RedHawk NightStar Tools Tutorial

concurrent

0898009-040
March 2005

Copyright 2005 by Concurrent Computer Corporation. All rights reserved. This publication or any part thereof is intended for use with Concurrent Computer Corporation products by Concurrent Computer Corporation personnel, customers, and end–users. It may not be reproduced in any form without the written permission of the publisher.

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

To report an error or comment on a specific portion of the manual, photocopy the page in question and mark the correction or comment on the copy. Mail the copy (and any additional comments) to Concurrent Computer Corporation, 2881 Gateway Drive, Pompano Beach, FL 33069-4324. Mark the envelope **"Attention: Publications Department**."

This publication may not be reproduced for any other reason in any form without written permission of the publisher.

RedHawk, NightProbe, NightSim, NightTrace, NightTune, and NightView are trademarks of Concurrent Computer Corporation.

Linux is a registered trademark of Linus Torvalds.

Printed in U. S. A.

Preface

General Information

The RedHawk™ NightStar™ Tools allow users on an iHawk™ system running RedHawk Linux® to schedule, monitor, debug and analyze the run-time behavior of their time-critical applications as well as the RedHawk Linux operating system kernel.

The RedHawk NightStar Tools consist of the NightTrace™ event analyzer; the NightSim™ frequency-based scheduler; the NightProbe™ data monitoring tool; the NightView™ symbolic debugger; the NightTune™ system and application tuner; **shmdefine**, a shared memory configuration aid tool; and the Data Monitoring API.

Scope of Manual

This manual is a tutorial for the RedHawk NightStar Tools.

Structure of Manual

This manual consists of one chapter which is the tutorial for the RedHawk NightStar Tools.

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:

0890395	*NightView User's Guide*
0890398	*NightTrace User's Guide*
0898515	*NightTune User's Guide*
0890458	*NightSim User's Guide*
0890465	*NightProbe User's Guide*

Contents

Chapter 1 Using the RedHawk NightStar Tools

Overview 1-1
Before you begin. 1-1
Getting Started 1-3
Building the program 1-3
Using NightSim 1-5
Invoking NightSim 1-5
Configuring the Scheduler 1-5
Scheduling a process 1-7
Setting up the scheduler 1-9
Using NightView 1-10
Setting a monitorpoint 1-11
Resuming execution 1-13
Starting the simulation 1-13
Monitoring the simulation 1-14
Using NightProbe 1-16
Invoking NightProbe 1-16
Configuring NightProbe 1-17
Connecting to the target program 1-19
Starting sampling 1-20
Modifying program data. 1-21
Using NightTrace. 1-23
Invoking NightTrace. 1-23
Configuring a user daemon 1-24
Creating a customized display page 1-26
Creating the user application daemon 1-27
Resuming execution of the user application daemon 1-28
Displaying the user trace data. 1-28
Inserting a patchpoint 1-29
Viewing streaming trace output 1-31
Configuring a kernel daemon 1-32
Creating the kernel daemon 1-33
Resuming execution of the kernel daemon 1-35
Displaying the kernel trace data 1-35
Flushing the trace data 1-36
Stopping the daemons. 1-36
Positioning the current time line 1-37
Loading an eventmap file 1-39
Searching for a user trace event 1-39
Zooming in 1-41
Examining the kernel trace data 1-42
Using NightTune 1-45
Invoking NightTune 1-45
Binding an Interrupt to a CPU 1-47
Shielding a CPU 1-49

Examining the kernel trace data after tuning 1-51
Exiting the Tools 1-52
Exiting NightTune 1-53
Exiting NightTrace 1-53
Exiting NightProbe 1-53
Exiting NightSim 1-53
Exiting NightView 1-54
Conclusion 1-54

Appendix A Tutorial Files

sim.c A-2
rcim_timer.c A-4
rcim_timer.h A-6

Illustrations

Figure 1-1. NightSim Scheduler 1-5
Figure 1-2. NightSim Edit Process 1-7
Figure 1-3. Process Scheduling Area 1-9
Figure 1-4. NightView Dialogue 1-10
Figure 1-5. NightView Principal Debug Window 1-11
Figure 1-6. Setting a new monitorpoint 1-12
Figure 1-7. NightView Monitor Window 1-12
Figure 1-8. Resuming execution 1-13
Figure 1-9. Starting the simulation 1-14
Figure 1-10. NightSim Monitor 1-15
Figure 1-11. NightProbe Main window 1-17
Figure 1-12. Configured NightProbe Main window 1-18
Figure 1-13. NightProbe Spreadsheet Viewer window 1-19
Figure 1-14. User Authentication dialog 1-20
Figure 1-15. Modified values in NightView Monitor Window 1-21
Figure 1-16. Modified values in NightProbe Spreadsheet Viewer 1-22
Figure 1-17. NightTrace Main Window 1-24
Figure 1-18. Daemon Definition dialog 1-25
Figure 1-19. Login dialog 1-25
Figure 1-20. Import Daemon Definition dialog 1-26
Figure 1-21. Customized NightTrace display page 1-27
Figure 1-22. User trace data in customized NightTrace display page 1-29
Figure 1-23. Setting a new patchpoint 1-30
Figure 1-24. User trace data after patchpoint inserted 1-32
Figure 1-25. Daemon Definition dialog 1-33
Figure 1-26. NightTrace kernel display page 1-34
Figure 1-27. NightTrace Main window showing halted daemons 1-37
Figure 1-28. NightTrace kernel trace data 1-38
Figure 1-29. Search NightTrace Events dialog 1-40
Figure 1-30. User trace data after search 1-41
Figure 1-31. Zoomed in view of user trace data 1-42
Figure 1-32. Zoomed in view of kernel display page 1-43
Figure 1-33. Example NightTune Window 1-46
Figure 1-34. NightTune CPU Shielding Page 1-47
Figure 1-35. RCIM Interrupt Bound to CPU 0 1-48
Figure 1-36. Kernel Display Page with shielded CPU 1-52

Figure 1-37. Removing the scheduler 1-53
Figure 1-38. Remove Scheduler dialog 1-54

1 Using the RedHawk NightStar Tools

The RedHawk NightStar Tools allow users on an iHawk system running RedHawk Linux to schedule, monitor, debug and analyze the run time behavior of their real-time applications as well as the RedHawk Linux operating system kernel.

The RedHawk NightStar Tools consist of the NightTrace event analyzer; the NightSim frequency-based scheduler; the NightProbe data monitoring tool; the NightView symbolic debugger; the NightTune system and application tuner; **shmdefine**, a shared memory configuration aid tool; and the Data Monitoring API.

Overview

This is a demonstration of the RedHawk NightStar Tools. In this tutorial, we will use the following RedHawk NightStar Tools:

- NightSim
- NightProbe
- NightView
- NightTrace
- NightTune

integrating them together into one cohesive example.

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

Before you begin.

Some of the activities in the RedHawk NightStar Tools Tutorial require either root access or user registration in the `fbscheduser` capabilities role. Either execute the commands shown in the tutorial as the `root` user, or have your system administrator register you as an FBS user according to the following instructions:

1. Add the following line to the **/etc/pam.d/rsh** and **/etc/pam.d/login** files:

```
session required /lib/security/pam_capability.so
```

NOTE

For those users that log into their system directly from the Gnome or KDE graphical desktop environment, it is necessary to add the above line to **/etc/pam.d/gdm** or **/etc/pam.d/kde**, respectively. In addition, you must restart your X server or reboot your system before these changes will take effect.

2. Add the following line to the bottom of the **/etc/security/capability.conf** file:

   user *username* **fbscheduser**

   where *username* is the login name of the desired user.

3. Log off and log back onto the RedHawk system for these changes to take effect.

4. You may verify your capabilities by issuing the following command:

   /usr/sbin/getpcaps $$

   which lists the current capabilities.

   This list should include the following capability:

   ```
   cap_sys_nice
   ```

   which is necessary for the proper execution of this tutorial.

Getting Started

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

To create a working directory

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

  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 the RedHawk NightStar Tools. We will copy these tutorial-related files to our **tutorial** directory.

To copy the tutorial-related files to the working directory

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

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

Building the program

Our example uses a cyclic program which intends to do some work every time an external event triggers it.

We will use RedHawk Linux's Frequency Based Scheduler to control the execution of the program. The Frequency Based Scheduler allows us to field an external interrupt and control the execution of one or more programs.

A portion of one of the source files, **sim.c**, is shown below:

```
main()
{
   int arg;

   counters.SetWorkload(0);

   trace_setup ("sim-data") ;

   while (fbswait() == 0) {
      timer.start();
      counters.Increment(1);
      trace_event_arg (cycle_start, counters.Get());
      counters.Work();
      timer.stop();
      arg = counters.Get() % 10;
      trace_event_arg (cycle_end, arg);
      counters.cycle_time = (float) timer.elapsed();
   }
}
```

The program calls `fbswait()` which will cause it to block until the frequency-based scheduler determines that it is time for this program to execute.

At that time, the program enters the loop where it increments some counters, logs a trace point with the NightTrace API `trace_event_arg()`, calls the procedure `counters.Work()`, logs another trace point to signal the end of the calculations done by `counters.Work()`, then returns to `fbswait()` to await the next cycle.

You only need to make a single FBS API call, `fbswait()`, to have a program which can be scheduled on the FBS.

Now that we have the source files, we need to build the program. We will use the **g++** compiler.

To build the executable

- From the local **tutorial** directory, enter the following commands:

```
g++ -c -g *.c
g++ -o sim *.o -lntrace -lccur_fbsched -lccur_rt
```

NOTE

The RedHawk NightStar 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, when compiling with releases prior to **gcc** 3.2, it is necessary to use the **-gdwarf-2** option in place of the **-g** option.

Using NightSim

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

Invoking NightSim

To invoke the NightSim Scheduler

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

  `nsim &`

Configuring the Scheduler

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

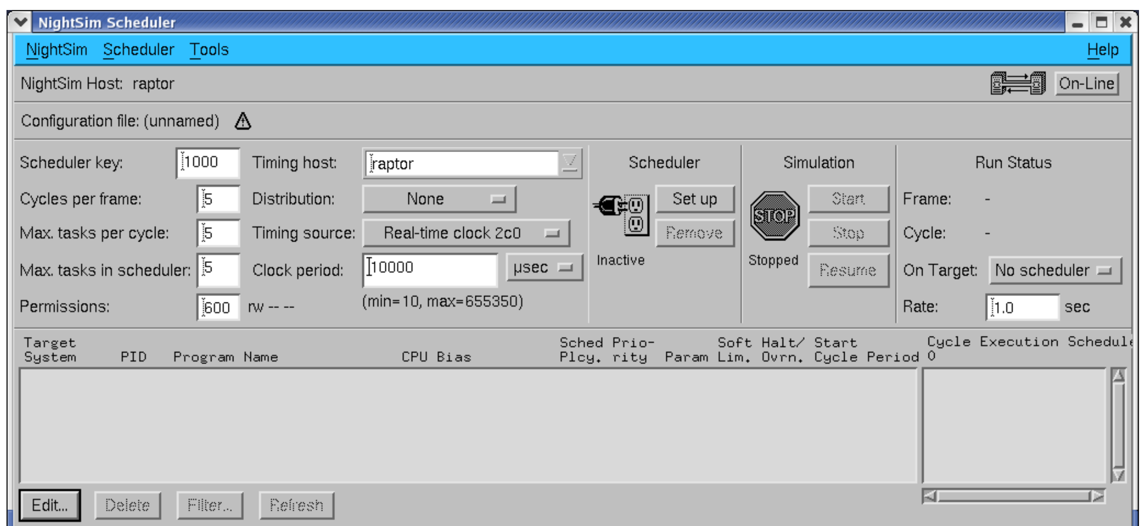


Figure 1-1. NightSim Scheduler

The FBS schedules processes in a cyclic manner based on some (usually cyclic) interrupt source.

We use the term *cycle*, or *minor cycle*, to represent the smallest amount of time between occurrences of the interrupt.

We use the term *frame*, or *major frame*, as simply a convenience to represent a set of one or more cycles. Often, the most simple schedulers have 1 cycle per frame. More complex applications may have different sets of activities that need to be accomplished before the entire application repeats; such applications would define multiple cycles per frame.

To configure a NightSim Scheduler

- Specify a Scheduler key. The key is a user-chosen numeric identifier with which the scheduler will be associated. For our example, we will use 1000.
- Specify the Cycles per frame. This field allows you to specify the number of cycles that compose a frame on the specified scheduler. We will use the value 5.
- Specify the Max. tasks per cycle. This field allows you to specify the maximum number of processes that can be scheduled to execute during one cycle. Enter 5 for our example.
- Specify the Max. tasks in scheduler. This field allows you to specify the maximum number of processes that can be scheduled on the specified scheduler at one time. For our example, we will specify the value 5.
- For the Timing host, enter the name of the RedHawk Linux system on which NightSim is running. For our example, we will enter raptor in this field.

NOTE

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

- Select a Timing source from the list provided. This list contains the set of devices available on the timing host. We will use Real-time clock 2c0.
- Specify Clock period.

  For our simulation, we would like the real-time clock to “fire” every .01 seconds (or 10000 microseconds).

  For our example, we will specify 10000 for the number of microseconds.

Scheduling a process

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

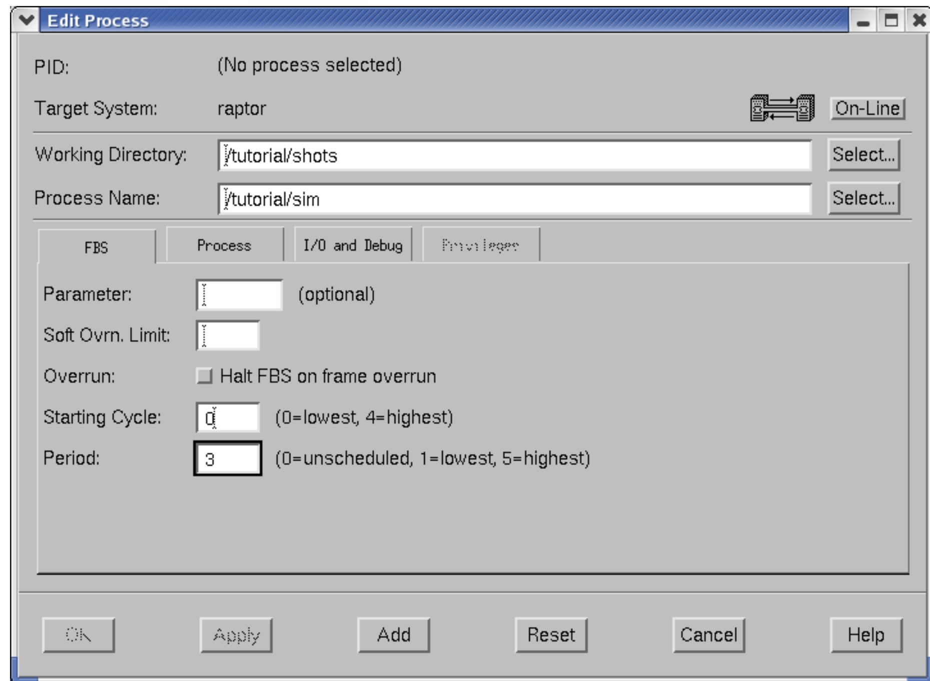


Figure 1-2. NightSim Edit Process

To add a process to the frequency-based scheduler

- Press the Edit... button on the NightSim Scheduler window. This will bring up the Edit Process window.
- Press the Select... button next to the Process Name field. This brings up the Select a Program dialog.
  - Choose the program we wish to schedule from the Files list. For our example, we will select **sim** from the list.
  - Press Select to select the program.
- Ensure that the Working Directory is the same directory that contains our program (the directory of the Process Name selected in the previous step).

- Click on the FBS tab:
  - Select Starting Cycle.

    This field allows you to specify the first minor cycle in which the specified program is to be wakened in each major frame.

    We will choose the lowest value, 0, for our example.
  - Select Period.

    This field allows you to establish the frequency with which the specified program is to be awakened in each major frame. Enter the number of minor cycles representing the frequency with which you wish the program to be awakened.

    For our example, we will specify a period of 3, indicating that the specified program is to be awakened every third minor cycle.
- Click on the Process tab:
  - Click on the All CPUs checkbox to deselect all of the CPUs
  - Choose a single CPU for this process to run on.

    For our example, we will specify CPU 0 by clicking on the checkbox labeled 0.
  - Ensure that the FIFO scheduling policy is selected.
  - Specify the Priority for this process.

    The range of priority values that you can enter is governed by the scheduling policy specified. NightSim displays the range of priority values that you can enter next to the Priority field. Higher numerical values correspond to more favorable scheduling priorities.

    For our example, we will give the process a priority of 50.
- Click on the I/O and Debug tab:
  - Check the Schedule program within a NightView dialogue checkbox. This will bring the program up in the NightView debugger before the program executes.
- Press Add to add the process to the frequency-based scheduler.

We would also like to measure the idle time on the same CPU. We can do this by scheduling the **`/idle`** process.

To schedule the /idle process

- In the Edit Process window, enter:

  `/idle`

  in the Process Name field.

- Press the Add button to add the **`/idle`** process.
- Press the Close button to dismiss the Edit Process window.

You will notice that two entries now appear in the Process Scheduling Area of the NightSim Scheduler window as shown below.

Target System	PID	Program Name	CPU Bias	Sched Plcy.	Prio-rity	Param	Soft Lim.	Halt/ Ovrn.	Start Cycle	Period	Cycle Execution Schedule 0
raptor	----	/tutorial/sim	0..............	F	50		0	No	0	3	X . . X .
raptor	----	/idle	0..............	F	0		0	No	0	1	X X X X X

Figure 1-3. Process Scheduling Area

Setting up the scheduler

To set up the scheduler

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

  This action:

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

Because we have specified the Schedule program within a NightView dialogue option when we added this process to the frequency-based scheduler (see "To add a process to the frequency-based scheduler" on page 1-7), the NightView Source Level Debugger will be started.

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. In addition to standard debugging capabilities, NightView supports application-speed eventpoint conditions, hot patches, synchronized data monitoring, exception handling and loadable modules.

Because we have specified the Schedule program within a NightView dialogue option when we added this process to the frequency-based scheduler (see "To add a process to the frequency-based scheduler" on page 1-7), NightView is started when the scheduler is set up (see "Setting up the scheduler" on page 1-9). A NightView Dialogue window is presented as well as a Principal Debug Window with the execution of the program stopped.

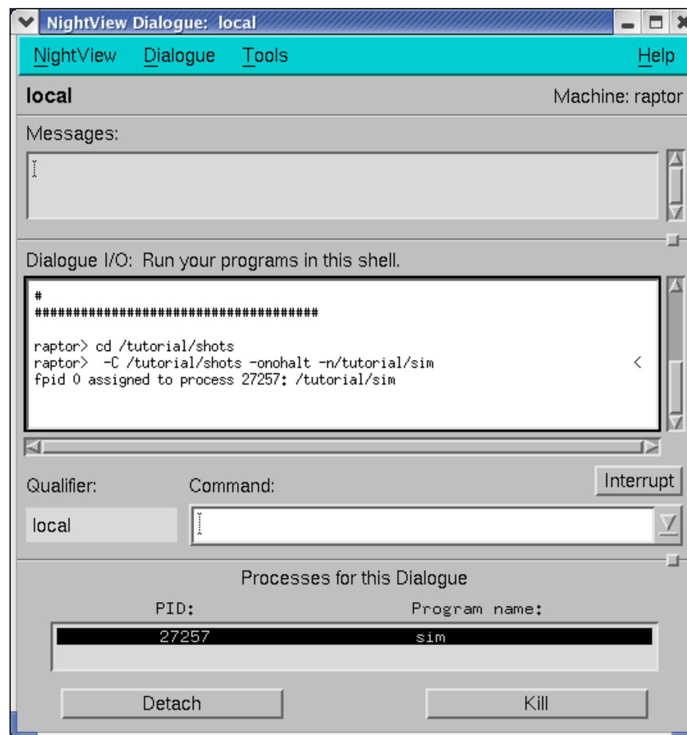


Figure 1-4. NightView Dialogue

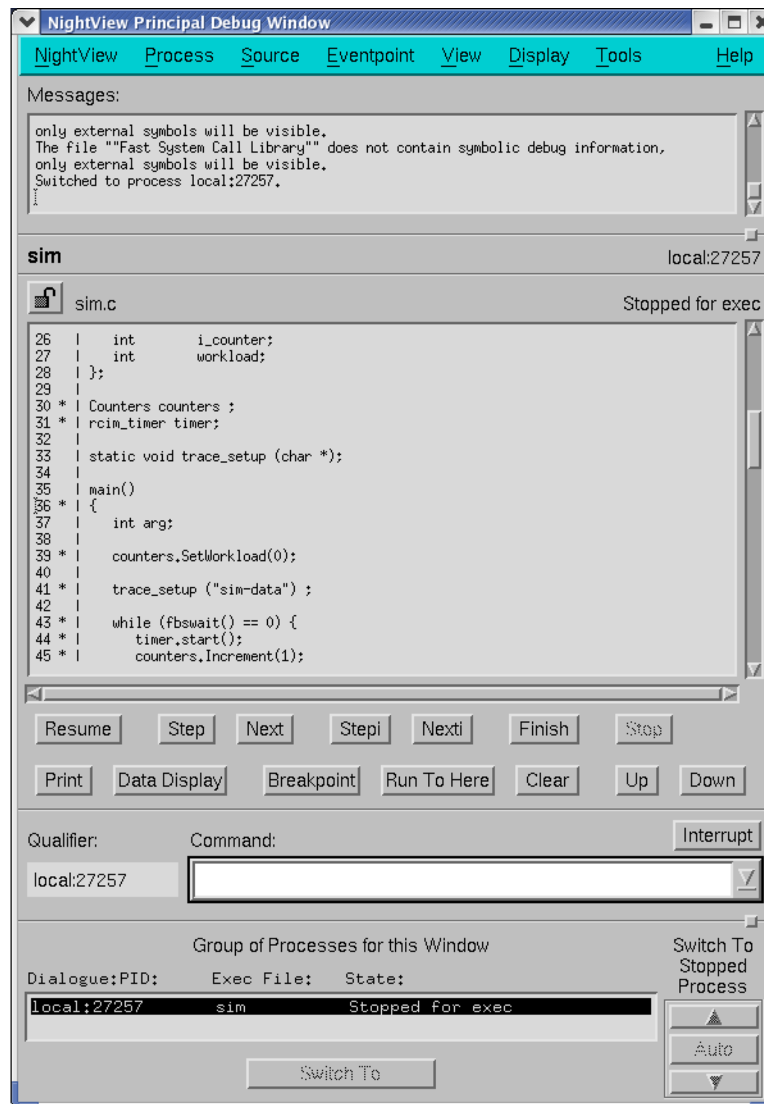


Figure 1-5. NightView Principal Debug Window

Setting a monitorpoint

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 Window.

To set a monitorpoint

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

```
while (fbswait() == 0) {
```

- Select Set Monitorpoint... from the Eventpoint menu. This will open the Set a New Monitorpoint dialog.

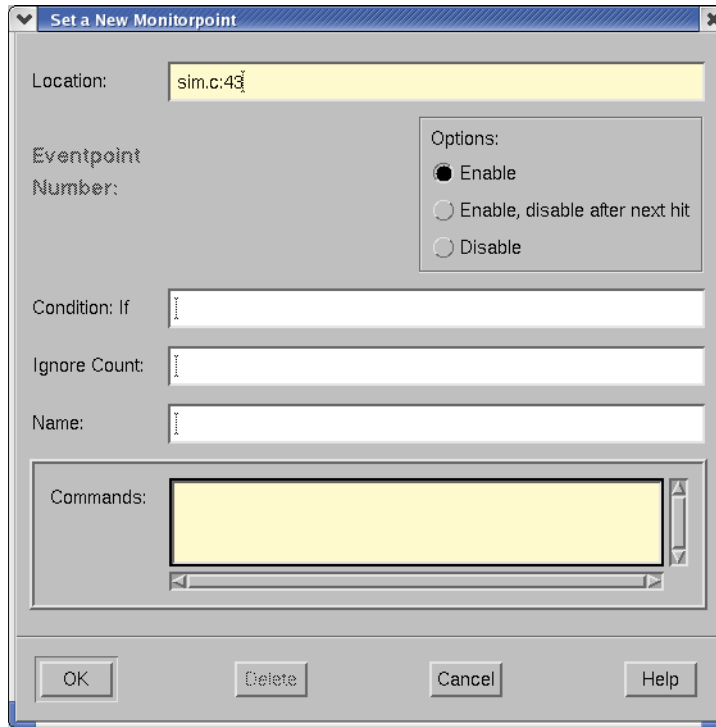


Figure 1-6. Setting a new monitorpoint

- Enter the following

  print counters.cycle_time

  in the Commands text box:.

- Press OK.

  A NightView Monitor Window is opened containing an entry for the `counters.cycle_time` variable.

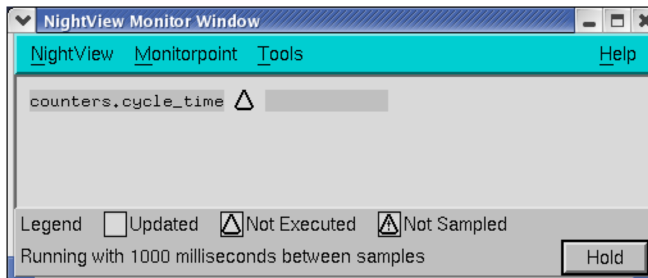


Figure 1-7. NightView Monitor Window

NOTE

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

```
monitorpoint at line_number
print counters.cycle_time
end monitor
```

where *line_number* coincides with the line:

```
while (fbswait() == 0) {
```

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

Resuming execution

Now it's time to let the program run.

To resume execution in NightView

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

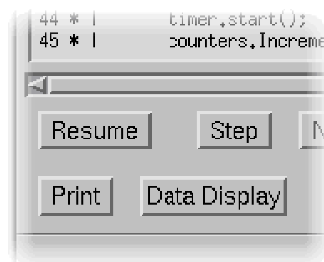


Figure 1-8. Resuming execution

Starting the simulation

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

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

To start a simulation in NightSim

- Press the Start button on the NightSim Scheduler window.

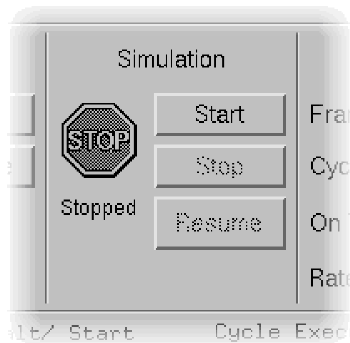


Figure 1-9. Starting the simulation

When the simulation begins, you should notice the values for Frame and Cycle in the Run Status Area begin to change.

Monitoring the simulation

The performance monitor is a mechanism that enables you to monitor FBS–scheduled processes' utilization of a CPU.

The performance monitor provides you with the ability to:

- Obtain performance monitor values by process or processor
- Start and stop performance monitoring by process
- Clear performance monitor values by processor

To create a performance monitor window

- Select Create Monitor Window from the NightSim menu on the NightSim Scheduler window.

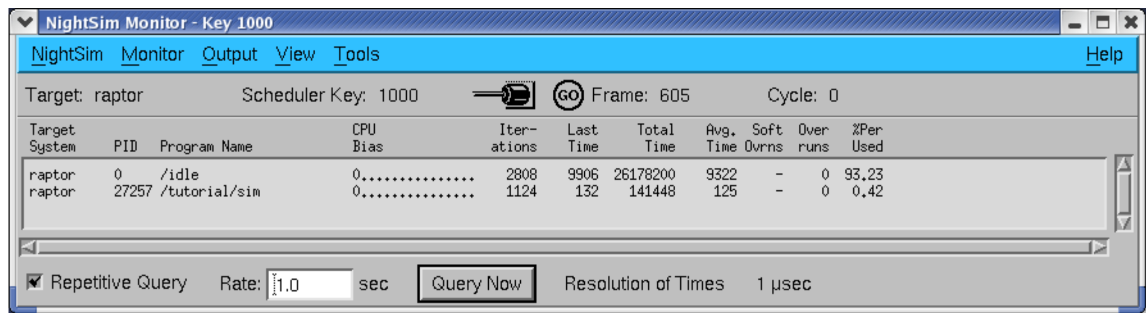


Figure 1-10. NightSim Monitor

Notice the value under the Last Time column for the process **`sim`**. This value shows the amount of time (in microseconds) that the process has spent running between the last time that it was wakened by the scheduler and the next time it called `fbswait()`.

Using NightProbe

NightProbe is a graphical tool for recording, viewing, and modifying data within a variety of resources:

- executing programs
- shared memory segments
- memory-mapped files and devices
- PCI devices

Data is sampled using non-intrusive techniques to guarantee short response time and minimal impact on the target resources and the target system. The source code of target programs does not need to be modified or recompiled in order to be monitored. Executing programs can be monitored and recorded without being stopped and restarted.

For more information on NightProbe, refer to the *NightProbe User's Guide* (0890465).

Invoking NightProbe

To invoke NightProbe Data Monitoring Tool

- From the Tools menu of the NightSim Scheduler window, select NightProbe Monitor.

The NightProbe Main window is opened.

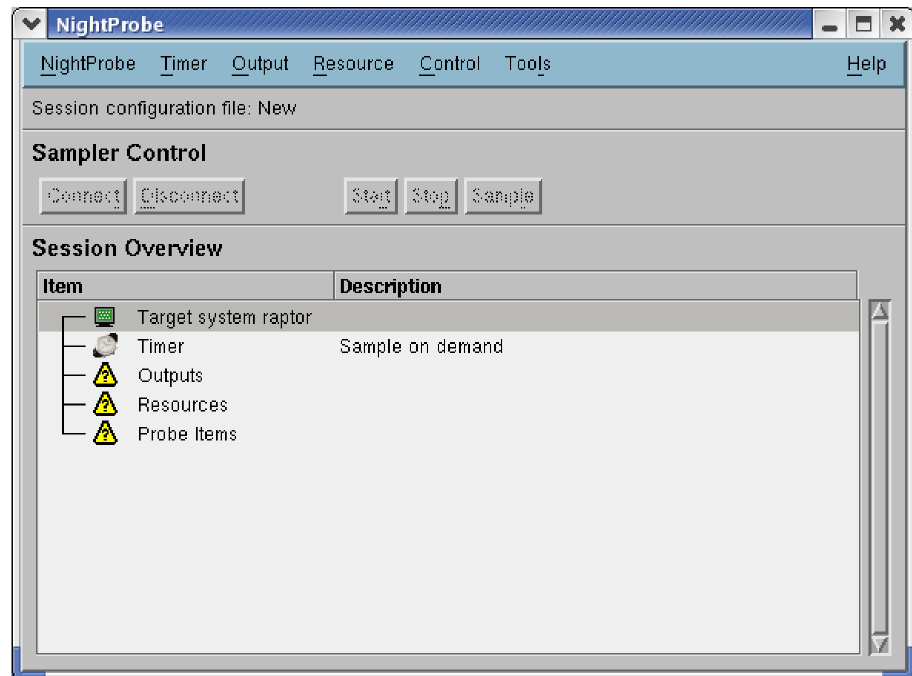


Figure 1-11. NightProbe Main window

Configuring NightProbe

Our example will use a configuration file shipped with the RedHawk NightStar Tools to configure NightProbe. This file, named **`nprobe.config`**, was copied to our local **`tutorial`** directory earlier in the step "Getting Started" on page 1-3.

To configure the NightProbe Data Monitoring Tool

- Select Open Session... from the NightProbe menu of the NightProbe Main window.

  You will be presented with a File Selection dialog.

- Maneuver to the local **`tutorial`** directory, if necessary.
- Select the file **`nprobe.config`** from the list of Files.
- Press Select to load the configuration file and dismiss the dialog.

You should see the following members of the `counters` class listed in the Probe Items list of the Session Overview area of the NightProbe Main window:

- `counters.cycle_time`
- `counters.i_counter`
- `counters.workload`

We will be probing and modifying these variables.

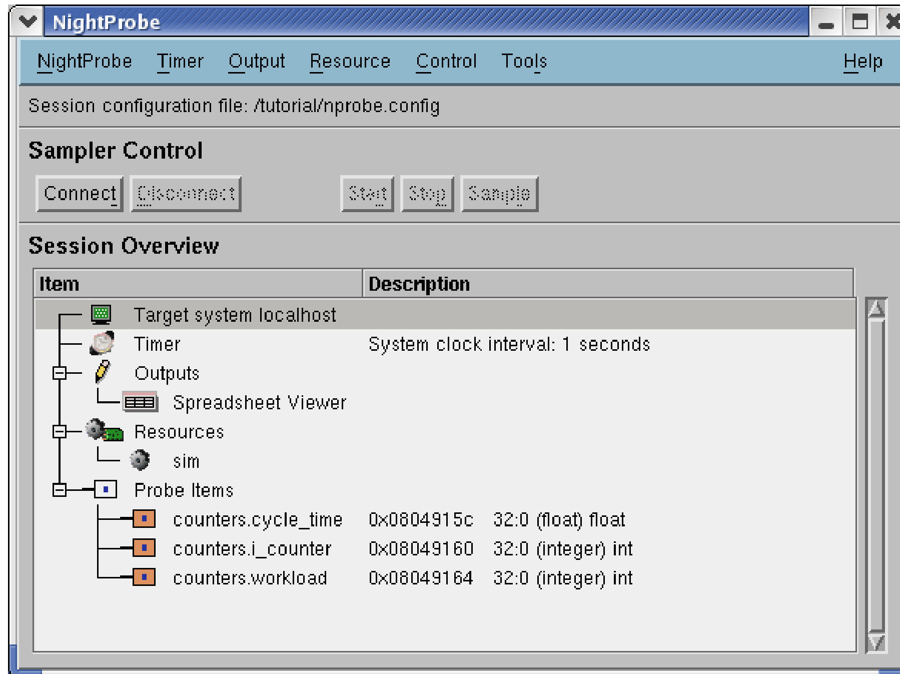


Figure 1-12. Configured NightProbe Main window

Since the **`nprobe.config`** file specifies that NightProbe is to direct its output to a spreadsheet window, the Spreadsheet Viewer window is automatically opened as well.

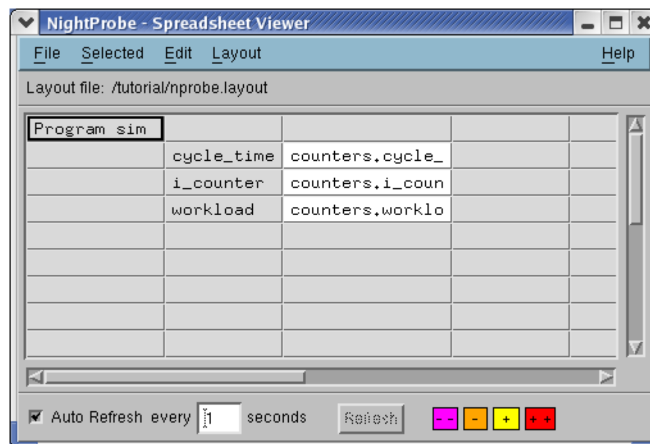


Figure 1-13. NightProbe Spreadsheet Viewer window

Connecting to the target program

When you are ready to perform data recording or monitoring, you must first connect to the target system and target system resources to be probed.

Initialization occurs during the connection phase - opening output devices, verifying target processes, and mapping target process variable addresses.

The probed applications are not affected by this operation.

To connect to the target program

- Press the Connect button in the Sampler Control area of the NightProbe Main window.
- If presented with the User Authentication dialog, enter the login name of the user on the target system in the User field along with the corresponding password in the Password field.
- Press the Login button to continue.

NOTE

The user authentication dialog is not presented if the target system matches the current hostname. The configuration file included with the tutorial explicitly specifies *localhost*, thereby requiring user authentication.

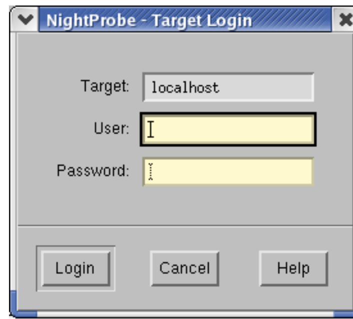


Figure 1-14. User Authentication dialog

Starting sampling

Once connected, we are ready to begin data recording.

Once started, the NightProbe server process will sample data based on the timing selection and will send the output to all specified output methods.

When we configured NightProbe (see "Configuring NightProbe" on page 1-17), we defined the timing selection to be the system clock (which fires once every second) and selected the Spreadsheet Viewer window as our output method.

To start sampling

- Press the Start button in the Sampler Control area of the NightProbe Main window.

  Note that the values in the Spreadsheet Viewer window will begin to change once a second.

  The user application that we are probing independently measures the time it takes for each cycle and saves that value in `counters.cycle_time`.

  Note that the value of `counters.cycle_time` (in units of seconds) is approximately the same as the Last Time statistic (in units of microseconds) in the NightSim Monitor window. (It will be slightly less than the value shown in the NightSim Monitor window because the application's calculations do not include all of its per-cycle activities.)

  Furthermore, the value of `counters.cycle_time` can also be seen in the NightView Monitor Window.

Modifying program data

NightProbe allows you to monitor and modify target locations while the program is running. We will modify the **sim** variable `counters.workload` to increase the amount of work the program does.

To modify the value of a variable

- In the Spreadsheet Viewer window, click on the value next to the label `workload`.
- Enter the value 10000.
- Press the Enter key.

Notice that the value for `cycle_time` has increased significantly. In our example, it is now approximately 0.00035 seconds (this value is dependent on your machine speed). (You can also see this reflected in the Last Time statistic in the NightSim Monitor window as well as in the `counters.cycle_time` monitorpoint in the NightView Monitor Window.)

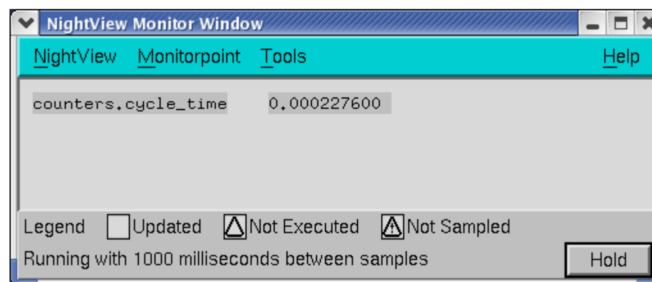


Figure 1-15. Modified values in NightView Monitor Window

In addition, the color of the cell containing the value of `cycle_time` has changed to yellow. NightProbe allows you to define caution and danger values for variables displayed in spreadsheets. Since the attributes for this cell (which were included in the configuration file **`nprobe.config`** - see "Configuring NightProbe" on page 1-17) specify that when the value exceeds 0.0002, the color of the cell will change to yellow signifying a state of high caution.

NOTE

On faster systems, you may have to choose a higher value for the `workload` in order to have the cycle time exceed 0.0002 seconds.

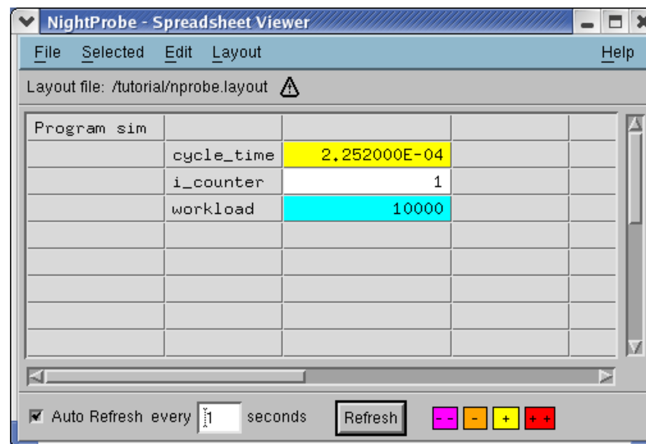


Figure 1-16. Modified values in NightProbe Spreadsheet Viewer

- Change the value of `workload` to 100000. Notice the color of the cell containing the `cycle_time` value changes to red, signifying a state of high danger.
- Change the value of `workload` back to 1000. Notice the color of the cell containing the `cycle_time` value changes back to white.

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. In addition, NightTrace can also log RedHawk Linux kernel events such as individual system calls, context switches, machine exceptions, page faults and interrupts. By combining application events with RedHawk Linux kernel events, NightTrace presents a synchronized view of the entire system. Furthermore, NightTrace allows users to zoom, search, filter, summarize, and analyze those events in a wide variety of ways.

Using NightTrace, users can manage multiple user and kernel NightTrace daemons simultaneously on multiple target systems 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

To invoke NightTrace from NightProbe

- From the Tools menu of the NightProbe Main window, select the NightTrace Analysis menu item.

  The NightTrace Main Window is opened.

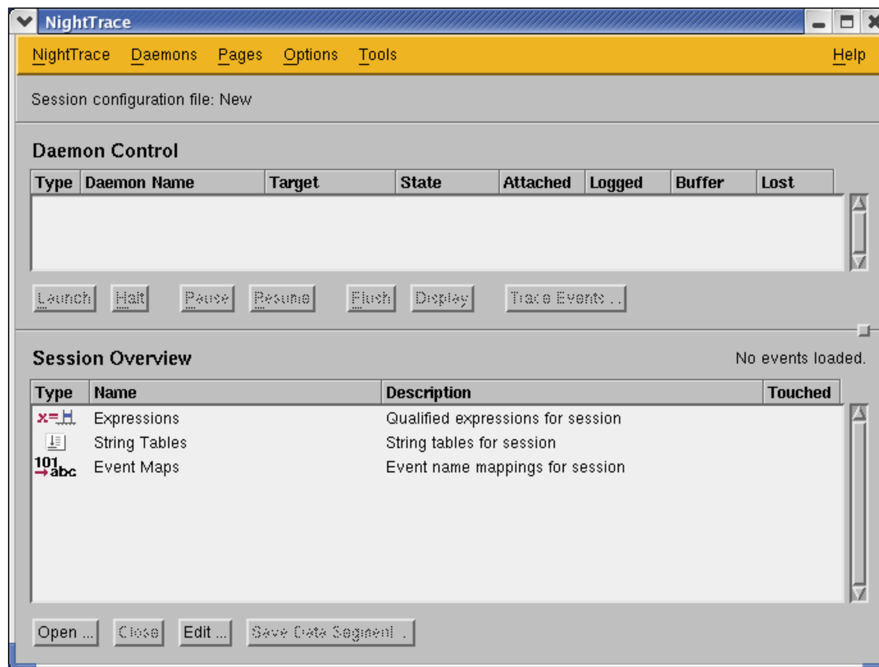


Figure 1-17. NightTrace Main Window

For more information on the NightTrace Main Window, see the chapter titled "Using the NightTrace Main Window" in the *NightTrace User's Guide* (0890398).

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
- NightProbe (see the description of the To NightTrace menu item in the chapter titled "Using the Data Recording Window" in the *NightProbe User's Guide* (0890480).

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

To configure a user daemon

- From the Daemons menu on the NightTrace Main Window, select the New... menu item.

  The Daemon Definition dialog is displayed.

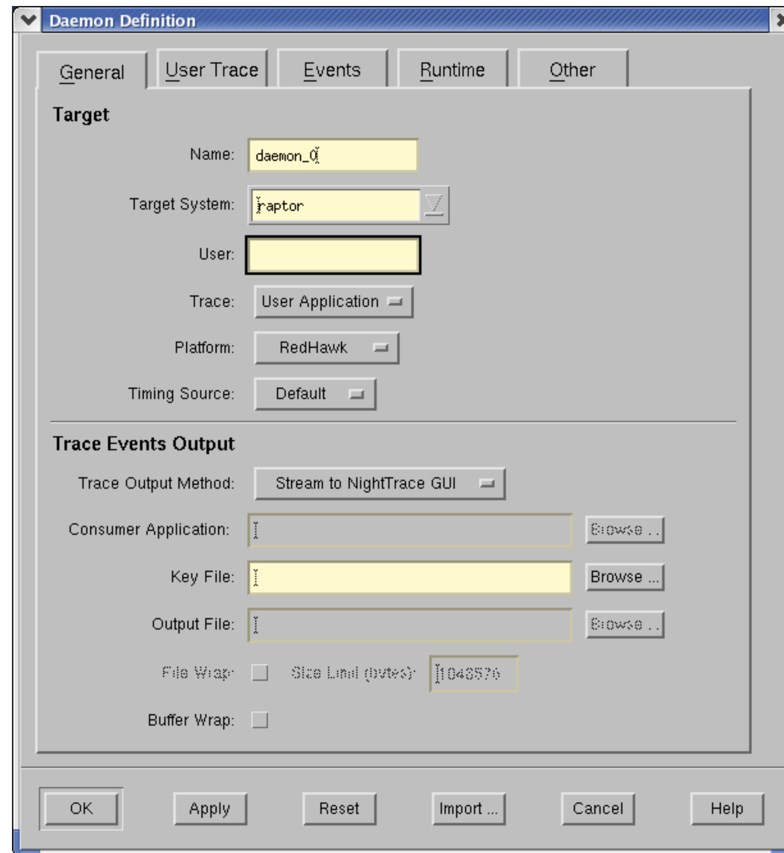


Figure 1-18. Daemon Definition dialog

- Press the `Import...` button at the bottom of the Daemon Definition dialog.

  You will be presented with a Login dialog.

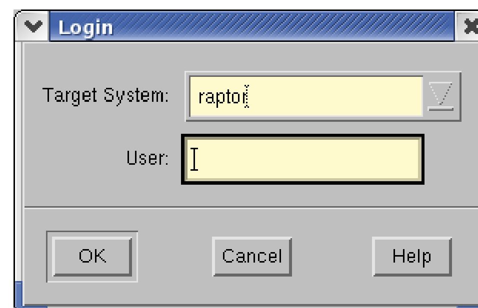


Figure 1-19. Login dialog

- Enter the name of the system on which the **`sim`** application is running in the `Target System` field.
- Enter your login name on that system in the `User` field.

- Press the OK button.

The Import Daemon Definition dialog is presented.

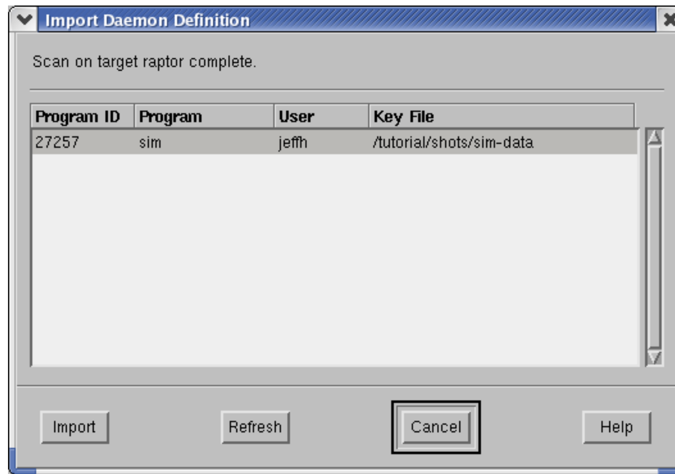


Figure 1-20. Import Daemon Definition dialog

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

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

The Import Daemon Definition dialog closes and the Daemon Definition dialog is populated with the imported attributes.

- Press OK on the Daemon Definition dialog to complete the configuration of the user application daemon.

Creating a customized display page

Now that we have configured our user application daemon, we can create a NightTrace display page in which we will view our trace data.

For this example, we would like to use a customized display page so we will use the configuration file shipped with the RedHawk NightStar Tools. This file, named **ntrace.config**, was copied to our local **tutorial** directory earlier in the step "Getting Started" on page 1-3.

To create a customized display page

- Press the Open... button at the bottom of the NightTrace Main Window.

You will be presented with an Open Display File dialog.

- Select the file **ntrace.config** from the list of Files.
- Press the OK button to create the display page as specified by the configuration file.

The customized NightTrace display page is presented.

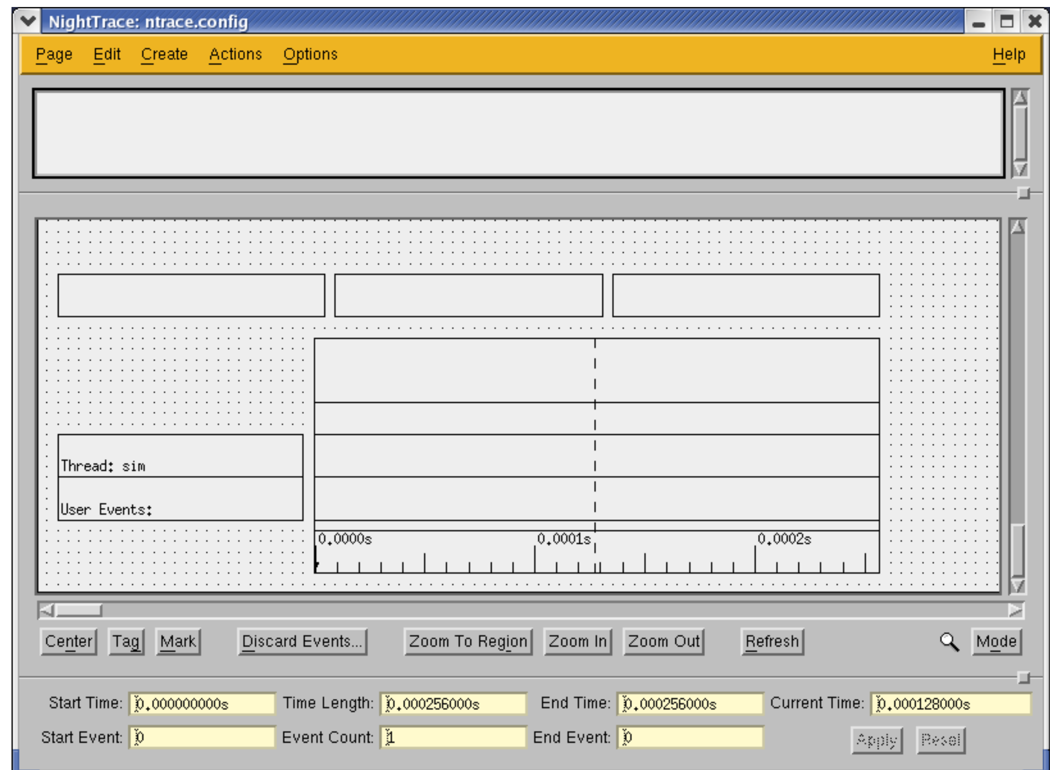


Figure 1-21. Customized NightTrace display page

Creating the user application daemon

Once the user application daemon is configured, it must be created before it can begin collecting events.

To create the user application daemon

- Select the user application daemon in the Daemon Control Area of the NightTrace Main Window.
- Press Launch.

The user application daemon is now created and ready to capture data. Note that the daemon is in a Paused state.

NOTE

Starting a daemon does not imply that the daemon begins to collect events.

Resuming execution of the user application daemon

Now that the daemon is configured and created, waiting in a Paused state, we may resume its execution so that it may begin collecting events.

To resume execution of the user application daemon

- Select the user application daemon in the Daemon Control Area of the NightTrace Main Window.
- Press Resume.

  The state of the daemon changes from Paused to Logging as it begins to collect trace data.

Displaying the user trace data

Now that we have our customized display page, we can display the user trace data.

To display the user trace data

- Press the Zoom Out button on the user display page repeatedly until data fills the grid area. You should see a saw-toothed pattern similar to the one shown in the figure below. You may need to repeatedly press the Right Arrow key to bring the data into view.

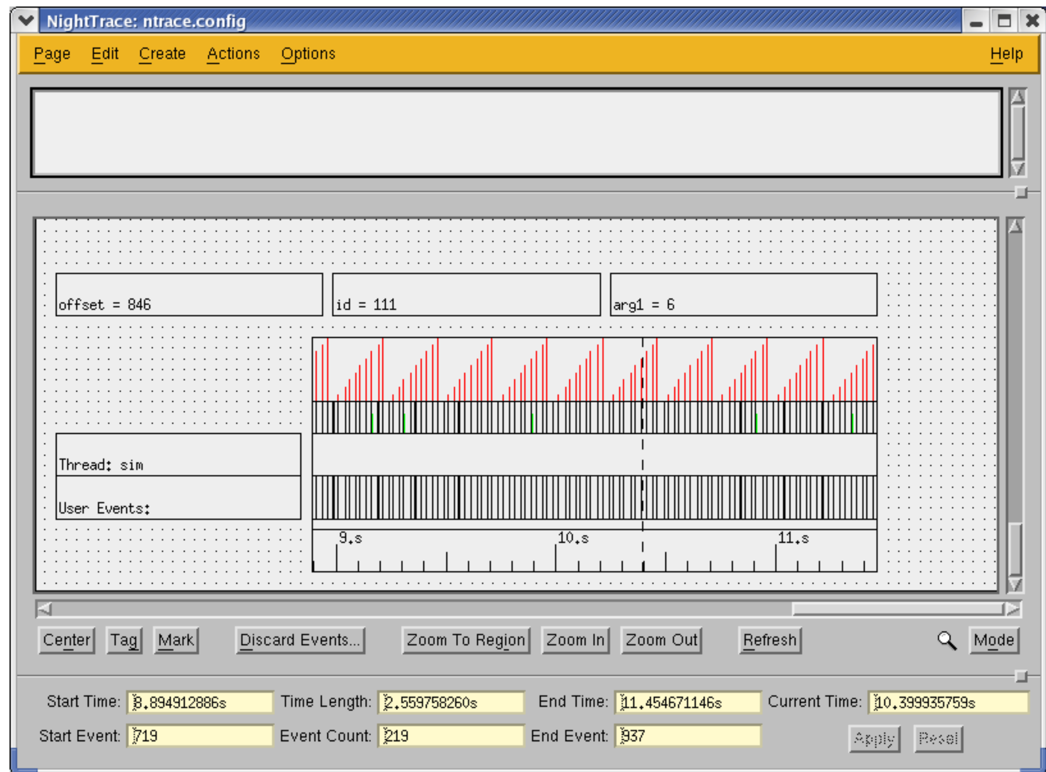


Figure 1-22. User trace data in customized NightTrace display page

The data may be appear differently depending on how closely you zoom in or out. Use the Zoom In and Zoom Out buttons (or the Down Arrow and Up Arrow keys, respectively), to adjust the zoom settings.

Use the left and right scrollbar arrows (or the Left Arrow and Right Arrow keys, respectively), to move the display interval over the data set.

NOTE

This display page is configured to only display events from the user application.

Inserting a patchpoint

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

In our example, we will insert a patchpoint in the loop to change the value of the `arg` variable in order to modify the output of the trace data:

```
arg = counters.Get() % 10;
```

```
trace_event_arg (cycle_end, arg);
```

To insert a patchpoint in a program

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

  ```
  trace_event_arg (cycle_end, arg);
  ```

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

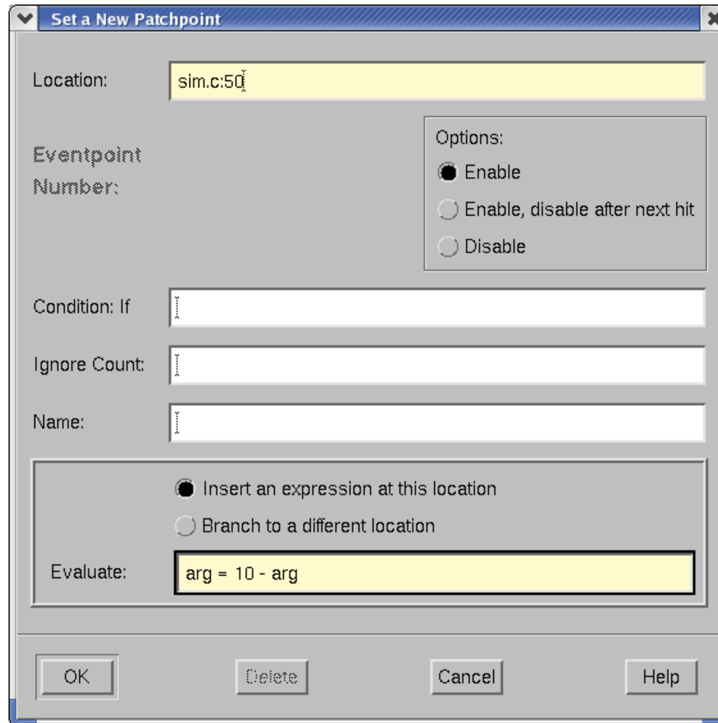


Figure 1-23. Setting a new patchpoint

- Enter the expression:

  ```
  arg = 10 - arg
  ```

  in the Evaluate field.

- Press OK.

NOTE

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

patchpoint at *line_number* **eval arg = 10 - arg**

where *line_number* coincides with the line:

```
trace_event_arg (cycle_end, arg);
```

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

Viewing streaming trace output

Now that we've modified the behavior of the program using patchpoints in NightView (see "Inserting a patchpoint" on page 1-29), we can see the effect our change has on the output of the user trace data.

Since the user trace daemon was configured to stream the output directly to the Night-Trace display buffer, we may view it immediately even while additional trace data is being collected.

To view streaming data

- On the Interval Scroll Bar under the grid on the NightTrace display page, press the right arrow continually until you see the shape of the saw-tooth pattern change from an ascending pattern to a descending pattern as shown in the figure below. (See the section titled "The Interval Scroll Bar" in the chapter "Viewing Trace Event Logs with ntrace" in the *NightTrace User's Guide* (0890398).

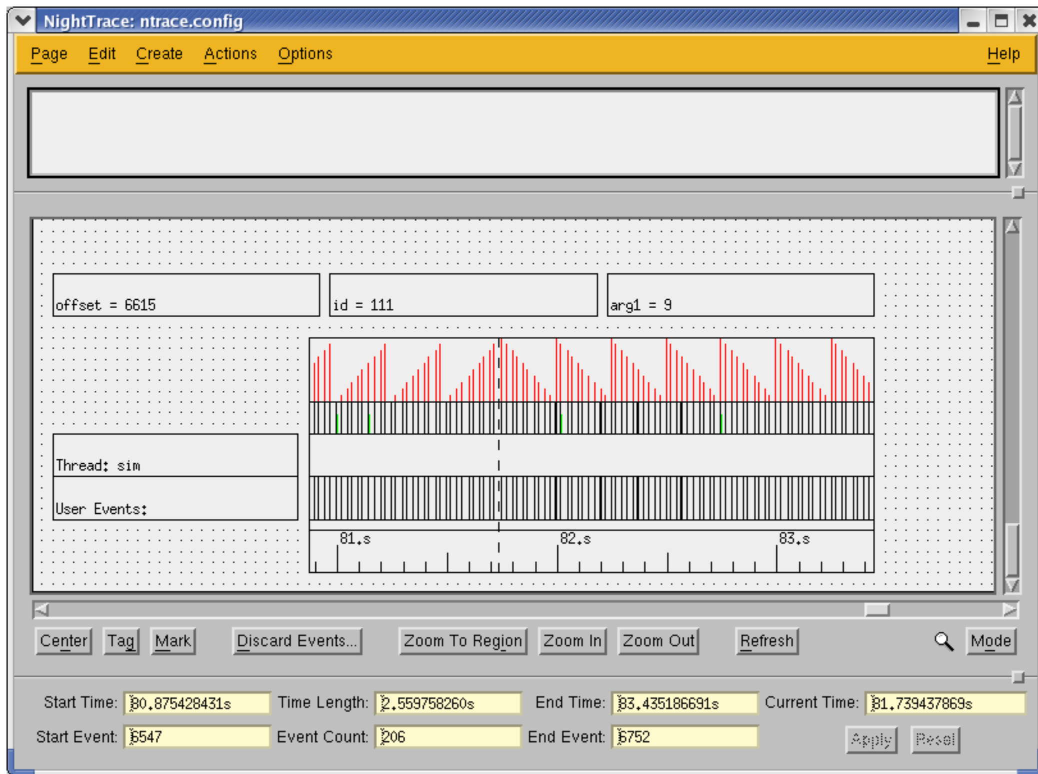


Figure 1-24. User trace data after patchpoint inserted

NOTE

We've just modified the path and behavior of our real-time application *without stopping it or causing it to miss any deadlines* - just one of the many features of NightView!

Configuring a kernel daemon

NightTrace allows the user to configure a kernel daemon to collect data about the execution time of interrupts, exceptions, system calls, context switches, and I/O to various devices.

To configure a kernel daemon

- From the Daemons menu on the NightTrace Main Window, select the New... menu item.

The Daemon Definition dialog is displayed.

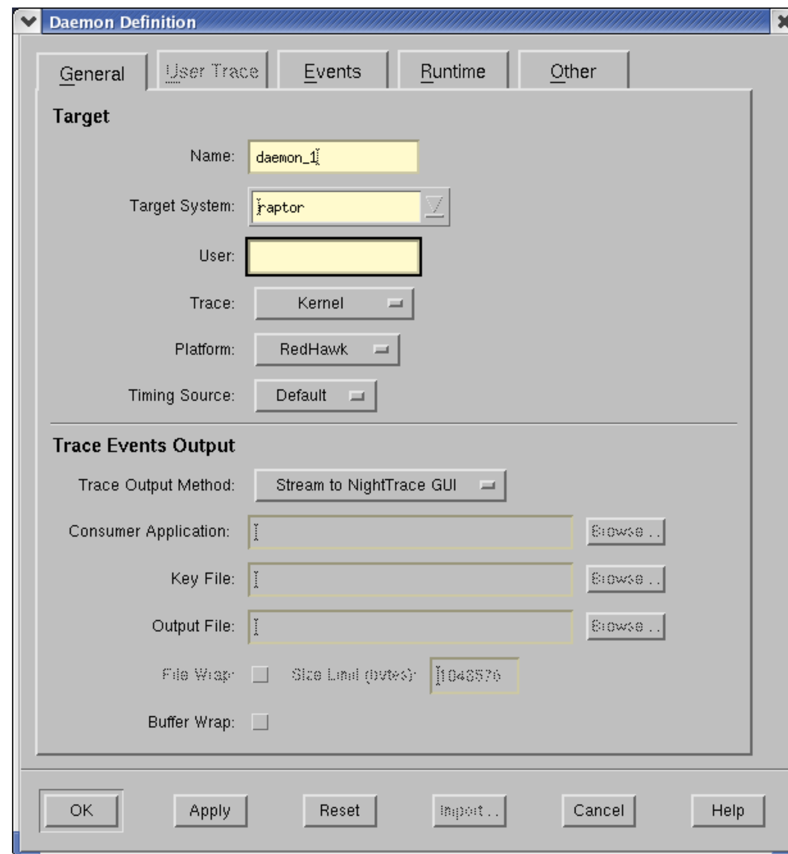


Figure 1-25. Daemon Definition dialog

- Select Kernel from the Trace drop-down menu located in the Target section on the General page to indicate that we want this daemon to collect kernel events.
- Press OK to complete the configuration of this daemon.

Creating the kernel daemon

Once the daemons are configured, they must be created before they can begin collecting events.

To create the daemons

- Select the kernel daemon in the Daemon Control Area of the NightTrace Main Window.
- Press Launch.

The kernel daemon is now created and ready to capture data. Note that the daemon is in a Paused state.

In addition, a NightTrace kernel display page appears.

NOTE

The page may differ in format depending on the number of CPUs on your system.

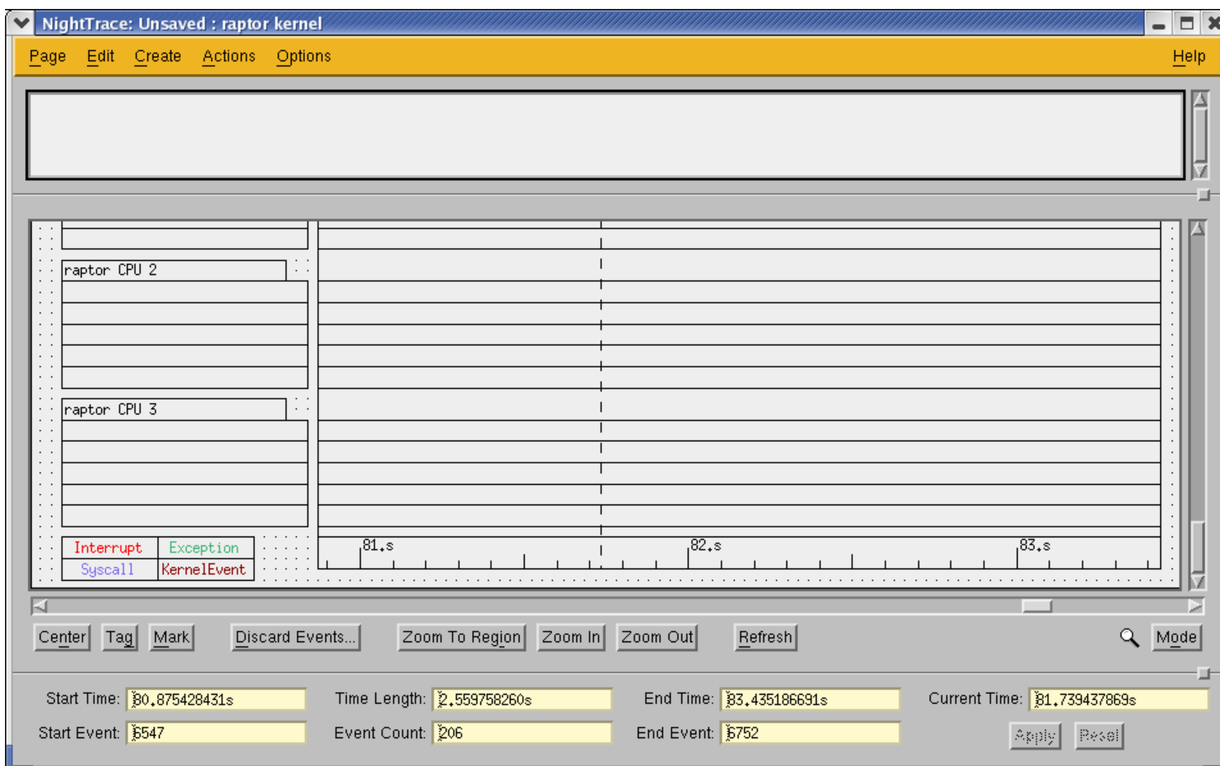


Figure 1-26. NightTrace kernel display page

NOTE

Starting a daemon does not imply that the daemon begins to collect events.

Resuming execution of the kernel daemon

Now that the kernel daemon is configured and created, waiting in a Paused state, we may resume its execution so it may begin collecting events.

To resume execution of the kernel daemon

- Select the kernel daemon in the Daemon Control Area of the NightTrace Main Window.

IMPORTANT

The current activity on the system has a drastic effect on how much data will be collected. Streaming data for a few seconds on a busy system may collect hundreds of thousands of kernel events while on a fairly idle system it may take a few minutes to reach that level.

- Press Resume.

  The state of the daemon changes from Paused to Logging as it begins to collect trace data.

NOTE

You may display the kernel data as it is streaming. See “Displaying the kernel trace data” in the following section.

- When the value in the Logged column reaches around 50000 events, press the Pause button.

Displaying the kernel trace data

As we are collecting trace data from the RedHawk Linux kernel, we can display that data in the NightTrace kernel display page.

To display the kernel trace data

- Select *only* the kernel daemon in the Daemon Control Area of the NightTrace Main Window (as indicated by the K in the Type column).
- Press Display.

  When data from the selected daemon(s) is being streamed to the NightTrace display buffer (as specified by setting the Trace Output Method on the General page of the Daemon Definition dialog to Stream to NightTrace GUI), pressing this

button causes a flush of the data currently in the trace buffer to the NightTrace display buffer.

Flushing the trace data

To flush the trace data

- Select *both* daemons from the Daemon Control Area of the NightTrace Main Window) by clicking on the top daemon and then clicking on the bottom daemon while the Shift key is depressed.
- Press Flush.

  This flushes any remaining trace events from the buffers associated with the daemons currently selected in the Daemon Control Area to the NightTrace display buffer. (If our trace data was being output to output files, the trace events would be flushed to those files.)

Stopping the daemons

Once we are finished accumulating enough data from the daemons, we can stop them.

To stop the daemons

- Ensure that *both* daemons are selected in the Daemon Control Area of the NightTrace Main Window.
- Press Halt.

  The state of both daemons changes to Halted.

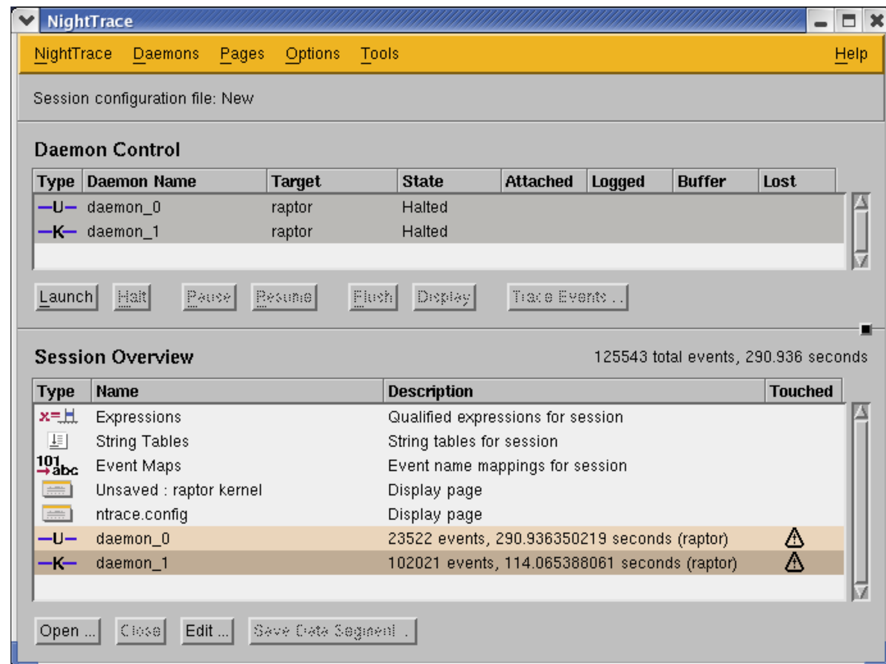


Figure 1-27. NightTrace Main window showing halted daemons

Positioning the current time line

We will position the current time line to a point somewhere after the kernel trace data started being generated.

To position the current time line

- On the Interval Scroll Bar under the grid on the *kernel display page*, press the right arrow continually until data appears in the grid area. (See the section titled “The Interval Scroll Bar” in the chapter “Viewing Trace Event Logs with ntrace” in the *NightTrace User’s Guide* (0890398).
- Click in the center of the data displayed in the grid area of the kernel display page.

  Note the information regarding interrupts, exceptions, system calls, and kernel events on each CPU displayed in the DataBoxes on the left side of the grid area.

NOTE

Depending on your zoom settings, the kernel data may be barely visible on the far right-hand side of the display page. This can occur if the user daemon has been running for a long time. For example, if the user application has been running for 10 minutes during this tutorial, and you capture just a few seconds of kernel

data, the relative portion of the display containing the kernel data may be miniscule if the display is zoomed out. If this occurs, click very near the far right edge in the kernel display area, zoom in and move the interval to the right using the Right Arrow key or right scrollbar arrow until the kernel data appears.

- Zoom in until details appear as shown in the following figure.

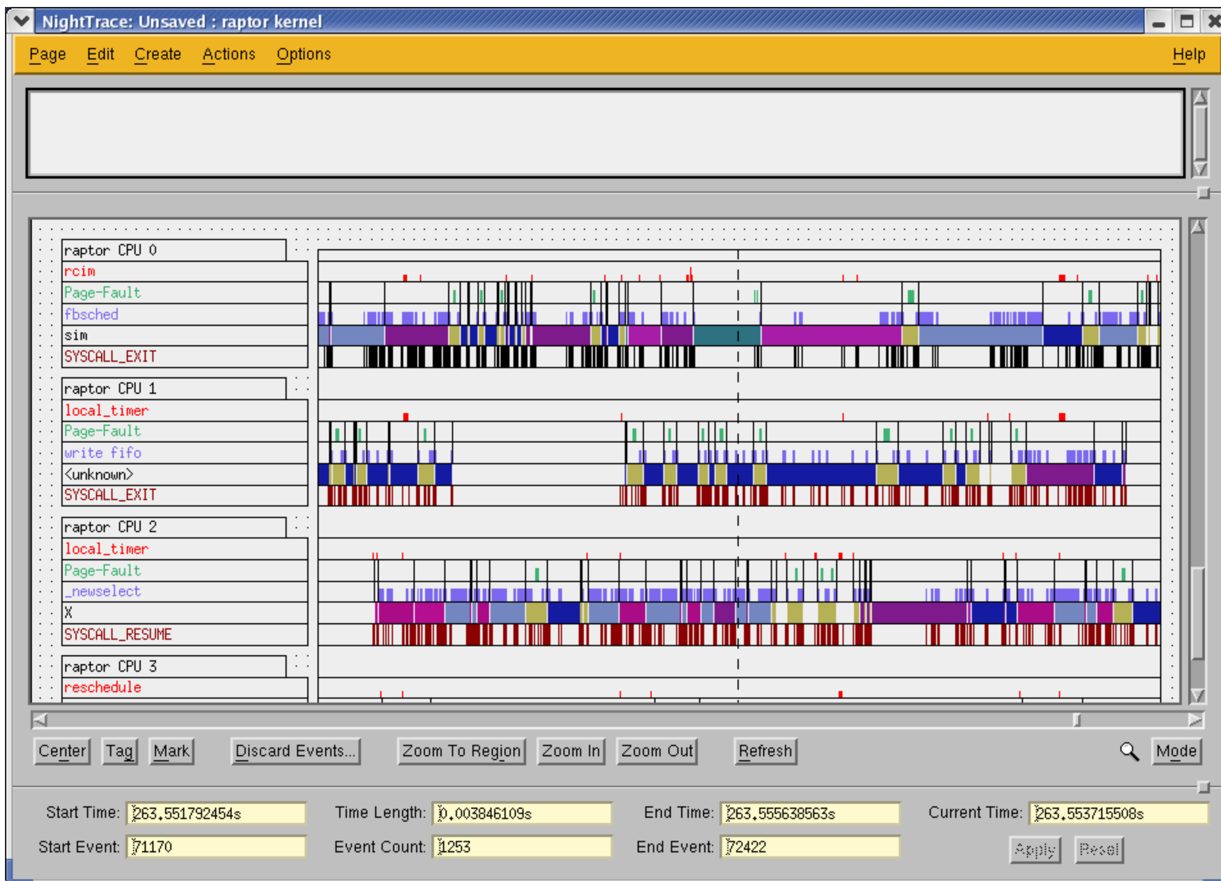


Figure 1-28. NightTrace kernel trace data

The DataBoxes are updated based on the current position of the current time line and indicate the last value of each data item that occurred on or before the time line on each CPU.

NOTE

By default, user events are not displayed on this page even though they may exist in the same interval.

Loading an eventmap file

Eventmap files map ASCII trace event names with numeric trace event IDs allowing the user to reference events based on mnemonic tags or meaningful labels.

An eventmap file, **ntrace.eventmap**, was copied from the **/usr/lib/NightStar/tutorial** directory to our working directory in the step "Getting Started" on page 1-3. This file contains a mapping of trace event names to the trace events IDs logged in our user application.

We will load that eventmap file now so that we can refer to those event names in the next section, "Searching for a user trace event".

To load an eventmap file

- Press the Open... button at the bottom of the NightTrace Main Window.

  You will be presented with an Open Display File dialog.

  - Select the file **ntrace.eventmap** from the list of Files.
  - Press the OK button to load the eventmap file.

Searching for a user trace event

To search for a user trace event

- Select the Change Search Criteria... menu item from the Actions menu on the NightTrace display page containing the user trace data.

  The Search NightTrace Events dialog is presented.

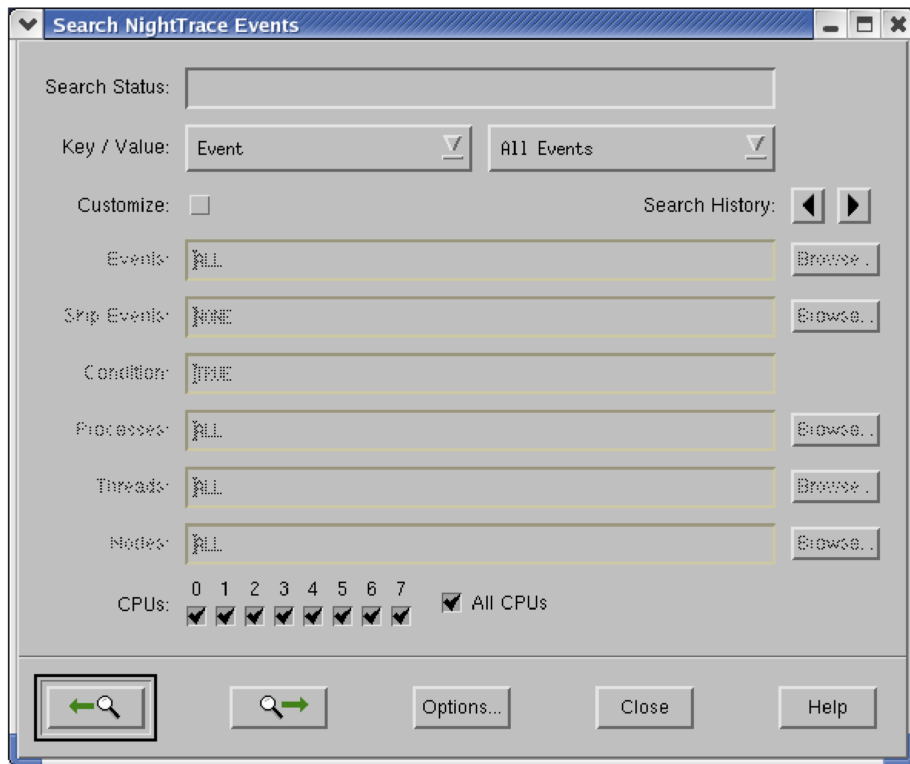


Figure 1-29. Search NightTrace Events dialog

- Select cycle_start from the list of events in the Value pull-down menu list.

  In **sim.c** (see “sim.c” on page A-2), we log a trace event immediately when we start our cycle (exiting fbswait):

```
trace_event_arg (cycle_start, counters.Get());
```

NOTE

Because we loaded the **ntrace.eventmap** file (“Loading an eventmap file” on page 1-39), we are able to specify the more meaningful event name, cycle_start, in the Event List field instead of the numeric trace event ID (110).

- Press the forward search button (represented by the right-hand green arrow).
- Press the Close button to dismiss the Search dialog.

  Both display pages are positioned at the first occurrence in our data which meets our search criteria.

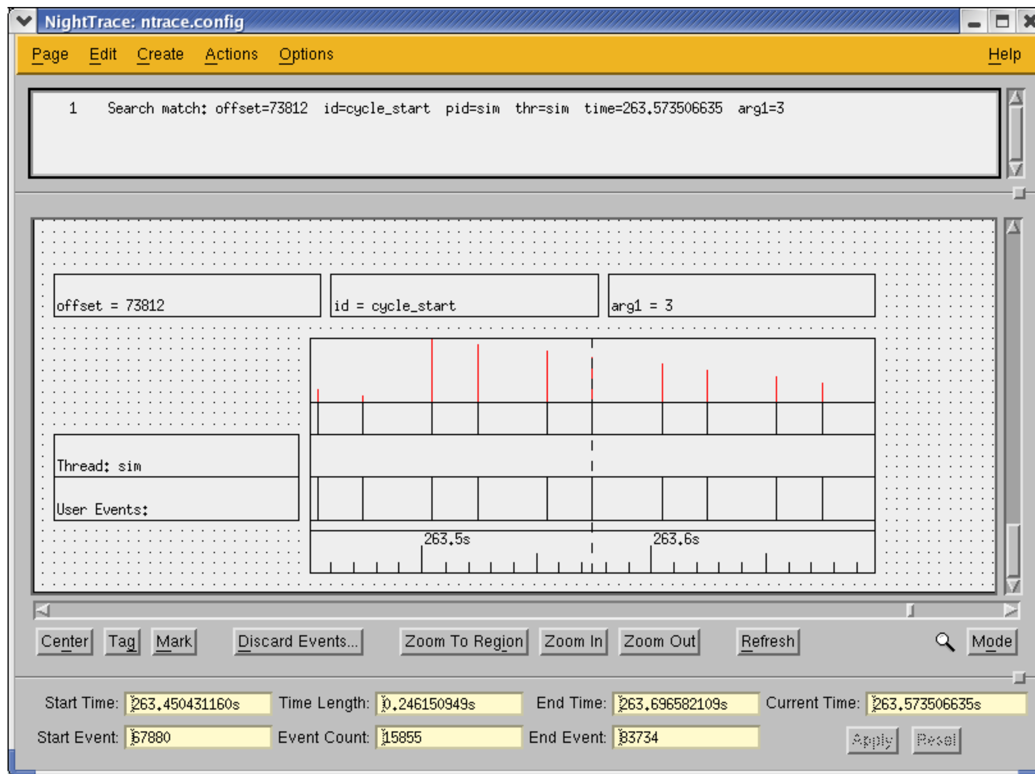


Figure 1-30. User trace data after search

Zooming in

We can see a finer level of detail by zooming in on the user trace display page.

To zoom in

- Press the Zoom In button repeatedly until two black vertical lines with a green bar between them appears.

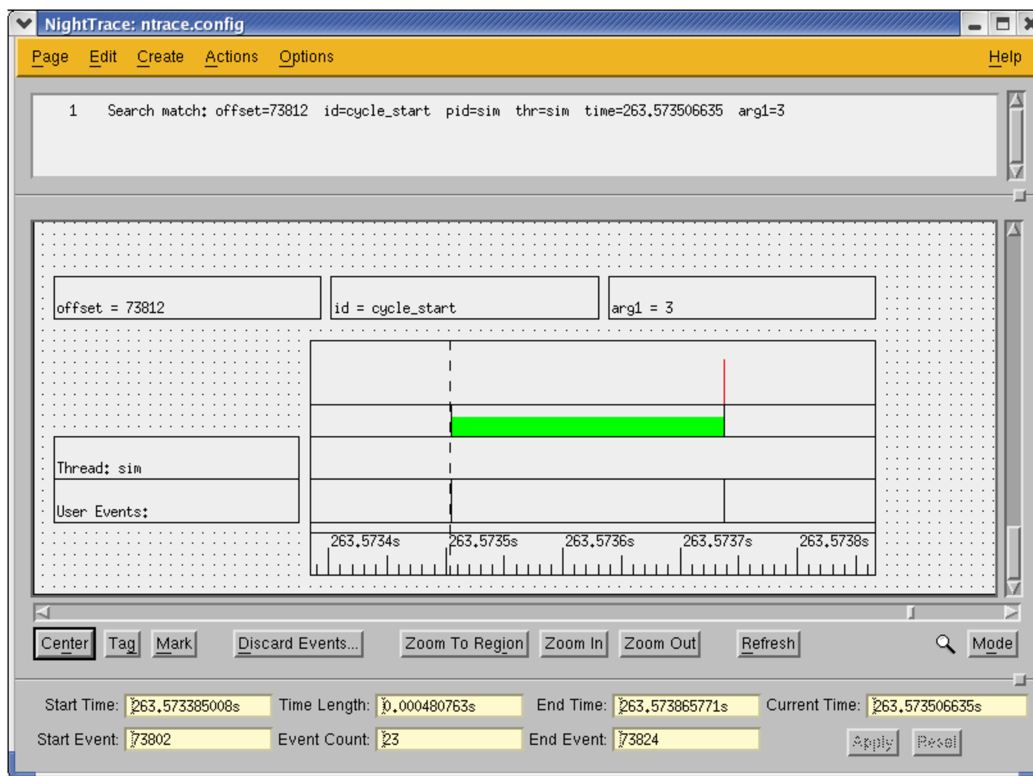


Figure 1-31. Zoomed in view of user trace data

Remember that our program logs a trace event immediately when we start our cycle (exiting `fbswait`), then it performs some calculations using `counters.work`, and finally it logs another trace event when it is finished before returning to the `fbswait` call at the top of the loop. (See “sim.c” on page A-2.)

The black lines represent the individual events logged in the application by the `trace_event_arg()` API calls. The green bar is a state graph; the start of the state is defined to be the `cycle_start` event logged when we begin our cycle (event #110) and the end of the state is defined by `cycle_end` (event #111) which is logged when we complete our cycle.

The red line that appears at the end of the state graph is an entry in a datagraph whose value is that of the argument logged with the `cycle_end` event in the second `trace_event_arg()` call. (This value which ranges from 1 to 9).

Examining the kernel trace data

Now let’s take a look at the kernel trace data to see how it coincides with the user trace data.

NOTE

NightTrace automatically synchronizes all display pages so that every display page shows the same time frame. Thus, our kernel display page reflects the system activity corresponding to the time period displayed in our user trace display page.

NOTE

If you do not see any kernel data in the kernel display page, it is likely that the search in the previous section located user data that was present before you stared the kernel daemon. Go back to the section titled “Positioning the current time line” on page 1-37 and ensure that the current time line is positioned in the display interval where kernel data has been retrieved and then try the search again.

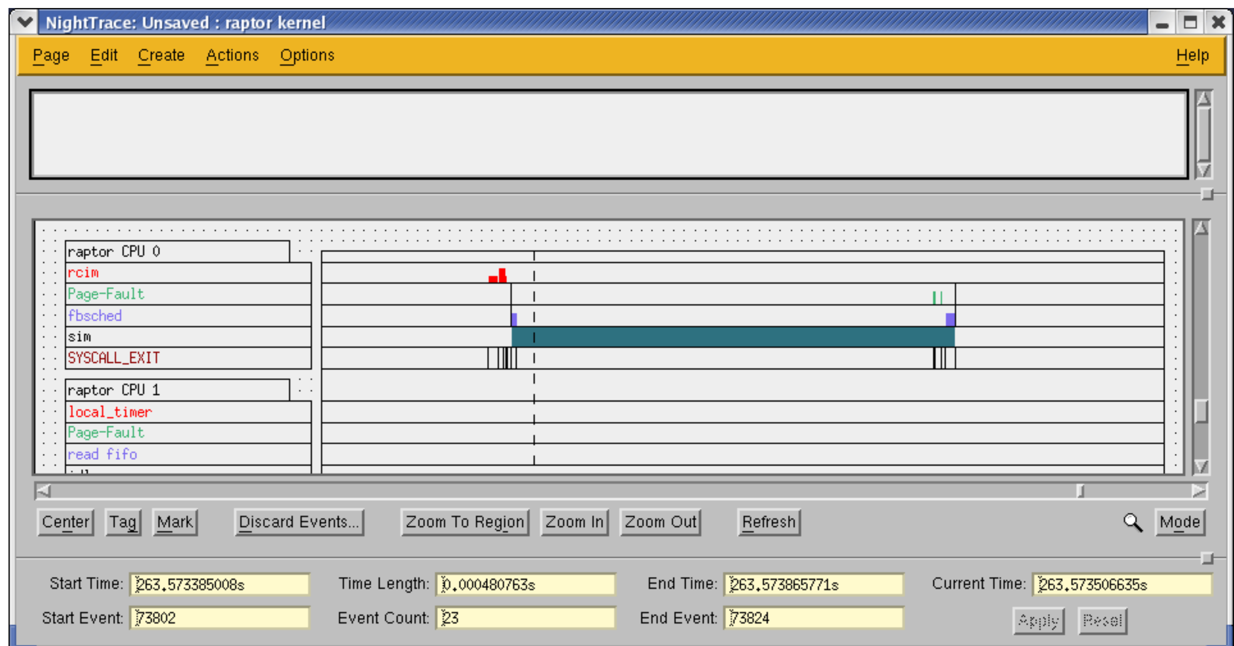


Figure 1-32. Zoomed in view of kernel display page

NOTE

The following analysis of the kernel trace data is based on Figure 1-32. If you are analyzing live data, your kernel display page may look different. You may see additional activity, most likely interrupt activity, between the exit and reentry to the `fbswait` API call (which corresponds to the `fbsched` system call on CPU 0 in Figure 1-32).

In Figure 1-32, the first red bar displayed on the grid for CPU 0 to the left of the current time line (represented by the dashed vertical line) indicates the interrupt from the RCIM device. (Note that if you collected your own kernel data, the CPU where the interrupt occurred could be on a different CPU in which case it may not be the first red bar to the left of the current time line, but it will be close.)

A context-switch then occurs on CPU 0 as indicated by the first black vertical line to the left of the current time. The blue bar following that first black line is the `fbsched` system call. In our source code, this is when we exit the `fbswait` call.

The application then performs its calculations (as indicated by the colored bar on the pid row) before it comes back to the `fbswait` call (the second blue bar).

The lack of any activity in the white space in the interrupt and system call rows indicates that the user application did not make any intervening system calls and was not disturbed by some other interrupt or exception.

The solid bar in the pid row indicates the time where the process **`sim`** was assigned to the CPU (executing in the kernel during the system calls or interrupt processing and executing in user space otherwise).

For optimal application performance, we will use NightTune to shield the CPU where **`sim`** executes from interrupts and other processes.

NOTE

If your system has only one CPU, continue to “Exiting the Tools” on page 1-52 .

Using NightTune

NightTune is a graphical tool for analyzing system and application performance including CPU usage, context switches, interrupts, virtual memory usage, network activity, process attributes, and CPU shielding. NightTune allows you to change the priority, scheduling policy, and CPU affinity of individual or groups of processes using pop-up dialogs or drag-and-drop actions. It also allows you to set the shielding and hyper-threading attributes of CPUs and change the CPU assignment of individual interrupts.

NOTE

NightTune may only be used on systems running RedHawk 2.1 or later. If your system is running an earlier version of RedHawk, continue to "Exiting the Tools" on page 1-52 .

Invoking NightTune

To invoke NightTune from a terminal session

- From a terminal session, launch NightTune using the predefined example configuration file as shown in the following command:

  `ntune -config /usr/lib/NightTune/config`

  The following window is displayed.

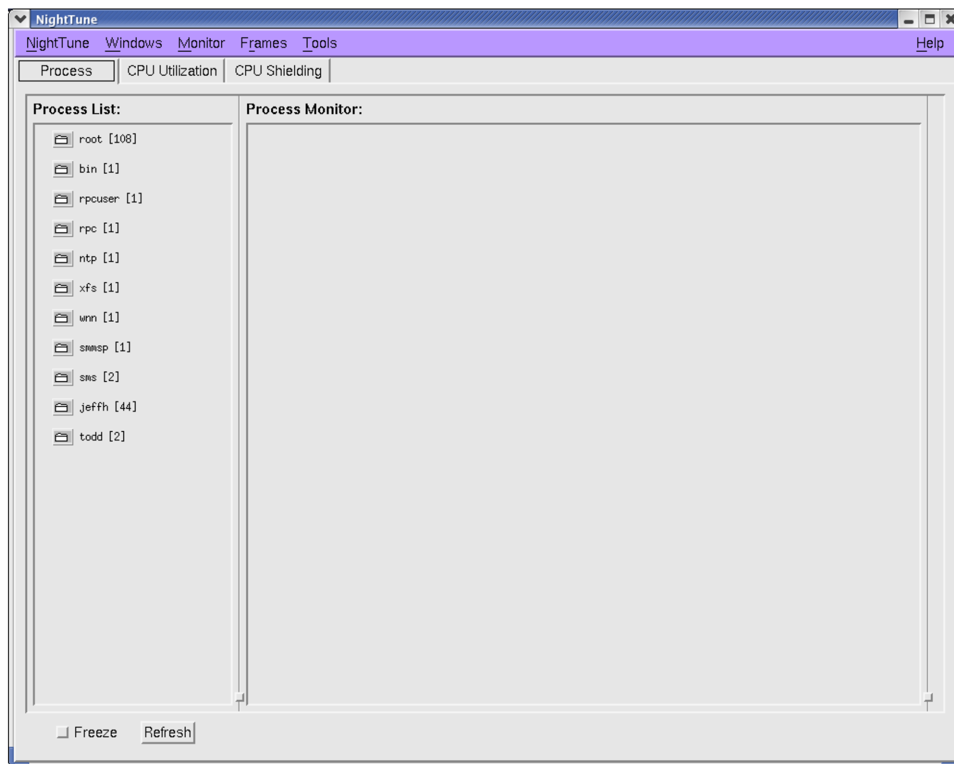


Figure 1-33. Example NightTune Window

- Click on the tab labelled `CPU Shielding` to display a page which includes panels describing interrupt activity and CPU shielding status.
- In the CPU panel click in the CPU box for CPU 0, click on the tab labelled `Processes`. The following window is displayed.

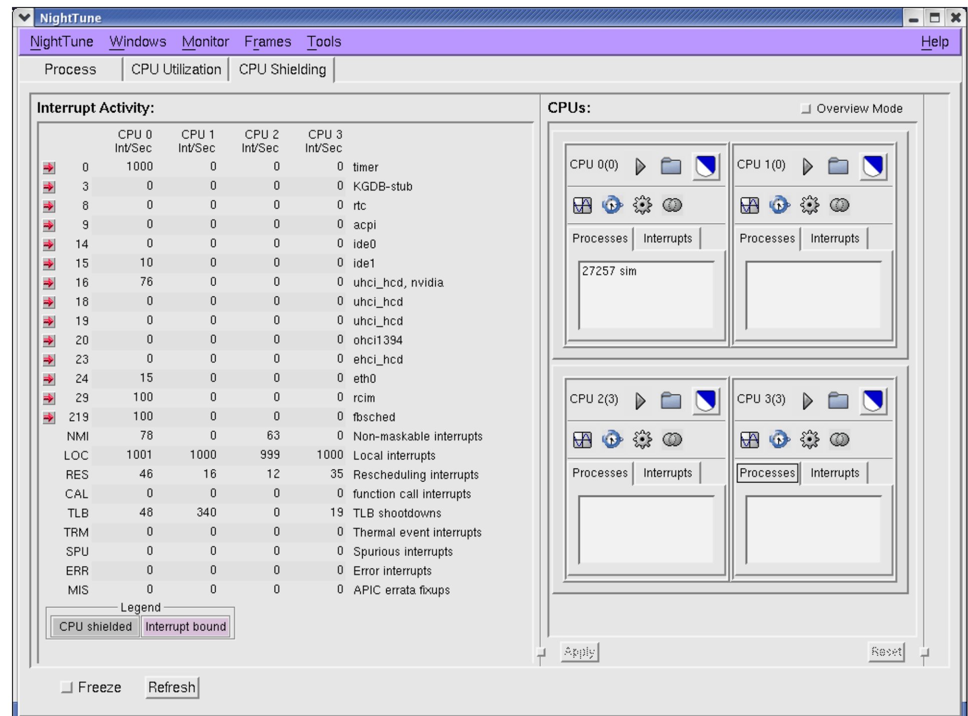


Figure 1-34. NightTune CPU Shielding Page

Each row in the Interrupt Activity area contains a red arrow icon, an IRQ value, the number of interrupts per second for each CPU corresponding to that IRQ, and a textual description of the devices using that IRQ.

In the CPU panel, a CPU box is shown for each virtual CPU with indicators describing the shielding status of each CPU. The Processes tab for CPU 0 includes the process ID and program name of processes bound to CPU 0. The **sim** process should appear in that list since its CPU affinity was set to CPU 0 during configuration of the Frequency Based Scheduler with NightSim.

Binding an Interrupt to a CPU

Our first step in tuning the system will be to bind the RCIM interrupt to CPU 0.

To bind an interrupt to a CPU

- In the Interrupt Activity area, locate the IRQ value associated with the RCIM device. The word rcim should be listed in the right hand column of one of the IRQ rows.
- Using the middle mouse button, click and hold anywhere in the row associated with the RCIM device. A new icon will appear. Drag the icon into the

CPU panel and release it when the icon is placed in the CPU box associated with CPU 0.

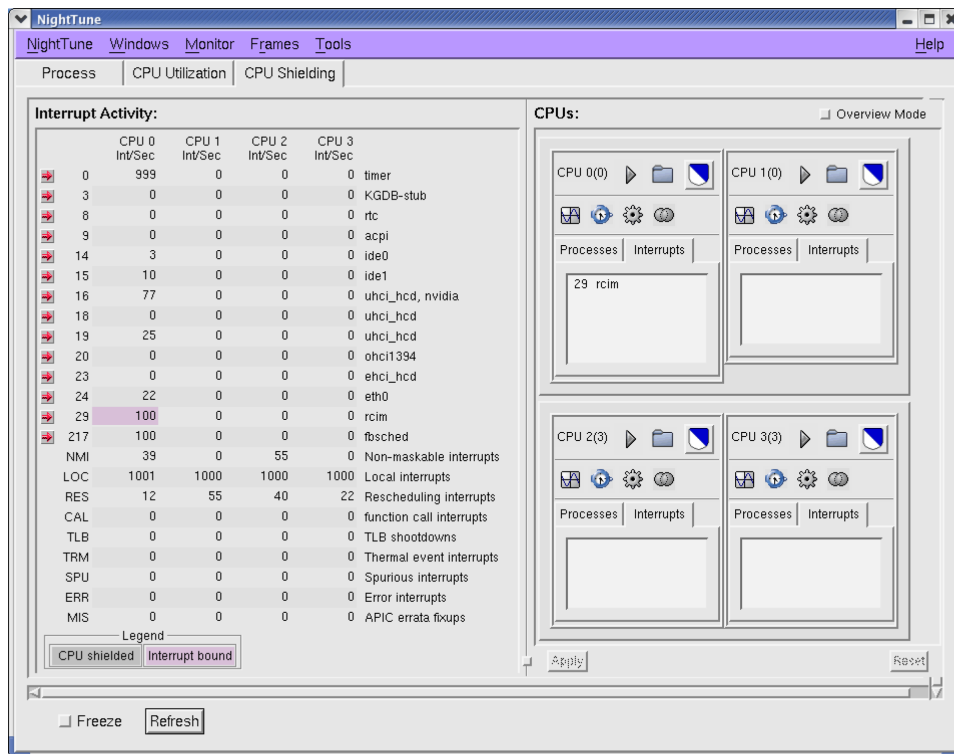


Figure 1-35. RCIM Interrupt Bound to CPU 0

The IRQ associated with the RCIM device is now bound to CPU 0.

The page has two indicators of the binding. In the Interrupt Activity pane, a purple background appears around the count of interrupts for the RCIM on CPU 0. Additionally, the CPU box for CPU 0 has raised the Interrupts list and the RCIM is included in the list

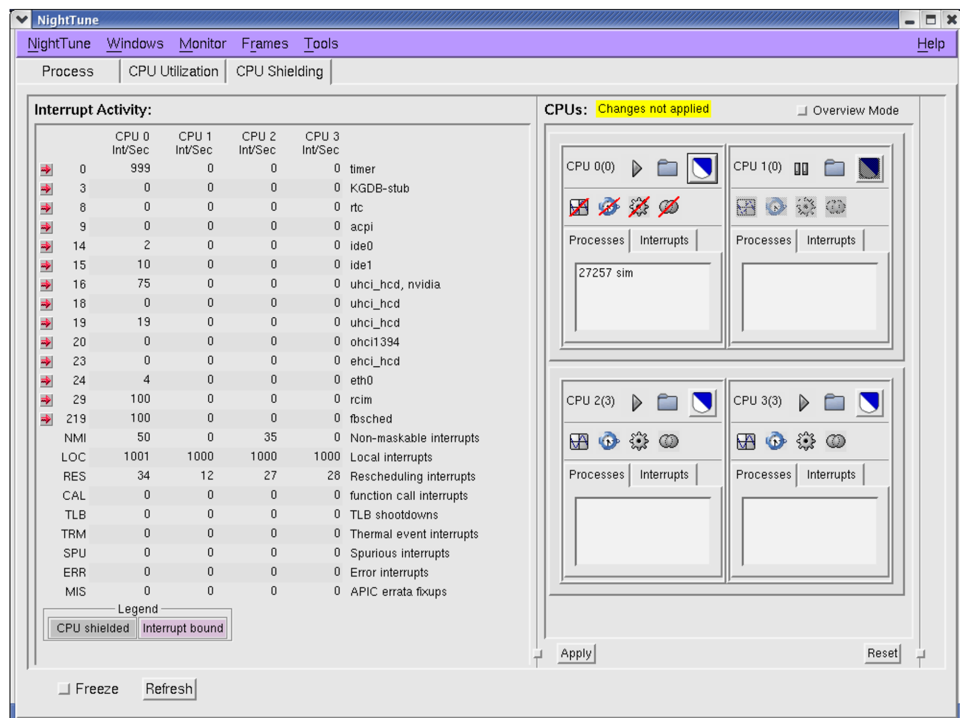
Alternatively, you can change the interrupt bindings for individual IRQs by clicking on the red arrow icon. This brings up a dialog which allows you to choose a single CPU or a set of CPUs that are allowed to field that interrupt

Shielding a CPU

The next step in tuning our system is to shield CPU 0 from all interrupts and processes except for those used in our tutorial.

To shield a CPU

- Press the Max Shield icon which is next to the Folder icon on the top row of icons in the CPU 0 box in the CPU panel. The window will change as shown in the following figure.



When Max Shield is pressed, NightTune automatically changes the CPU configuration in the following ways:

-

  The Interrupts icon for CPU 0 indicates 'shielded from interrupts'.

  Shielded from interrupts means that CPU 0 will only field interrupts that have specifically been bound to CPU 0 and CPU 0 only. By default, the CPU affinity of most interrupts includes all CPUs, so this setting effectively isolates CPU 0 to just the RCIM device (and other devices that use the same IRQ value).

-

  The Local Timer icon for CPU 0 indicates 'shielded from local timer interrupts'.

  Shielded from local timer interrupts means that CPU 0 will not field local timer interrupts.

-

  The Processes icon for CPU 0 indicates 'shielded from processes'

  Shielded from processes means that CPU 0 will only execute processes that have been specifically bound to CPU 0 and CPU 0 only.

NOTE

When we configured our application with NightSim, we bound our **sim** program to CPU 0; it will remain there even after these changes are applied.

-

  The Hyperthreading icon for CPU 0 indicates 'shielded from hyper-threading'.

  The Active icon for the hyper-threaded sibling CPU (CPU 1) indicates 'cpu down'.

  Marking the sibling hyper-threaded CPU as inactive prevents activity on that logical CPU, which shares the physical CPU with CPU 0.

NOTE

Not all systems include hyper-threading support; therefore, this action may not occur on your system.

NOTE

The hyper-threaded sibling CPU may not be logical CPU 1, it could be any logical CPU number. Hyper-threaded sibling CPUs are presented in horizontal pairs in the CPU panel.

- Press the Apply button to activate these changes.

  These configuration changes are not applied to the system until the Apply button is pressed.

NOTE

When Apply is pressed, the changes may be rejected if CPU 0's hyper-threaded sibling has processes or interrupts bound to it. If this is the case, drag the processes and interrupts shown in the sibling's Processes and Interrupts lists to another CPU and then Apply the Maximum Shielding change again.

Examining the kernel trace data after tuning

We will now capture some fresh kernel trace data to examine the effects of our tuning of CPU 0.

To examine the kernel trace data after tuning

- In the NightTrace Main window, select the kernel daemon from the Daemon Control area.
- Press Launch.

  The kernel daemon is now ready to capture data. Note that the daemon is in a Paused state.
- Press Resume.

  The state of the daemon changes from Paused to Logging as it begins to collect trace data.
- When the value in the Logged column reaches around 50000 events, press the Pause button.
- Press the Flush button.
- On the kernel display page, continually press the Zoom Out button until you see new kernel data on the right-hand side of the display page.
- Click in the middle of the new data and press Zoom In until more detail appears.

  See that the activity on CPU 0 is limited to the RCIM interrupt, followed by the execution of our **sim** application. No other processes or interrupts affect CPU 0 and CPU 0's hyper-threaded sibling (if any) shows no activity.

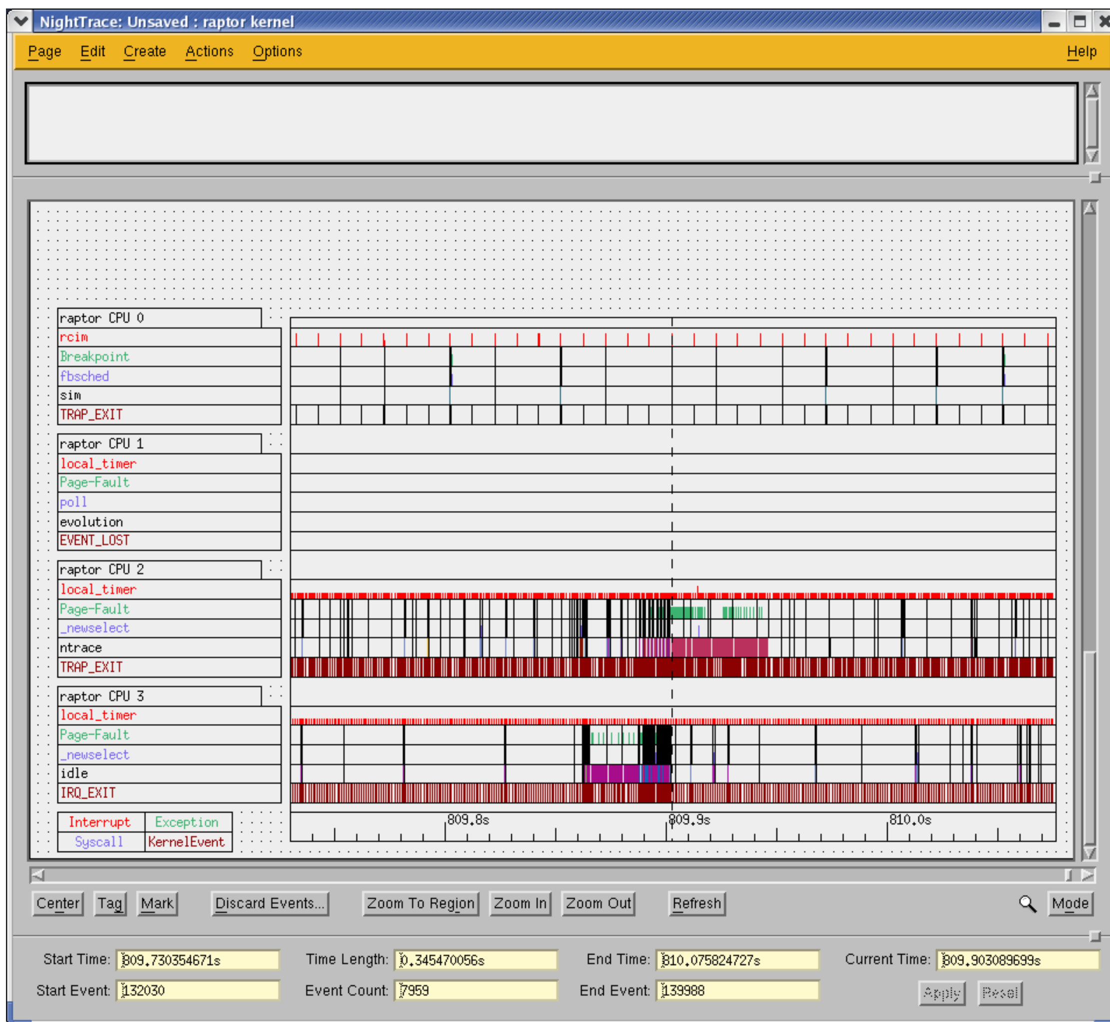


Figure 1-36. Kernel Display Page with shielded CPU

Exiting the Tools

In conclusion of our tutorial, we will exit each of the tools.

Exiting NightTune

To exit NightTune

- From the NightTune System Activity Window, select Exit from the File menu.

Exiting NightTrace

To exit NightTrace

- From the NightTrace Main Window, select Exit Immediately from the NightTrace menu.

Exiting NightProbe

To exit NightProbe

- From the NightProbe Main window, press the Stop button to stop sampling data.
- Press the Disconnect button to disconnect from the application.
- From the NightProbe menu, select Exit.
- When NightProbe presents the warning dialog asking if you would like to save configuration changes, press No.

Exiting NightSim

To exit NightSim

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

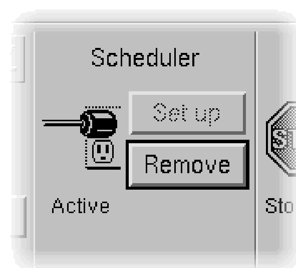


Figure 1-37. Removing the scheduler

You will be presented with the following dialog:

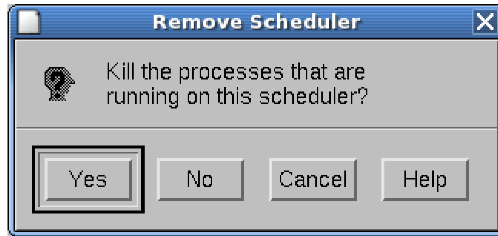


Figure 1-38. Remove Scheduler dialog

- Press Yes to kill the processes that are currently scheduled on the scheduler.
- From the NightSim menu, select Exit.
- When NightSim presents the warning dialog asking if you would like to save the current configuration, press No.

Exiting NightView

To exit NightView

- From the NightView Principal Debug Window, select

  Exit (Quit NightView)

  from the NightView menu.

Conclusion

This concludes the RedHawk NightStar Tools Tutorial.

For more in-depth tutorials, refer to the sections in the manuals listed below:

- *NightProbe User's Guide*
  - Appendix C - "Probing Programs Tutorial"
  - Appendix C - "Probing Devices Tutorial"
- *NightView User's Guide*
  - Chapter 4 - "Tutorial" (command-line interface)
  - Chapter 5 - "GUI Tutorial" (graphical user interface)

A
Tutorial Files

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

sim.c

```
#include <unistd.h>
#include <stdio.h>
#include <ntrace.h>
#include <string.h>
#include <errno.h>
#include <time.h>
#include <fbsched.h>
#include "rcim_timer.h"

#define cycle_start 110
#define cycle_end 111

class Counters ;

class Counters {
public:
   Counters (int i=0, int load=10000);
   void Increment(int i) { i_counter = (i_counter + i) % 10; }
   void SetWorkload (int load);
   int  Calculate (void);
   int  Get(void);
   void Work (void);
   float      cycle_time;
private:
   int        i_counter;
   int        workload;
};

Counters counters ;
rcim_timer timer;

static void trace_setup (char *);

main()
{
   int arg;

   counters.SetWorkload(0);

   trace_setup ("sim-data") ;

   while (fbswait() == 0) {
      timer.start();
      counters.Increment(1);
      trace_event_arg (cycle_start, counters.Get());
      counters.Work();
      timer.stop();
      arg = counters.Get() % 10;
      trace_event_arg (cycle_end, arg);
      counters.cycle_time = (float) timer.elapsed();
   }
}

Counters::Counters (int i, int load)
{
   i_counter = i;
   workload  = load;
}
```

```
void
Counters::Work (void)
{
   int i;
   volatile int x = 0;
   for (i=0; i<workload; ++i) {
      x = x * Calculate();
   }
   timer.spin (100);
}

int
Counters::Calculate (void)
{
   return i_counter*2;
}

int
Counters::Get(void)
{
   return i_counter;
}

void
Counters::SetWorkload (int load)
{
   workload = load;
}

static
void
trace_setup (char * key)
{
   struct pgm2_ds ds;
   ntconfig_t config;
   char thread_name[20];
   char dont_care[2048];
   int status;

   config.ntc_buffer_size      = 1024*16;
   config.ntc_use_spl          = 0;
   config.ntc_use_resched      = 0;
   config.ntc_lock_pages       = 0;
   config.ntc_clock            = 0;
   config.ntc_shmid_perm       = 0666;
   config.ntc_daemon_preferred = 1;

   trace_begin (key, &config);
   trace_open_thread ("sim");
}
```

rcim_timer.c

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/mman.h>
#include <fcntl.h>
#include <errno.h>
#include <ntrace.h>

#include "rcim_timer.h"

rcim_timer::rcim_timer (void)
{
   char      * rcim_tick_addr;
   int         rcim_tick_fd;

#define PAGE_SIZE 4096
#define PATH "/dev/rcim/sclk"

   clock = NULL ;

   rcim_tick_fd = open(PATH, O_RDONLY, 0);
   if (rcim_tick_fd == -1) {
      //printf ("failed to open RCIM \n") ;
      return ;
   }

   rcim_tick_addr = (caddr_t) mmap(
      NULL, (size_t) PAGE_SIZE, PROT_READ, MAP_SHARED, rcim_tick_fd, 0);
   if (rcim_tick_addr == (caddr_t) -1) {
      //printf ("failed to mmap RCIM \n") ;
      close(rcim_tick_fd);
      return;
   }
   close(rcim_tick_fd);

   clock = (timestamp_t *) rcim_tick_addr;
}

void
rcim_timer::start (timestamp_t * stamp)
{
   if (!stamp) {
      stamp = &start_time;
   }
   for (;;) {
      stamp->high = clock->high ;
      stamp->low = clock->low ;
      if (clock->high == stamp->high) {
         return ;
      }
   }
}

void
rcim_timer::stop (timestamp_t * stamp)
{
   if (!stamp) {
      stamp = &stop_time;
   }
   for (;;) {
```

```
      stamp->high = clock->high ;
      stamp->low = clock->low ;
      if (clock->high == stamp->high) {
         return ;
      }
   }
}

#define SECONDS_PER_TICK 0.000000400 // 400 ns

double
rcim_timer::elapsed (timestamp_t * start, timestamp_t * stop)
{
   if (!start && !stop) {
      start = &start_time;
      stop  = &stop_time;
   }
   int upper = stop->high - start->high;
   int lower = stop->low  - start->low;

   return double(upper) * SECONDS_PER_TICK * 4294967296.0 +
          double(lower) * SECONDS_PER_TICK ;
}

rcim_timer::~rcim_timer (void)
{
   (void) munmap ((caddr_t)clock,PAGE_SIZE);
}

void
rcim_timer::spin (int micro_seconds)
{
   timestamp_t start, stop;

   rcim_timer::start (&start);
   for(;;) {
      rcim_timer::stop(&stop);
      if (elapsed(&start,&stop) >= (double)micro_seconds/1000000.0) {
         return;
      }
   }
}
```

rcim_timer.h

```
class rcim_timer {
   typedef struct {
      int high ;
      int fill ;
      int low ;
   } timestamp_t ;
public:
   rcim_timer (void);
   ~rcim_timer (void);
   void start (timestamp_t * stamp = NULL);
   void stop  (timestamp_t * stamp = NULL);
   double elapsed (timestamp_t * start = NULL,
                   timestamp_t * stop  = NULL);
   void spin (int micro_seconds);
private:
   volatile timestamp_t * clock ;
   timestamp_t start_time ;
   timestamp_t stop_time ;
} ;
```