.168_711_736 3.218_798_791 3.268_882_005 3.318_948_118 3.369_006_049 3.419_064_658	Tag 📤
Offset ▼ 3 3 6 10 13 13 16 19 22 25 28 31	End Offset 5 8 11 14 17 20 23 26 29 32	Duration (sec) 0.000_003_417 0.000_003_168 0.000_002_693 0.000_001_877 0.000_002_337 0.000_002_146 0.000_002_545 0.000_002_579 0.000_002_003 0.000_002_837	Gap (sec) 0.000_000_000 0.050_094_051 0.050_081_071 0.050_085_178 0.050_080_877 0.050_062_966 0.050_055_387 0.050_056_031 0.050_061_915	Event cycle_start cycle_start	CPU Process app app app app app app app app app a	Thread sin sin sin sin sin sin sin sin sin	Time (sec) 3.018_492_735 3.068_590_204 3.118_674_444 3.168_711_736 3.218_798_791 3.268_882_005 3.318_948_118 3.369_006_049 3.419_064_658 3.469_128_576	Tag

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 clicking 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.

	soccossoccossocces sine Durations accossoccossoccos	
Current Time 439.513 282 82	8 State duration graph for state sine	
End Time 651.045 141 35	3 Statistics for state durations left of current time (439.463_226_567s):
Span 651.045 141 35	min = 0.000_001_316s @ 0; max = 0.000_003 active = false: last duration = 0.000_001_541s	3_303s @ 0; avg = 0.000_001_900s
This is a star and a second star as	1999 Il de la coltra contra la casa da la cada contra da cada da cada da cada contra cada contra da da da ser s Note	a na mana ang na mang kang kalaka kang kang kang kang kang kang kang k
IB-		1
0.15 100.15	200.1s 300.1s 400.1s	500.1s 600.1s
Hower time from surrent timeline1	02.070.570.014	
Hover time from current timeline = 1	22.970_570_914	
Current offset=26158 id=cycle_star	proc=app thr=sin time(sec)=439.513_282_828	(0.000_000_000 from current time)
		_

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+Up.

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.

We can rebuild the page using a different standard deviation index.

- Right-click the tab that contains the summary and click Delete Current Page.
- From the Summary menu, select Graphs State Durations and supply a value of 1 to the standard deviation request dialog.

second	
Current Time 482.698 909 949 Start Time 0.000 000 000 End Time 651.045 141 353 Span 651.045 141 353 Active = false; last_duration = 0.000_002_110s 0.000_002_110s	_ 852_902s): 0; avg = 0.000_001_894s
Bits 100.1s 200.1s 300.1s 400.1s 0.1s 100.1s 200.1s 300.1s 400.1s	500.1s 600.1s 500.1s 600.1s 500.1s 600.1s
Hover time from current timeline = $9.574_{193_{255}}$	
arg1=-0.017452	

Figure 4-19. State Durations Graph Modified

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 4-20. Timeline in Edit Mode

- 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.

app_data
Thread: cos(12515)
Thread: sin(12514)
User Events:
cycle = 0.000000. ms
0.1s 100.1s 200.1s 300.1s 400.1s 500.1s 600.1s
Current Time 482.672 852 902 Hover time from current timeline = 231.555_441_237 Start Time 0.000 000
Span 651.045 141 353 Current offset=28745 id=cycle_end proc=app thr=sin time(sec)=482.672_852_902 (0 .000_000_000 from current time) arg1=-0.017452 ▲

Figure 4-21. 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		
Kay (Malua	Condition Deset Choose Brefile		
Key / Value	Condition V Reset		•
Events	ALL	Browse	3
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 4-22. 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 4-23. Display Page with Data Graph

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 4-24. 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 -Intrace 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 16.878716 duration 0.000004 sine (end)offset 677 occur 339 code 2 pid 3399 time 16.928745 duration 0.000003 sine (end)offset 679 occur 340 code 2 pid 3399 time 16.978760 duration 0.000003 sine (end)offset 679 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
```

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 	Program:
 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:
	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 4-25. 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 staportion of each program, descriptions of those events for NightTrace, and various other files. An is be created for each program and will be called <i>programName.ai</i>. Program: app Define an illuminator for this program. Functions may be included or excluded from being traced by matching their names against recexpressions. The inclusions and exclusions in the list below are applied in order from top to both default, all functions are included except those beginning with underscore, those in C++ <i>std</i> m<i>main</i>, and Ada's internal I/O routines. Functions Included or Excluded from Being Traced: 	tically linked lluminator may v gular :om. By iamespace,
		Add Edit Delete Up Down
Creating Illuminator Done	As advanced features: (1) the Editor may be used to customize the user-defined illuminator, (2) may be used to customize additional illuminators, including the predefined ones, (3) to assist wit advanced customizations, the user-defined illuminator may be populated with all functions and of found in the program, and (4) a detailed report about the user-defined illuminator may be writte Console. Advanced, Build Prev Next	the Manager :h doing Jlobal variables n to the Help

Figure 4-26. 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 **pro-**gram 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	
 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 phtread ccur_rt
	customize the predefined ones. Glibc's debuginfo package(s) must be installed to customize glibe and pthread. Advanced

The Select Illuminators step is now displayed.

Figure 4-27. nlight Wizard - Select Illuminators Step

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

NOTE

The list of predefined illuminators may be different on your system. However, all systems should have main, glibc, and pthread.

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

	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 Relink Programs Activate Illuminators Run Scripts	Relink Illuminated Programs Illuminators have object files that must be linked with programs along with 1±bntrace. 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 By default, the copy of the program with the illuminators and 1±bntrace linked in is named originalNameAI. Illuminated Program Path: appAI The command to relink the program with illuminators may be specified using some substitution variables (itegyword) 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" Relink All 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

The Relink Programs step is now displayed.

Figure 4-28. 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
    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.

- 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	
 Select Programs Define Illuminators 	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
 Select Illuminators 	Detail Level controls how much detail is recorded as arguments to events.
 Relink Programs 	The gliba illuminator traces function calls to the system C library.
 Activate Illuminators 	🕱 glibc Detail Level: 2 🗸
 Run Scripts 	
	The <i>pthread</i> illuminator traces function calls to the POSIX threads library.
	🕱 pthread Detail Level: 2 💌
	This illuminator is the user-defined illuminator for the current program.
	🕱 app.ai Detail Level: 2 💌
	As an 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 Wizard).
	Advanced

The Activate Illuminators step is now displayed.

Figure 4-29. 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 app.ai illuminator to 3.
- Press the Next button to proceed to the next step.

Running the Program

The Run Scripts step is now displayed in the wizard.

The wizard provides this step for convenience.

We'll go ahead and close **nlight** now and run the application ourselves outside of **nlight**.

- Select Exit Immediately from the File menu.
- In a shell session, start the illuminated program: ./appAI &

IMPORTANT

Make sure you invoked **appAI**, the instrumented program, and not **app**.

Analyzing Application Illumination Events

Now we'll invoke NightTrace to analyze the data generated by our instrumented program.

 Enter the following command while positioned in the directory that contains the appAI program: ntrace --import=appAI The NightTrace analysis interface appears.

100				nti	race							
<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> rofiles	Ti <u>m</u> e	lines	<u>T</u> ools	<u>H</u> e	lp		
		- *	\$ =	1	1	P	۶	P	Σ		© ∥	»
				Import F ease specify OK	ile Name / trace file Car	name:	3					
Interva	ai:1e	events (0 to	0), 0.00	0400000 se	conas (0.0	00000	0000	to 0.00	0400	0000)	Curren	E III //

Figure 4-30. 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.

Use of the **--import** option instructs NightTrace to load auxiliary data created by **nlight** so that it can fully describe the trace events it collects. The location of that information is embedded within the instrumented application, in our case, **appAI**.

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.

8				Nig	ghtTra	ace - N	New S	essic	on(Ur	nsave	d)								
<u>F</u> ile <u>V</u> iew	<u>D</u> aemons	Sea <u>r</u> ch	S <u>u</u> mn	nary <u>P</u>	rofiles	Ti <u>m</u> e	elines	<u>T</u> ools	<u>H</u> elp	p									
) 🖗 🎽	\$ ≓	1				Þ	P	Σ		Ů II	*_	Ħ	Ħ	101 abc	œ			
			************				 Daer 	nons									.*.*.*.*.*.*.		ð×
Type Da	emon			Tar	get	L	.oggeo	ł		Lost		Sta	te	At	tache	ed	I	Buffer	
—U— /tn	np/data_impo	ort			narf							Halt	ed						
															F F	-			
<u>O</u> Lau	ncn	sume	II Paus	se	<u>H</u> ait		Flus		<u>D</u> ispia	ау	lrig	gers	•	ļ	nable	Even	ts	Delet	e
na ana ana ana ana ana ana ana ana ana						т	Trace Se	egmen	ts ····										ð×
	Trace Seg	ment		Target	L	oaaed	4	Los	t Du	ration	(sec)	Unsa	ved						
									_										
												Sa	ve Tra	ace Da	ata	0	lose Tr	ace Dai	ta
																			//

Figure 4-31. NightTrace - Daemon Ready to Launch

- Press the Launch button to launch the daemon.
- Press the **Resume** button to start collecting trace events.
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	NightTrace - New Session(Unsaved)									ox)						
<u>F</u> ile	<u>/</u> iew <u>D</u> aemons	Sea <u>r</u> ch	S <u>u</u> mn	mary <u>P</u> ro	ofiles	Ti <u>m</u> elin	nes <u>T</u> ool	s <u>H</u> elp	D							
	📮 🕴 🎘 🕻	\$ =	1	R	Þ	P	99	Σ		ڻ <mark>ا</mark>	u- 😐	H	101 abc	Œ		
Trace	/tmp/data_im	port														
							Daemons									Ð×
Туре	Daemon			Targ	et	Log	gged		Lost	5	State	Atta	ched	1	Buffer	
—U-	/tmp/data_imp	ort		n	arf		0		0	Lo	gging		3	3	11071	
U	Launch	esume	 <u>P</u> au	ise	<u>H</u> alt		<u>F</u> lush	<u>D</u> ispla	у	<u>T</u> rigge	rs	<u>E</u> n	able E	Events	<u>D</u> elet	e
						ංං Tra	ce Segme	nts 🕬								ð×
Тур	e 🔻 Trace Se	gment		Target	Lo	gged	Lo	st Du	ration	(sec) Ur	saved					
										[Save Tra	ace Data	3	Clos	e Trace Da	ta

Figure 4-32. NightTrace - Daemon Collection Events

- Wait until the event count in the Buffer column reaches 10,000 or more.
- 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.

- Click in the middle of the activity in the timeline.

	assassassas /tmp/data_import							
Current offset=208 id=ENTER_semop proc=appAI thr=cos time(sec)=3.627_639_266 (0.020_838_973 from current time) calling semop(semid=360459,sops=0x2aec6e7fbe50,nsops=1) *sops={ sem_num=0, sem_flg=0} caller=0x400de7 [cosine_thread() at app.c:64] frame=0x2aec6e7fbe60								
Hover time from current timeline = 0.630_173_038; 12 events around offset=412 id=ENTER_nanosleep proc=appAI thr=main tim e(sec)=4.278_651_277 calling nanosleep(req=0x7fff205ea3f0,rem=0x0) *req={ tv_sec=0, tv_nsec=50000000} *rem={ tv_sec=4646731, tv_nsec=12884901888} caller=0x400f40 [main() at app.c:94]								
Thread: heap_thread								
Thread: cos								
Thread: sin								
Thread: main								
All Application Illumination Events								
Current Time 3.648 478 239 Start Time 0.000 000 000 End Time 5.399 356 536 Span 5.399 356 536	$\begin{array}{c ccccccccccccccccccccccccccccccccccc$							
Image: A state of the state								

Figure 4-33. NightTrace - /tmp/data_import Timeline

AI timelines are much like standard user trace timeline, except that the event description and hover description areas are much bigger, because such descriptions are more detailed and verbose than most ordinary trace data.

In the figure above, notice the description of the semop and nanosleep library calls, including details about their arguments.

Let's turn our attention to the Events panel.

Offset	Event	Process	Thread	Tag	Time (sec)	Description
199	RETURN_malloc	appAl	main		3.627_632_309	returning from malloc()=0x6b6d00 errno=3
200	RETURN_add_link	appAl	main		3.627_632_428	returning from add_link()=7040256 errno=3
201	RETURN_work	appAl	main		3.627_632_539	returning from work() errno=3
202	ENTER_semop	appAl	main		3.627_632_668	calling semop(semid=360459,sops=0x7fff205ea4
203	RETURN_semop	appAl	main		3.627_636_031	returning from semop()=0 errno=3
204	ENTER_nanosleep	appAl	main		3.627_636_209	calling nanosleep(req=0x7fff205ea3f0,rem=0x0)
205	RETURN_semop	appAl	sin		3.627_637_407	returning from semop()=0 errno=3
206	ENTER_semop	appAl	sin		3.627_637_842	calling semop(semid=360459,sops=0x2aec6e5fa
207	RETURN_semop	appAl	COS		3.627_638_671	returning from semop()=0 errno=3
208	ENTER_semop	appAl	COS		3.627_639_266	colling semop(semid=360459,sops=0x2aec6
209	RETURN_nanosleep	appAl	main		3.677_685_671	turning from papacloon() 0 orrno 2
210	ENTER_random	appAl	main		3.677_68	ng semop(semid=500455,sops=0x2aecoe/ide50,nsops- ops=/
211	RETURN random	appAl	main		3.677_68	sem num=0,
212	ENTER_work	appAl	main		3.677_68	sem_op=65535,
213	ENTER_add_link	appAl	main		3.677_72	sem_flg=0}
214	ENTER_malloc	appAl	main		3.677_72 cal	ller=0x400de7 [cosine_thread() at app.c:64]
215	RETURN malloc	appAl	main		3.677_72	IIIE-0X2dec0e/iDe00
					2 6 7 7 7 2	

- Hover the mouse over the description area of the selected event.

Figure 4-34. NightTrace - Events Panel w/ Tool Tip

As mentioned before, trace event descriptions are quite long, and the description in the last column in the Events panel may be truncated. Hovering over those areas provides the full description.

- Activate the Textual Search dialog by pressing Ctrl+T while the focus is in the Events panel.

A textual search dialog is shown.

🔹 Event Panel Search 🛛 🗙							
This dialog allows you to search the Events Panel for text. Searching via this panel will not locate matching text in timeline panels, only in the Events Panel. Press the Profile Search button if you wish a more powerful search mechansim. Profile Search							
Event Panel Text Search							
Search Options							
□ Treat search text as a regular expression 🕱 Close dialog on successful search							
Case sensitive search Match any criteria 💌							
C Search Criteria							
CPU CPU							
Process							
Time Description							
Ready to search							
Search Backward Search Forward Halt Search Close Help							

Figure 4-35. NightTrace - Event Panel Search Dialog

- Activate the Event Name field by checking its checkbox.
- Type ENTER_work into the Event Name text field and press the Search Forward button.

				Events		
Offset	Event	Process	Thread	Tag	Time (sec)	Description 📤
203	RETURN_semop	appAl	main		3.627_636_031	returning from semop()=0 errno=3
204	ENTER_nanosleep	appAl	main		3.627_636_209	calling nanosleep(req=0x7fff205ea3f0,rem=0x0)
205	RETURN_semop	appAl	sin		3.627_637_407	returning from semop()=0 errno=3
206	ENTER_semop	appAl	sin		3.627_637_842	calling semop(semid=360459,sops=0x2aec6e5fa
207	RETURN_semop	appAl	COS		3.627_638_671	returning from semop()=0 errno=3
208	ENTER_semop	appAl	COS		3.627_639_266	calling semop(semid=360459,sops=0x2aec6e7fb
209	RETURN_nanosleep	appAl	main		3.677_685_671	returning from nanosleep()=0 errno=3
210	ENTER_random	appAl	main		3.677_685_873	calling random() caller=0x400f45 [main() at a
211	RETURN_random	appAl	main		3.677_686_031	returning from random()=31308902 errno=3
212	ENTER_work	appAl	main		3.677_686_167	calling work(control=902) caller=0x400f8
213	ENTER_add_link	appAl	main		3.677_720_874	calling add_link() caller=0x401378 [work() at
214	ENTER_malloc	appAl	main		3.677_721_040	calling malloc(bytes=16) caller=0x4013b3 [a
215	RETURN_malloc	appAl	main		3.677_721_317	returning from malloc()=0x6b6d20 errno=3
216	RETURN_add_link	appAl	main		3.677_721_431	returning from add_link()=7040288 errno=3
217	RETURN_work	appAl	main		3.677_721_545	returning from work() errno=3
218	ENTER_semop	appAl	main		3.677_721_731	calling semop(semid=360459,sops=0x7fff205ea4
219	RETURN_semop	appAl	main		3.677_725_165	returning from semop()=0 errno=3
220	CNITCO					

The Events panel now has the next occurrence of the ENTER_work event selected.

Figure 4-36. NightTrace - Events Panel after Search

Notice that the description field 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 (hover the mouse over the description to see it):

caller=0x400f83 [main() at app.c:95]

NOTE

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

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

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

Text Search	Ctrl+T
🖄 Sea <u>r</u> ch Forward	Ctrl+G
🖄 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	1
<u>D</u> isplay Fields	•

Figure 4-37. NightTrace - Events Panel Context Menu

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



Figure 4-38. NightTrace - Launches Editor with Source File at Line Number

NOTE

As mentioned above, the return PC is always in the next instruction after the call, which may mean it is associated with the next source line, as it is in the example above.

NOTE

NightTrace selects your editor via the EDITOR environment variable.

- Close the editor before proceeding.

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**.

- Select Summarize Functions from the Summary menu.

A panel with a summary of all instrumented functions that were called appears.

			Function Call Sum	mary (0 to 4812)				
# Completed	Total Time 🔻	Min Duration	Max Duration	Avg Duration	Min Offset	Max Offset	Active	Name
900	29.993_530_399	0.000_000_000	0.050_104_956	0.033_252_251	4812	1826	true	semop
3	14.996_382_769	1.948_880_211	5.000_057_676	3.749_095_692	4812	4185	true	sleep
300	14.988_897_527	0.000_003_917	0.050_060_280	0.049_797_002	4812	305	true	nanosleep
300	0.006_002_366	0.000_000_962	0.000_043_153	0.000_020_008	3169	1821	false	work
300	0.000_261_374	0.000_000_446	0.000_006_992	0.000_000_871	3216	2656	false	add_link
303	0.000_125_690	0.000_000_185	0.000_005_901	0.000_000_415	3519	583	false	malloc
300	0.000_075_874	0.000_000_121	0.000_006_569	0.000_000_253	3675	4171	false	random

Figure 4-39. NightTrace - Functions Summary Table

A table is created that presents a single row for each instrumented function. It contains statistics about the number of invocations, their minimum, maximum, and average length, and the name of the function.

The column labelled Active indicates whether a function call was ongoing at the end of the data set (or the end of the summarized interval).

The context menu provides the following actions:

Set current time to start of shortest call
Set current time to end of shortest call
Set current time to start of longest call
Set current time to end of longest call
Launch detailed summary of calls for this function
Save table as <u>t</u> ext
Export table as <u>c</u> omma separated list
Resize columns to contents

You can obtain details of a specific function by right-clicking its row in the table.

- Right click on the row for the work function and select Launch detailed summary of calls for this function.

		www. Call Details for	⁻ work (0 to 481	2) 1000000000		x
Duration 🔻	Start Time	End Time	Start Offset	End Offset	Thread	
0.000_043_153	8.684_260_243	8.684_303_395	1816	1821	main	
0.000_042_501	12.239_798_813	12.239_841_313	2956	2961	main	
0.000_042_338	7.733_230_454	7.733_272_792	1512	1517	main	
0.000_041_824	14.843_342_523	14.843_384_347	3788	3793	main	
0.000_039_343	11.288_374_448	11.288_413_791	2652	2657	main	
0.000_039_318	6.632_063_223	6.632_102_541	1160	1165	main	
0.000_039_049	15.043_614_293	15.043_653_342	3852	3857	main	
0.000_039_015	7.482_967_786	7.483_006_801	1432	1437	main	
0.000_038_895	9.034_830_203	9.034_869_098	1928	1933	main	
0.000_038_738	5.480_445_938	5.480_484_676	788	793	main	
0.000_038_273	15.294_039_323	15.294_077_597	3932	3937	main	
0.000_038_131	7.883_373_082	7.883_411_213	1560	1565	main	
0.000_037_968	13.541_495_314	13.541_533_281	3372	3377	main	
0.000_037_950	13.341_186_742	13.341_224_692	3308	3313	main	
•		* * * * * * * * * * * * * * * *				

A table appears with a row for every invocation of that function.

Figure 4-40. Function Details Table for the work function

This table has a context menu that is similar to the Function Summary table's context menu.

Batch Summary of Functions

You can also use **ntrace** in non-GUI mode to obtain summary information for all functions or 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 either of the following commands:

```
ntrace --verbose --summary=fs:* appAI /tmp/data
ntrace --verbose --summary=fs:work appAI /tmp/data
```

and it would generate output similar to the contents of the tables generated in the figures above, without presenting the graphical interface.

Shutting Down

- Select Exit Immediately from the File menu of NightTrace to terminate the NightTrace 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 LX Tutorial.

NightStar LX 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 1-2 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 RT takes advantage of significant performance capabilities of the RedHawk or SLERT kernel, eliminating intrusion on the process by sampling and modifying variables in other programs using direct memory fetches and stores. However, those capabilities are not present on standard Linux kernels, and so NightProbe LX must control the process using ptrace and start and stop the process to obtain memory samples and to modify memory. As a result, NightProbe LX cannot probe a process under the control of a debugger.

- If the app program is still under control of NightView, which was the case in a previous chapter, locate that NightView session, stop the process, and exit NightView.
- The run the applicagtion from a shell session: ./app

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.

•	nprobe Processes Target: r	s		
	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>File Target Programs View Record T</u> ools <u>H</u> elp	
📂 🌢 📮 🐽 🔅 🛸 🌆 🖹 🏥 🖽 🚇 🔵 🗉	1
<u>C</u> onfiguration <u>B</u> rowse	
_ Live Browser	
Filter Filter Clear Apply To Variables View All	•
ltem Value	
Ē- ♥ app pid=31229	
Refresh Auto Refresh	1.50s
	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.

Item	Value
🗄 🌞 app	pid=4640
⊕ f(x) add_link	"
🕨 head	0x0804b220
🕂 💶 data	
🕂 🖬 data[0]	
~ 🗸	
🔳 sema	294919
🕨 tail	0x0804e200
🔳 rate	5000000
🕂 🖷 🖬 ptrs	
state	run

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 Auto refresh checkbox.

- Click the Auto Refresh checkbox.

This causes the display to automatically refresh at the rate shown in the spinbox to the right of the Auto 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]	
\cdots 🕨 name	0x08048e4c
🔳 count	23098
- 🔳 delta	8.726646259971648E-03
🔤 🔳 angle	2.015680753127312E+02
🔤 🔳 value	4.848096201641618E-01
🖻 🗉 🖬 data[1]	
- 🕨 🕨 🕨	0x08048e50
- count	23098
🔳 delta	8.726646259971648E-03
🔤 🔳 angle	2.015680753127312E+02
🔤 💶 value	8.746197071849462E-01
V	
🗉 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 Tar <u>g</u> et <u>P</u> rograms <u>V</u> ie	ew <u>R</u> ecord <u>T</u> ools <u>H</u> elp		
6 🔌 🛛 🔉	o 📚 🌆 📄 🎫 🏥 🖽 🖓 🛑 💷 🖻		
Configuration Browse			
Filter .*	Filter Clear Apply To Variables 👻 View All 👻		
ltem	Value		
🗄 🖷 🌞 app	pid=31450		
🕂 🔳 data			
🖻 🖪 🖬 data[0]			
- 🕨 name	0x08048e4c		
🗖 count	26674		
🔳 delta	8.726646259971648E-03		
🖬 angle	2.327745623383434E+02		
🔤 value	2.923717045885066E-01		
🖨 🖬 🖬 data[1]			
- 🕨 🖿 🕨 name	0x08048e50		
Count	26593		
🔳 delta	8.726646259971648E-03		
🖬 angle	2.327745623383434E+02		
🔤 🖬 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.

X						Ni	ghtP	robe					
<u>F</u> ile	Tar <u>q</u> et	<u>P</u> rogram	s <u>V</u> i	iew	<u>R</u> ecor	d <u>T</u> ool	s <u>H</u>	elp					
E	> 🤌			۰									
<u>C</u> o	nfigurati	on <u>B</u> ro	wse	Tal	ole				 	 	 		
Мо	de: Vie	ew Live Sa	mples	5	•	Select	Items		 	 ****	 *****	eren eren (
Sa	mple #:	0 Auto	natic	Samr	olina 🗆	1 50						Clear	
	Sample	Auto	naue :	Janip	, ing (1.50	· · ·					Ciedi	
													/

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 View 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 Sampling checkbox, new samples are taken of the variables in the table.

Configuration Browse Table								
Mode:	View Live	Samples 🔻	Select Items					
San	nple 🔺	data[0].count	data[0].value	data[0].angle				
	22	14871	-8.241261886292722e-01	1.297739565320512e+02	50			
	23	14875	-8.433914458197415e-01	1.298088631170910e+02	50			
	24	15140	1.736481776761093e-01	1.321214243759801e+02	50			
	25	15160	3.420201433341833e-01	1.322959573011792e+02	50			
	26	15180	5.00000000076222e-01	1.324704902263784e+02	50			
	27	15200	6.427876096930828e-01	1.326450231515776e+02	50			
	28	15220	7.660444431243018e-01	1.328195560767768e+02	50,			
	29	15240	8.660254037884501e-01	1.329940890019759e+02	50			
	30	15260	9.396926207885635e-01	1.331686219271751e+02	50			
	31	15280	9.848077530135111e-01	1.333431548523743e+02	50			
•				1				
Sample Sam	#: 31 ple A	utomatic Sampling	1.50s		Clear			

Figure 5-10. Table in Automatic Sampling 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 Sampling 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	
<u>F</u> ile Tar <u>q</u> et <u>P</u> rograms <u>V</u> iew <u>R</u> ecord <u>T</u> ools <u>H</u> elp	
🌮 🤌 📮 💭 🗢 🎕 🌭 🌆 🚍 🏥 🖽 🔐 🛑 🗉	4
Configuration Browse Table Page 4	
Graph	
Mode: View Live Samples Select Items	
1000 E	
800 -	
% 600	
200 -	
0 200 400 600 800 Most Pecent Samples	1000
Sample #: 0 Automatic Scaling 🕱 🔺 🕨 💌 🔊 😒	amples
Sample Automatic Sampling 1.50s	Clear

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 Sampling 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).

M	Edit Curve 🔲 🗙
Varial	ble: data[0].value
Attr	ibutes
s	tyle Sticks 💌
Syn	nbol Ellipse 🔻
c	olor
	Scaling
	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

- Check the Automatic Scaling checkbox
- 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
🖹 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
i 🔳 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 Interval 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	
_ Process	
Program Path Select	
Program Arguments	
Output File //dev/null Select	
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 1-2.

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

cc -g -o api api.c -lnprobe

Description ltem 🇊 Target System narf Programs pid=18128 🛅 app O Views Playback Recording ldle Ė ① Timer 100 Milliseconds 🞴 Destinations 📖 🍎 api /home/jeffh/work/tutorial/api 🖮 🔳 Variables data[0].count int data[0].angle double double data[0].value data[1].count int data[1].angle double data[1].value double rate int

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

				g	nome-ter	minal _ C	ı x
<u>F</u> ile	<u>E</u> dit	<u>V</u> iew	<u>T</u> erminal	Ta <u>b</u> s	<u>H</u> elp		
item:	count:			290	72		
item:	angle:			253	.70106	3	
item:	value:			0.6	94658		
item:	count:			290	72		
item:	angle:			253	.70106	3	
item:	value:			-0.	719340		
item:	rate:			500	00000		
Sampl	e 479						
item:	count:			290	74		
item:	angle:			253	.71851	3	
item:	value:			0.6	81998		
item:	count:			290	74		
item:	angle:			253	.71851	3	
item:	value:			-0.	731354		
item:	rate:			500	00000		
Sampl	e 480						
item:	count:			290	76		
item:	angle:			253	.73596	7	
item:	value:			0.6	69131		
item:	count:			290	76		
item:	angle:			253	.73596	7	
item:	value:			-0.	743145		
item:	rate:			500	00000		
							-

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*.

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 LX Tutorial.

NightStar LX 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 1-2 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.

🔁 NightTune	
<u>File View Monitor Tools Help</u>	
🖻 📮 😥 🤌 📗 🚰 🇠 😇 🖛 📽 🍣 👂 🚳 🗢 🔮 Create Par	nels For: zoey 💌 » 🔀 🖘
conservation conservation conservation and conservation conservation conservation and conservation and conservation conservation and conservation	costocoto zeey CPU Shielding and Binding: costocoto 🗗 🗙
PID State Parent Size %CPU CPU Time CPU Affinity Nice RPri CL Command A	zoey: Intel(R) Xeon(TM) CPU 3.20GHz
Image: Second	
	0 100
6	Logond: User System Wait Idle n/a

Figure 6-1. NightTune initial panels

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.

**********					saasaa narf	Proces	s List: 🔗				- ×
PID	State	Parent	Size	%CPU	CPU Time	CPU	Affinity	Nice	RPri	CL	Command 🔺 📤
											🕂 🗊 haldaemon
											🗄 🕡 jeffh
											🛱 🗊 jojo 🔤
31422			32436	0.4	1.74						— 🍈 app
31430	Running	1	31896	5.7	49.35	1	all	0	0	OT	- 🧼 ntune
31333	Waiting	31257	9144	1.4	4.59	1	all	0	0	OT	🗀 🤲 🌞 sshd
31336	Waiting	31333	4464	0.0	0.24	1	all	0	0	OT	🖳 🌞 bash
											🖶 🗊 messagebus
											🗄 🗊 mysql 🔛
											🖶 🗊 ntp
											🗄 🕡 root
											🗄 🕡 rtkit
											🗄 🕡 🗊 statd
											🖶 🗊 syslog 💽
											🗄 🗑 tomcat6 🛛 🗨
× Find:	app								Next	1	Previous 🗌 Match case Found

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

					oppose narf	Proces	ss List: 🔗	an a			a x
PID	State	Parent	Size	%CPU	CPU Time	CPU	Affinity	Nice	RPri	CL	Command 🔺 📤
· · · · ·											🕂 🗊 haldaemon
											🗄 🗑 jeffh
											🖃 🗊 jojo
31422		1	32436	0.8	1.79						🖹 🧑 app
31422	Waiting			0.0	1.37	1	all	0	0	OT	🧼 🎡 main
31425	Waiting			0.4	0.21	0	all	0	0	OT	🎲 sin
31426	Waiting			0.4	0.21	0	all	0	0	OT	🎲 COS
31427	Waiting			0.0	0.00	1	all	0	0	OT	🦾 🎡 heap_thread
31430	Running	1	31896	5.2	51.45	0	all	0	0	OT	- 🌞 ntune
31333	Waiting	31257	9144	0.4	4.84	1	all	0	0	OT	🖻 🌼 sshd
31336	Waiting	31333	4464	0.0	0.24	1	all	0	0	OT	🦾 🌞 bash 📃
											🕀 🗊 messagebus
											🗄 🗊 mysql 🔄
											🗄 🖞 ntp 💽
											🗄 🛯 🚺 root 💽
× Find:	арр							- 1	Next	1	Previous 🗌 Match case Found

- 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 thread name, if it is being debugged by NightView, or, if the application is using the NightTrace API and names its threads via a call to trace_set_thread_name(3x).

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.

8	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

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

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;
      struct sembuf wait = \{0, -1, 0\};
40
41
      work(1);
42
43
      trace_set_thread_name (data->name);
44
45
      for (;;) {
46
         semop(sema, &wait, 1);
47
         data->count++;
48
         data->angle += data->delta;
         data->value = sin(data->angle);
49
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.

kB Total 12084 Reserved 0 Text 3 Library 11933 Data/Heap 136 Stack 12 Shared 5752 Non-Shared 6332 Resident 652 Locked 0 Unlocked 652 Non-resident 11432	Memory <u>U</u> sage	<u>M</u> emo	ny <u>E</u>	ile Descripto	rs	<u>S</u> ignals <u>C</u> a	ıpabilities	<u>E</u> nvironr	nent		
Total 12084 Reserved 0 Text 3 Library 11933 Data/Heap 136 Stack 12 Shared 5752 Non-Shared 6332 Resident 652 Locked 0 Unlocked 652 Non-resident 11432		kB				Usage					
Reserved 0 Text 3 Library 1193 Data/Heap 136 Stack 12 Shared 5752 Non-Shared 6332 Resident 652 Locked 0 0 11432	Total	12084				Shared					
Text 3 0 1208- Library 11933 Usage: Reserved Text Library Data/Heap Stack Shared Stack 12 Shared 5752 Shared 6332 Resident 652 On-Shared 652 Resident & Unlocked Non-resident Non-resident 11432 Itage: Itage:	Reserved	0				Residency					
Library 11933 Data/Heap 1136 Stack 12 Shared 5752 Non-Shared 6332 Locked 0 Unlocked 652 Non-resident 11432	Text	3				0					1208
Data/Heap 136 Stack 12 Shared 5752 Non-Shared 6332 Resident 652 Locked 0 Unlocked 652 Non-resident 11432	Library	11933				Usage:	Reserved	Text Li	brary Data/H	leap Sta	ack
Stack 12 Shared 5752 Non-Shared 6332 Resident 652 Locked 0 Unlocked 652 Non-resident 11432	Data/Heap	136				Sharodi	Shared	Non-charge			-
Shared 5752 Non-Shared 6332 Resident 652 Locked 0 Unlocked 652 Non-resident 11432	Stack	12				Shareu.	Shareu	Non-snared	<u> </u>		
Non-Shared 6332 Resident 652 Locked 0 Unlocked 652 Non-resident 11432	Shared	5752				Residency:	Resident	& Locked	Resident & Ur	locked	Non-resident
Resident 652 Locked 0 Unlocked 652 Non-resident 11432	Non-Shared	6332									
Locked 0 Unlocked 652 Non-resident 11432	Resident	652									
Unlocked 652 Non-resident 11432	Locked	0									
Non-resident 11432	Unlocked	652									
	Non-resident	11432									
					,						
						•			r -		
									·		

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

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.

Seven tabbed pages provide detailed information about the process, including:

- Memory Usage
- Memory Maps (not shown)
- 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> </u>	Descrip	tors	<u>S</u> ign	als	<u>(</u>	<u>a</u> p:	abilities	<u>E</u> nvir	onme	nt							
NUMA Node								1											
Local NUMA Nod	e							\downarrow											
Process bias is n	ot restricte	d to a si	ngle nod	e				+											
Locked/Resident																			
Exists																			
									_	_			_		_		_	_	_
)x00002aaaaa	_0000							+											
a0 a2	a4	a6	a8	aa	ı l	ac	ı	ae	ь0	ı b	2	b4		b6	b8	ba	, b	c I	be
~ ^^^ ^	0000			_					<u> </u>	<u> </u>	-	-							
0 1	2	3	4	5		6		7	8	9		a		h	6	d	-		f
	Ĺ L	Ĺ	ſ⊥	Ĺ				Ĺ	цĽ	Ĩ		<u> </u>		Ľ		Ľ	Ĺ		Ĺ
	00007.5.5		000					'						ſ	7 M				
current Page.	0000244	iaaaert	00					_							Zoom Max				
Status: Re	esident							F	Previous	Region	Pre	vious	Page	:	Zoom Out	Ne	xt Page	Ne:	xt Region
NUMA Node: 0								ſ	Shift N	/lin		Shift I	Left	ו	Zoom In	Sh	ift Riaht	s	hift Max
									5			211111			2001111				
															Zoom Min				
-Memory Regior	Informati	on——																	
File Mapping: /	SYSV010	50025,0	offset 0x7	000															
Addresses: (0x00002a	aaaaae(0000 - 0x	00002	aaaaa	aeea	afff		Act	ive: 0					Sh	ared:	0		
Permissions:	× Read	× Write	e 🗌 Ex	ecute					Inact	ive: 18	44674	4073	7095	516	515 Shared C	lean:	0		
Shared	Shared						F	Back	ed by Sw	an: 18	44674	4073	7095	516	515 Shared	Dirty:	0		
Size: A	1239360							Jack	cuby Sw	αp. 10		-1075	.055.	. 10	Dr	ivate:	40960		
Resident:	10960														Private C	lean	0		
NUMA Policy: 1	Default														Private	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 is not available in NightTune LX because the operating system support is not present in standard Linux.

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.

Me	mory <u>U</u> sage	<u>M</u> emory	<u>F</u> ile Descriptors	<u>S</u> ignals	<u>C</u> apabilities	<u>E</u> nvironment						
			Pathi	name/Desc	ription							
0	/dev/pts/0											
1	/dev/pts/0											
2	/dev/pts/0											
3	3 pipe:[121803] (pid 8344/ntune fd 4)											
4 pipe:[121803] (pid 8344/ntune fd 3)												
5	5 pipe:[121806] (pid 8344/ntune fd 6)											
6	pipe:[121806]	(pid 8344/nt	une fd 5)									
7	socket:[12180	7]: unix/strea	m: state=CONNECT	ED								
8	/usr/lib/NightTune/lib/ntune.msg											
9	socket:[121829]: tcp: local=raptor:42017 remote=raptor:25517 state=ESTABLISHED (pid 2320/nslm fd 5)											
10	/proc											
11	/proc/shield/irc	ąs		File or device associated with the file descriptor in one of these formats:								
12	/proc/shield/ltn	nrs	File or de filenam									
13	/proc/shield/pr	ocs	filenam	e (deleted)								
14	/proc/ccur/swit	ches	pipe:[in socket:	ode] (other- [inode]: tcp/i	pid) Jdp/raw: local=ip	:port remote=ip:p	ort state=s (otł	ner-pid)				
15	/proc/stat		socket:	[inode]: unix	: name=associat	ed-filename state	2=5					
16	/proc/meminfo		States ar	[inode]: pac e either TCF	ket Pstates like EST/	ABLISHED, LIST	EN, FIN_WAIT	1,				
17	/proc/vmstat		etc., or S	TREAM stat	es like LISTENIN	G, CONNECTED), etc.					
18	/proc/diskstats		pipe inoc	le, or conne	cted to the other	end of a socket.	sing the same					
19	/proc/interrupt	s										
20	/proc/net/dev											

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> sag	je <u>M</u> emory	<u>F</u> ile De	escriptors <u>S</u> i	ignals <u>C</u> a	pabilities	<u>E</u> nviro	nment	
Number 🔻	Name	Pending	Shared Pendi	ng Blocked	Ignored	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
~ •	CIONODIA	_	_	_	_	_		

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 the sin thread in the **app** process.

🐨 zoey: NightTune - Proces	s Scheduler									
6670 (app)										
	Current System Values									
Priority	Scheduling Class: Other									
Scheduling Class: Other	Nice Value: 0									
Nice Value: 0	Real-Time Priority: 0									
Real-Time Priority: 0	Time Quantun: 100 msecs									
Time Quantum: 100 🔻	CPU Affinity: all									
CPU Affinity 0 🕱 1 🕱 2 🕱 3 🕱										
Set All Clear All	\$									
OK Apply	Reset Close 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.

The Process List panel now reflects these changes to the thread.

about the second s											
PID	State	Parent	Size	%CPU	CPU Time	CPU	Affinity	Nice	RPri	CL	Command 🔺 📥
											🗄 🗊 haldaemon
											🗄 🧃 jeffh
											🖕 🧃 jojo
31422		1	32436	0.0	2.90						🖨 🍈 🍈 app
31422	Waiting			0.0	2.22	1	all	0	0	OT	r 💮 🎂 main
31425	Waiting			0.0	0.34	1	all	0	3	RR	R 🧼 🥸 sin
31426	Waiting			0.0	0.34	0	all	0	0	OT	Г — 🎡 соs
31427	Waiting			0.0	0.00	1	all	0	0	OT	Г 🌼 🎡 heap_thread
31333	Waiting	31257	9144	1.4	7.60	0	all	0	0	OT	Г 🗄 🌞 sshd 🥤
											🖶 🧊 messagebus
											🗄 🗊 mysql
											🕀 🗊 ntp
											🕀 🗊 root
											🕀 🗊 rtkit 🚬
											🕀 🗊 statd
											🖬 📶 svsloa 🗌 🔳
× Find:	app								🖊 Ne	xt	🕈 Previous 🗌 Match case 🛛 Wrapped

### 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.

sas zoey CPU Shiel	ding and Binding: 🕬 🗗 🗶								
zoey: Intel(R) X	eon(TM) CPU 3.20GHz								
È System È Chip 0									
📄 Core 0									
i⊕ CPU 0	[0% Usage]								
⊡ CPU 2	[0% Usage]								
🖮 Chip 3									
🖻 🛛 Core 0									
🗄 CPU 1	[2% Usage]								
⊡ CPU 3	[3% Usage]								
	2								
	~								

### 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. 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.

🖬 NightTune 🗔 🗊								
<u>F</u> ile <u>V</u> iew <u>M</u> onitor <u>T</u> ools <u>H</u> elp								
D 🕽 😣 🤌 🛛 🛃 🚳 😇 🗢 💲		<mark>≱</mark> -	<b>66</b> •	¢	Cre	eate Panels For: zoey 🔹 » 🗙 🔹		
consistence zoey CPU Shielding and Binding: consistence 🗗 🗙				zoey Int	errupt A	Activity (Interrupts/Second): CONSISTENT CPUS		
zoey: Intel(R) Xeon(TM) CPU 3.20GHz		0	1	2	3	Description		
⊖. Chip 0	0	0	0	0	0	timer		
i⊟ Core 0 i⇒ CPU 0 [7% Usage]	1	0	0	0	0	i8042		
⊡ Bound Processes: 1 Threads: 1	3	0	0	0	0			
	4	0	0	0	0			
Bound Processes: 0 Threads: 0	7	0	0	0	0	parport0		
i⇒ Core 0	8	0	0	0	0	rtc0		
Bound Processes: 0 Threads: 0	9	0	0	0	0	acpi		
□- CPU 3 [18% Usage]	12	0	0	0	0	i8042		
Bound Processes: 0 Threads: 0	14	0	0	0	0	ata_piix		
	15	0	0	0	17	ata_piix		
zoey CPU Usage:	16	0	0	0	61	uhci_hcd:usb2, uhci_hcd:usb5, nvidia		
	17	0	0	0	0	Intel ICH5		
	18	0	0	0	0	ata_piix, uhci_hcd:usb4		
	19	0	0	0	0	uhci_hcd:usb3		
	23	0	0	0	0	ehci_hcd:usb1		
0 100	48	0	0	0	3546	eth0		
Logond: User System Wait Ide n/a	NMI	0	0	0	0	Non-maskable interrupts		
Legend. User System ware rule ma	LOC	168	99	61	161	Local timer interrupts		
	RES	540	421	126	76	Rescheduling interrupts		
	CAL	0	0	0	0	Function call interrupts		
	TLB	0	0	0	0	TLB shootdowns		
	SPU	0	0	0	0	Spurious interrupts		
	ERR	0						
	MIS	0						

### 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:

```
For SUSE:
    /sbin/service irq_balancer stop 2>/dev/null
For other Linux distributions:
    /sbin/service irqbalance stop 2> /dev/null
```

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.

🔏 zoey: NightTune - Interrupt Af	finity 🗆 🗙
15 IO-APIC-edge       ata_piix       ▼         Interrupt Affinity	Current Affinity all
OK Apply Reset Can	cel 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.

# **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.

X

- Terminate NightTune by selecting Exit from the File menu.

This concludes the NightTune portion of the NightStar LX Tutorial.

NightStar LX Tutorial

# A Tutorial Files

The following sections show the source listings for the files used in the *NightStar LX Tutorial*.

- api.c
- app.c
- function.c
- report.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;
        count;
  int
  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);
  trace_set_thread_name("heap_thread");
   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;
   }
}
struct node t {
  int value;
  struct node t * link;
};
struct node t * head;
struct node_t * tail;
int add_link (void)
{
  static int count;
  count++;
  if (count > 5 && count < 1000) {
      struct node_t * n = (struct
node t*)malloc(sizeof(struct node_t));
     n->value = count;
     n->link = NULL;
      if (tail) {
          tail->link = n;
      } else {
         head = n_i
      }
         tail = n;
   }
}
#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);
}
```