

# PowerWorks Linux Development Environment Tutorial

---



0898100-030  
December 2004

Copyright 2004 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.

PowerWorks, PowerMAX OS, Power Hawk, NightBench, NightProbe, NightSim, NightTrace, NightView, and MAXAda are trademarks of Concurrent Computer Corporation.

Motorola is a registered trademark of Motorola, Inc.

Linux is a registered trademark of Linus Torvalds.

UNIX is a registered trademark of The Open Group.

Printed in U. S. A.

## General Information

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

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

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

## Scope of Manual

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

## Structure of Manual

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

## Syntax Notation

The following notation is used throughout this guide:

<i>italic</i>	Books, reference cards, and items that the user must specify appear in <i>italic</i> type. Special terms and comments in code may also appear in <i>italic</i> .
<b>list bold</b>	User input appears in <b>list bold</b> type and must be entered exactly as shown. Names of directories, files, commands, options and man page references also appear in <b>list bold</b> type.
list	Operating system and program output such as prompts and messages and listings of files and programs appears in list type. Keywords also appear in list type.

<u>emphasis</u>	Words or phrases that require extra emphasis use <u>emphasis</u> type.
window	Keyboard sequences and window features such as push buttons, radio buttons, menu items, labels, and titles appear in <b>window</b> 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 <i>is defined as</i> in Backus-Naur Form (BNF).

## Referenced Publications

The following publications are referenced in this document:

0890395	<i>NightView User's Guide</i>
0890398	<i>NightTrace Manual</i>
0890465	<i>NightProbe User's Guide</i>
0890458	<i>NightSim User's Guide</i>
0890514	<i>NightBench User's Guide</i>
0890516	<i>MAXAda Reference Manual</i>

# Contents

## Chapter 1 Using the PLDE

Overview .....	1-1
Before you begin .....	1-1
Remote shell access .....	1-2
Privileges .....	1-3
Additions to PATH .....	1-4
Getting Started .....	1-5
Creating a working directory .....	1-5
Exporting the filesystem .....	1-6
Mounting the filesystem .....	1-6
Copying tutorial-related files from the installation CD .....	1-7
Using NightBench .....	1-9
Creating a new environment .....	1-9
Introducing existing source files into the environment .....	1-12
Setting compile options .....	1-15
Adding an environment to the Environment Search Path .....	1-16
Defining a partition .....	1-17
Building a partition .....	1-17
Showing a reference .....	1-19
Correcting an error .....	1-20
Rebuilding a partition .....	1-21
Before you continue .....	1-22
Using NightSim .....	1-24
Invoking NightSim .....	1-24
Configuring the Scheduler .....	1-24
Scheduling a process .....	1-26
Setting up the scheduler .....	1-28
Using NightView .....	1-29
Setting a monitorpoint .....	1-31
Resuming execution .....	1-33
Starting the simulation .....	1-34
Monitoring the simulation .....	1-35
Using NightProbe .....	1-37
Invoking NightProbe .....	1-37
Configuring NightProbe .....	1-38
Selecting the target system .....	1-41
Connecting to the target program .....	1-42
Starting sampling .....	1-42
Modifying program data .....	1-43
Using NightTrace .....	1-46
Invoking NightTrace .....	1-46
Invoking NightTrace from the command line .....	1-46
Loading data and configuration files .....	1-47
Invoking NightTrace from NightProbe .....	1-51
Configuring a user daemon .....	1-52

Creating a customized display page . . . . .	1-54
Creating the user application daemon . . . . .	1-55
Resuming execution of the user application daemon . . . . .	1-56
Displaying the user trace data . . . . .	1-56
Inserting a patchpoint . . . . .	1-57
Viewing streaming trace output . . . . .	1-59
Configuring a kernel daemon . . . . .	1-60
Creating the kernel daemon . . . . .	1-61
Resuming execution of the kernel daemon . . . . .	1-63
Displaying the kernel trace data . . . . .	1-63
Flushing the trace data . . . . .	1-64
Stopping the daemons . . . . .	1-64
Positioning the current time line . . . . .	1-64
Loading an eventmap file . . . . .	1-66
Searching for a user trace event . . . . .	1-66
Zooming in . . . . .	1-68
Examining the kernel trace data . . . . .	1-69
Exiting the tools . . . . .	1-71
Exiting NightTrace . . . . .	1-72
Exiting NightProbe . . . . .	1-72
Exiting NightSim . . . . .	1-72
Exiting NightView . . . . .	1-73
Exiting NightBench . . . . .	1-73
Conclusion . . . . .	1-73

## Appendix A Tutorial Files

sim.ada . . . . .	A-2
counters.ada . . . . .	A-3
art.conversions.ada . . . . .	A-5

## Illustrations

Figure 1-1. NightBench Project . . . . .	1-9
Figure 1-2. Creating a new environment - language selection . . . . .	1-10
Figure 1-3. Creating a new environment - specifications . . . . .	1-10
Figure 1-4. Introducing existing source files into the environment . . . . .	1-12
Figure 1-5. Source files introduced into the environment . . . . .	1-13
Figure 1-6. Units introduced into the environment . . . . .	1-14
Figure 1-7. Setting environment-wide compile options . . . . .	1-15
Figure 1-8. Adding an environment to the Environment Search Path . . . . .	1-16
Figure 1-9. Builder window - Build page with errors for <b>sim</b> partition . . . . .	1-18
Figure 1-10. HyperHelp viewer displaying a reference . . . . .	1-20
Figure 1-11. Editing the source file <b>sim.ada</b> in NEdit . . . . .	1-21
Figure 1-12. Build completed with no errors . . . . .	1-22
Figure 1-13. NightSim Scheduler . . . . .	1-24
Figure 1-14. NightSim Edit Process . . . . .	1-26
Figure 1-15. Start NightView Session on Remote Host dialog . . . . .	1-29
Figure 1-16. NightView Dialogue . . . . .	1-30
Figure 1-17. NightView Principal Debug Window . . . . .	1-31
Figure 1-18. Setting a new monitorpoint . . . . .	1-32
Figure 1-19. NightView Monitor Window . . . . .	1-33
Figure 1-20. Resuming execution . . . . .	1-34

Figure 1-21. Starting the simulation	1-34
Figure 1-22. Monitoring <code>cycle_time</code> in the NightView Monitor Window	1-35
Figure 1-23. NightSim Monitor	1-36
Figure 1-24. NightProbe Main window	1-37
Figure 1-25. NightProbe Timer menu	1-38
Figure 1-26. NightProbe Output menu	1-38
Figure 1-27. Configured NightProbe Data Recording window	1-39
Figure 1-28. NightProbe Spreadsheet Viewer window	1-40
Figure 1-29. Target System Selection dialog	1-41
Figure 1-30. User Authentication dialog	1-42
Figure 1-31. Modified values in NightView Monitor Window	1-43
Figure 1-32. Modified values in NightProbe Spreadsheet Viewer	1-44
Figure 1-33. NightTrace Main window	1-46
Figure 1-34. NightTrace user trace display page	1-47
Figure 1-35. NightTrace user trace display page with data loaded	1-48
Figure 1-36. NightTrace kernel display page	1-49
Figure 1-37. NightTrace Main Window	1-50
Figure 1-38. Daemon Definition dialog	1-51
Figure 1-39. Login dialog	1-52
Figure 1-40. Import Daemon Definition dialog	1-52
Figure 1-41. Customized NightTrace display page	1-54
Figure 1-42. User trace data in customized NightTrace display page	1-56
Figure 1-43. Setting a new patchpoint	1-57
Figure 1-44. User trace data after patchpoint inserted	1-59
Figure 1-45. Daemon Definition dialog	1-60
Figure 1-46. NightTrace kernel display page	1-61
Figure 1-47. NightTrace kernel trace data	1-64
Figure 1-48. NightTrace Search dialog	1-66
Figure 1-49. User trace data after search	1-67
Figure 1-50. Zoomed in view of user trace data	1-68
Figure 1-51. Zoomed in view of kernel display page	1-69
Figure 1-52. Removing the scheduler	1-71
Figure 1-53. Remove Scheduler dialog	1-72





# Using the PLDE

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

The PowerWorks Linux Development Environment consists of high-performance C/C++ and MAXAda™ (Ada95) compilers, the NightView™ symbolic debugger, NightTrace™ event analyzer, NightSim™ frequency-based scheduler, the NightProbe™ data monitoring tool, the NightBench™ Program Development Environment, and the **shmdefine** shared memory helper tool.

## Overview

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

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

integrating them together into one cohesive example.

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

## Before you begin

In order to run the portion of the tutorial that uses the NightSim Scheduler and the NightView Source-Level Debugger, a system running PowerMAX OS should be networked to

your Linux system. If you have a PowerMAX OS system networked to your Linux system, the following items must also be taken into consideration:

- Remote shell access
- Privileges
- Additions to PATH

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

#### NOTE

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

## Remote shell access

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

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

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

An example entry might look like:

```
remote_machine_name username
```

where *remote\_machine\_name* is the name of the NightSim host and *username* is your login name on that system.

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

**or**

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

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

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

where *remote\_machine\_name* is the name of the target system. You should see the date and time on the remote system if successful.

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

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

## Privileges

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

- P\_CPUBIAS
- P\_PLOCK
- P\_RUNTIME

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

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

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

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

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

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

```
exec /sbin/tfadmin PLDE_USERS: shell
```

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

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

## **Additions to PATH**

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

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

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

## Getting Started

It is *highly recommended* that the paths to the executables and working directories on the host system (the system running Linux) and the target system (the system running PowerMAX OS) are identical. Their mount points should be based on a common name.

The following sections will guide us through creating a working directory on our host system, exporting that filesystem to our target system, and mounting the filesystem on that target system. We will also copy the tutorial-related files from the PowerWorks Linux Development Environment Installation CD to our working directory.

### NOTE

The following steps will likely require root access

## Creating a working directory

We will start by creating a directory on the Linux system in which we will do all our work.

### NOTE

The following steps will likely require root access

### To create a working directory on the Linux system

- On the Linux system, use the **mkdir (1)** command to create a working directory.

### NOTE

The pathname should be unique such that it can be mounted with the same pathname on the PowerMAX OS system.

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

```
mkdir /tutorial
```

- Ensure that the directory is writable by issuing the following command:

```
chmod 777 /tutorial
```

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

```
cd /tutorial
```

#### NOTE

Ensure that your **umask** setting on the Linux system will allow the PowerMAX OS system to read and write files in your working directory, or use the same user and group ID on both systems.

To automatically ensure that all files your user creates on the Linux system are publicly readable and writeable, include the following command in your shell startup script:

```
umask 000
```

## Exporting the filesystem

In order for the PowerMAX OS system to be able to access the Linux directory that we created in “Creating a working directory” on page 1-5, we must export the filesystem.

#### NOTE

The following steps will likely require root access

### To export the Linux filesystem to the PowerMAX OS system

- Enter the following command on the Linux system:

```
exportfs -o rw pmax_system:work_dir
```

where *pmax\_system* is the name of the PowerMAX OS system and *work\_dir* is the pathname of our working directory.

For our example, we will enter the following command:

```
exportfs -o rw yoshi:/tutorial
```

where **yoshi** is the name of our PowerMAX OS system and **/tutorial** is our working directory.

## Mounting the filesystem

In order for the PowerMAX OS system to be able to access the Linux directory that we exported in “Exporting the filesystem” on page 1-6, we must mount that filesystem on the PowerMAX OS system.

#### NOTE

The following steps will likely require root access

## To mount the filesystem on the PowerMAX OS system

- On the PowerMAX OS system, use the **mkdir (1)** command to create a place in which to mount the Linux filesystem.

### NOTE

The pathname should be the same as that created on the Linux system (see “Creating a working directory” on page 1-5).

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

```
mkdir /tutorial
```

- Mount the filesystem using the following command:

```
mount -F nfs linux_system:work_dir work_dir
```

where *linux\_system* is the name of the Linux system and *work\_dir* is the pathname of our working directory.

For our example, we will enter the following command:

```
mount -F nfs raptor:/tutorial /tutorial
```

where **raptor** is the name of our Linux system and **/tutorial** is our working directory.

## Copying tutorial-related files from the installation CD

Source files, as well as configuration files for the various tools, are included on the PowerWorks Linux Development Environment Installation CD. We will copy these tutorial-related files to our **tutorial** directory.

### NOTE

The following steps will likely require root access

### To copy the files from the PowerWorks Linux Development Environment Installation CD

- Insert the PowerWorks Linux Development Environment Installation CD in the CD-ROM drive on the Linux system.
- Mount the CD-ROM drive on the Linux system (assuming the standard mount entry for the CD-ROM device exists in **/etc/fstab**).

```
mount /mnt/cdrom
```

- Copy all tutorial-related configuration files to our working directory.

```
cp /mnt/cdrom/tutorial-sup/* work_dir
```

where *work\_dir* is the pathname of our working directory.

For our example, we will enter the following command:

```
cp /mnt/cdrom/tutorial-sup/* /tutorial
```

where **/tutorial** is our working directory.

- Unmount the CD-ROM drive (otherwise, you will be unable to remove the PowerWorks Linux Development Environment Installation CD from the CD-ROM drive).

```
umount /mnt/cdrom
```



## Using NightBench

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

Let's open the NightBench Project window.

### To invoke NightBench from the command line

- From the command line, type the following command:

```
nbench
```

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

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

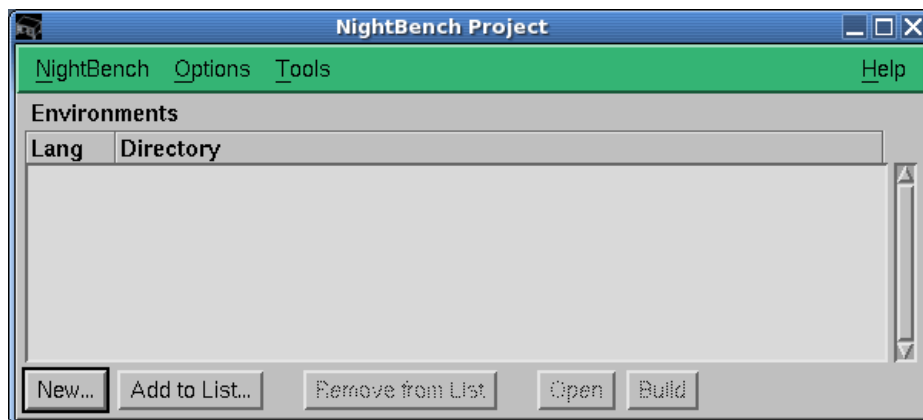


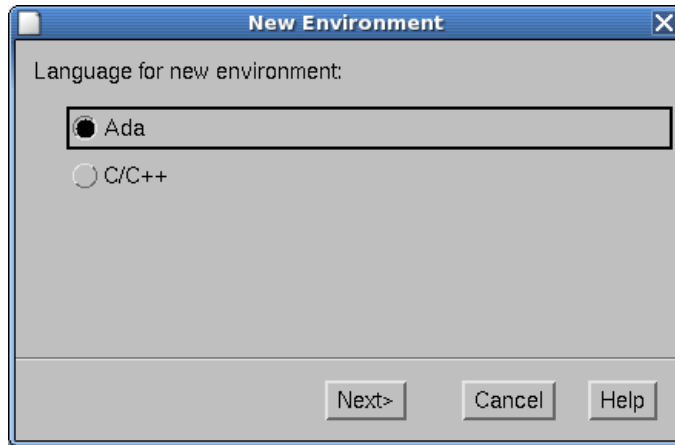
Figure 1-1. NightBench Project

## Creating a new environment

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

**To create a new environment from the NightBench Project window**

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



**Figure 1-2. Creating a new environment - language selection**

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



**Figure 1-3. Creating a new environment - specifications**

- Type the directory name in the **Directory for new environment** field where you want NightBench to create the new environment. This can be the name of an existing directory or NightBench can create the directory

for you. (Note that NightBench can only create a subdirectory of an existing directory.) We will enter the name of the directory we created in “Getting Started” on page 1-5. The full directory name in our example is **/tutorial**. Since we invoked NightBench from that directory, the path-name will appear in the **Directory for new environment** field.

- Select a **Compiler Release** if you have more than one release of MAX-Ada installed on your system. If you have only one release of MAXAda installed on your system, it will appear here.
- Choose a **Target Architecture**.

Because we are building an executable that will run on a Concurrent real-time computer system, we must choose which type of system we are targeting.

To determine which item to select from the drop-down list, issue the following command on the PowerMAX OS target system:

```
uname -m
```

Make your selection based on the machine hardware name returned from the **uname** command:

<b>Machine Hardware Name</b>	<b>Target Architecture</b>
Motorola	<b>moto</b>
Synergy	<b>synergy</b>
all others	<b>nh</b>

For our example, we will be targeting a Power Hawk™ 640 so we will select **moto** from the drop-down list. For more information on target architectures, see the section titled “Target Architectures” in the *MAXAda Reference Manual* (0890516).

- Identify the **PowerMAX OS Version**.

To determine the version of PowerMAX OS running on the target system, issue the following command on that target system:

```
uname -r
```

In our example, we will select **5.1** from the drop-down list for the version of the operating system running on the system we are targeting.

- Press **Done**

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

## Introducing existing source files into the environment

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

Source files may already have been created outside the NightBench environment or you may use the editing features of NightBench to create a new file. For this example, we will introduce the files we copied from the PowerWorks Linux Development Environment Installation CD (see “Getting Started” on page 1-5).

### NOTE

Listings of the source files can be found in Appendix A “Tutorial Files”.

### To introduce existing source files into a NightBench environment

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

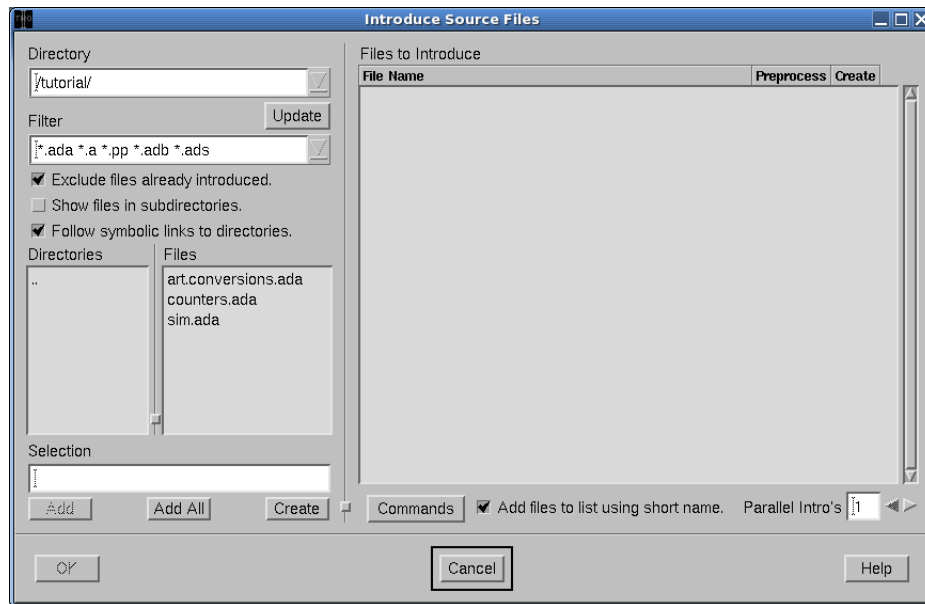


Figure 1-4. Introducing existing source files into the environment

- Maneuver to the directory in which the source files are contained. You may type the path to the directory name in the Directory field or use the entries in the Directories list to navigate to the desired directory.

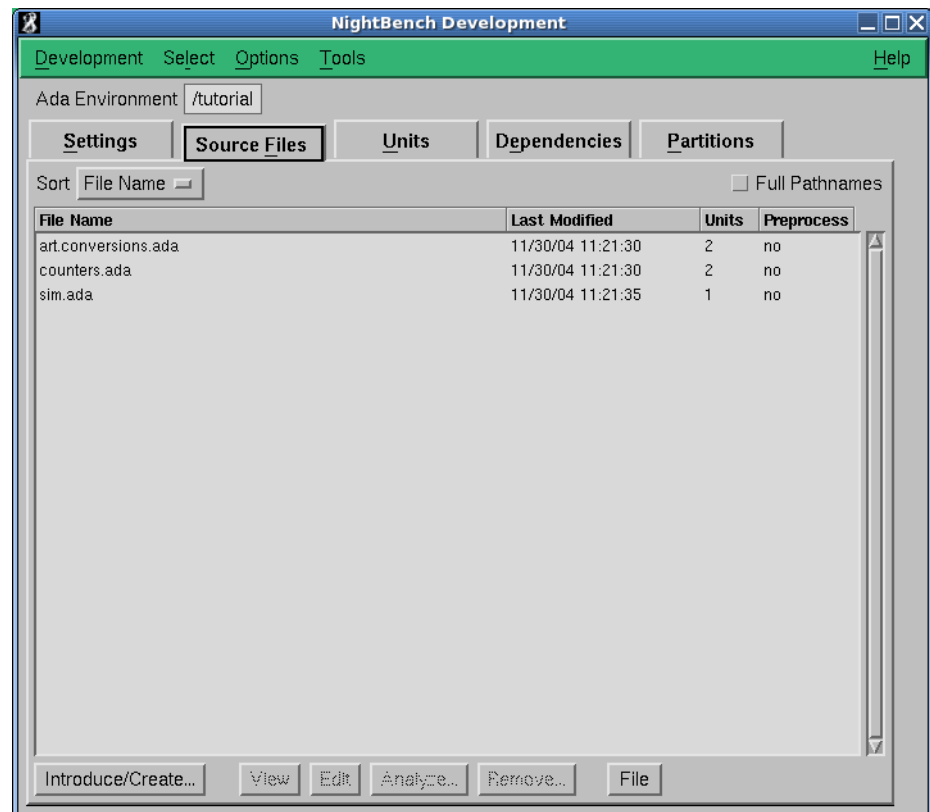
**NOTE**

Since we invoked NightBench from the directory in which our file resides, we should already be positioned in that directory.

- Press the **Add All** button located beneath the **Selection** field. The names of the three source files will then appear in the list of **Files to Introduce**.
- Press the **OK** button to introduce the source files into the environment.

The source files now appear in the list of files on the **Source Files** page of the NightBench Development window and the units contained within them now appear on the **Units** page.

Figure 1-5 shows the **Source Files** page of the NightBench Development window. You can see the three source files we have just introduced.



**Figure 1-5. Source files introduced into the environment**

Figure 1-6 shows the **Units** page of the NightBench Development window. This figure shows the units contained in the source files that were just introduced into the environment: the body of `sim` (contained in the source file `sim.ada`), the specification and body of `counters` (contained in the source file `counters.ada`), and the specification and

body of `ada.real_time.conversions` (contained in the source file `art.conversions.adb`). Note that the state of these units is uncompiled.

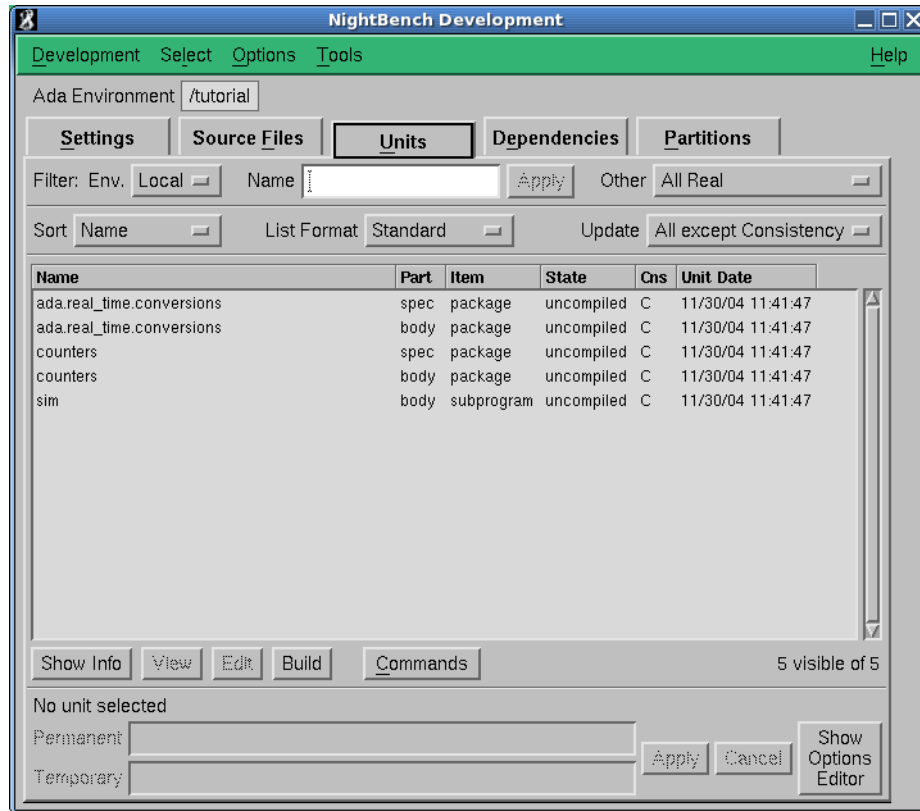


Figure 1-6. Units introduced into the environment

## Setting compile options

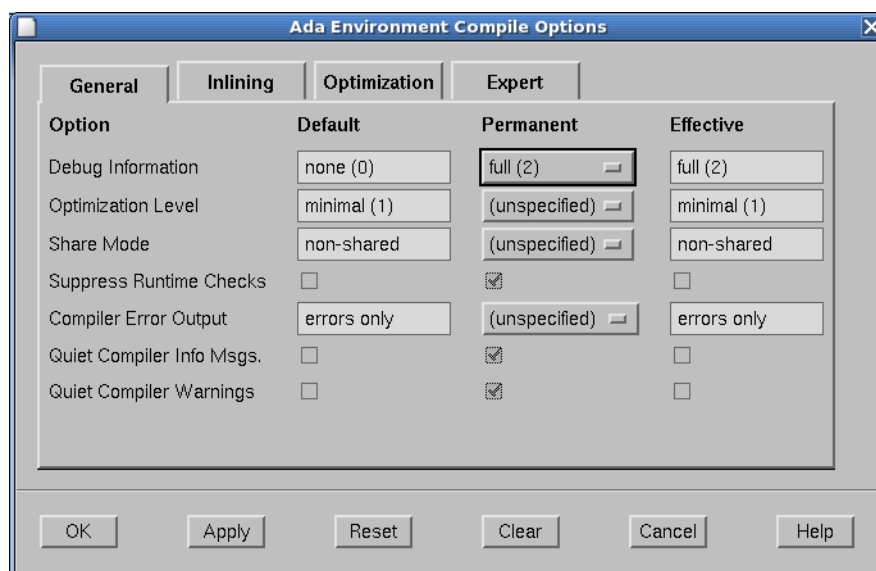
In order to debug the program using the NightView Source Level Debugger, we need to compile the program with debug information. We do this by setting an environment-wide compile option which will apply to all units within the current environment.

### NOTE

NightProbe also requires that the user application is built with debugging information in order to read symbol table information from user application program files. This enables NightProbe to determine which variables may be probed.

### To set environment-wide compile options

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



**Figure 1-7. Setting environment-wide compile options**

**NOTE**

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

## Adding an environment to the Environment Search Path

MAXAda supplies a number of pre-built environments containing various packages that can be used for program development. These packages consist of pre-compiled Ada source code with units grouped together on the basis of Ada language definitions and functionality.

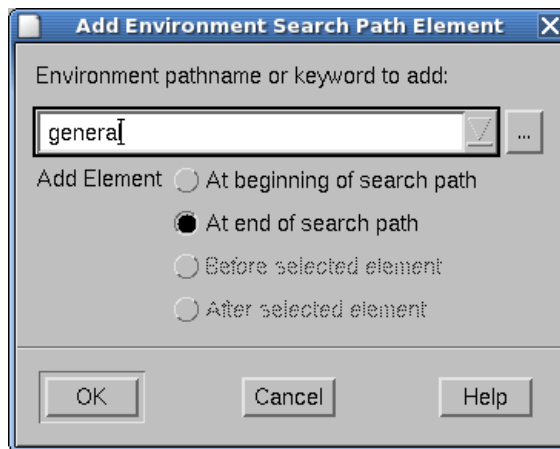
By default, when you create a new environment, you automatically have access to the MAXAda environment **predefined** which contains the standard Ada-defined packages.

Our program utilizes the MAXAda-defined package **night\_trace\_bindings** which is included in the **general** environment. In order for NightBench to use that package, we must add that environment to our Environment Search Path.

### To add an environment to the Environment Search Path

- Click on the **Settings** tab of the NightBench Development window.
- Press the **Add...** button at the bottom of the **Environment Search Path** area.

The **Add Environment Search Path Element** dialog is presented.



**Figure 1-8. Adding an environment to the Environment Search Path**

- Enter **general** in the **Environment pathname or keyword to add:** field.
- Ensure that the **At end of search path** radiobutton is selected.



- Press OK.

You will see the environment `/usr/ada/release/bindings/general` added to the Environment Search Path (where *release* is the name of the Ada release).

## Defining a partition

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

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

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

### To define all active partitions in the environment

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

At this point, we have an environment, `/tutorial`, that has within it the definition for the active partition, `sim`, made up of a main unit, `sim`, contained in the source file, `sim.ada`, and two other units - `counters`, contained in the source file `counters.ada`, and `ada.real_time.conversions` contained in the source file `art.conversions.ada`. Full debug information will be generated for the program and the environment containing the `night_trace_bindings` package has been added to our Environment Search Path.

## Building a partition

We are now ready to build our partition. We do this using the NightBench Builder.

### To build the partition

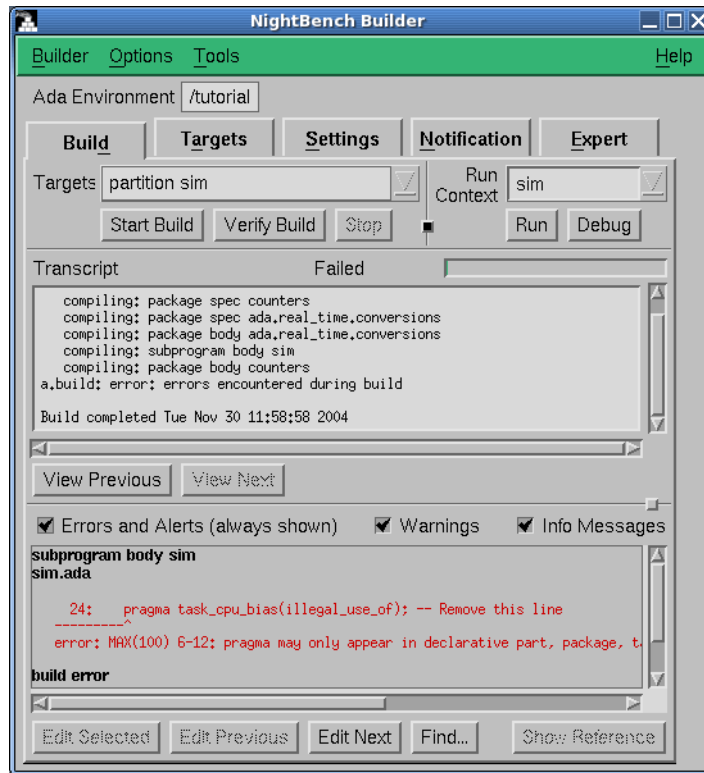
- Click on the **Partitions** tab of the NightBench Development window to get to the **Partitions** page.
- In the list of partitions on the **Partitions** page, make sure the partition `sim` is selected.

- Press the button marked **Build**. This will open the NightBench Builder window so we can build our new partition.

In Figure 1-9, you will see that **partition sim** has been automatically entered in the **Targets** field on the **Build** page. This is because it was selected on the **Partitions** page when the **Build** button was pressed.

- Press **Start Build**.

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



**Figure 1-9. Builder window - Build page with errors for sim partition**

When the build finishes, a **Build Completed** dialog notifies the user.

#### **NOTE**

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

In our example, the **Build Completed** dialog notifies us that the build completed with errors. This is due to the following line which we have included in **sim.ada** (see "sim.ada" on page A-2):

```
pragma task_cpu_bias(illegal_use_of); -- Remove this line
```

This was done to demonstrate the error handling capabilities of NightBench.

- Press OK to dismiss the Build Completed dialog.

## Showing a reference

NightBench allows Ada users to display the section of the *MAXAda Reference Manual* or the *Ada 95 Reference Manual* related to the error selected in the NightBench Builder.

Since our build completed with errors, we can use this feature to display the references associated with those errors.

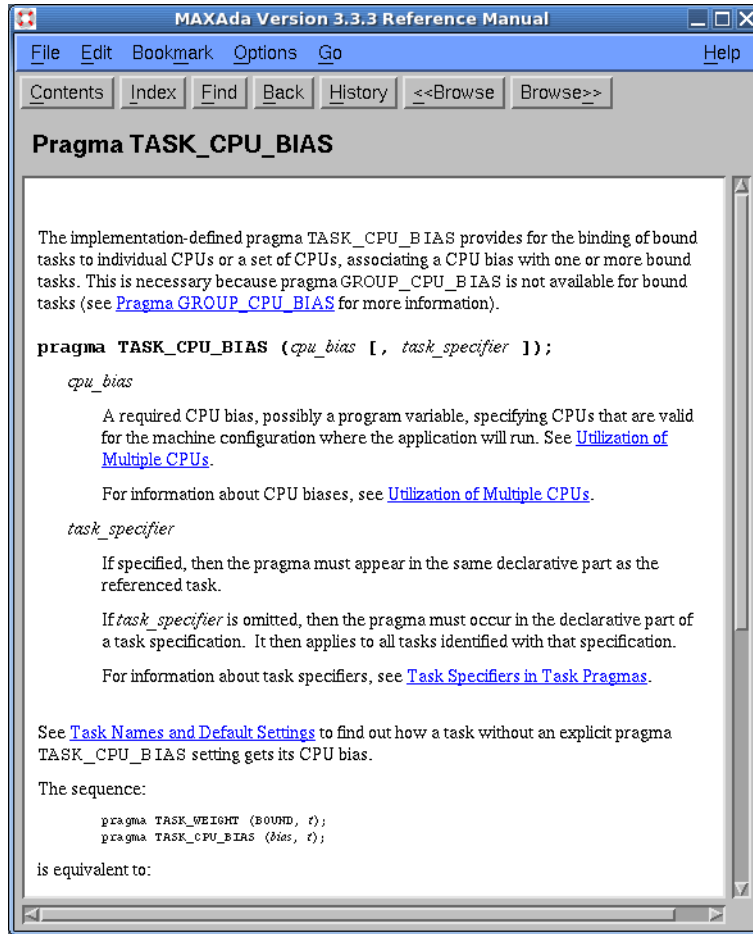
### Note

When executing the steps described below, if you are using the HyperHelp tool to display this tutorial, the page displayed will automatically switch to the *MAXAda Reference Manual* when you click the **Show Reference** button. To return to this tutorial, click the **Back** button in the HyperHelp tool

### To show a reference

- Click on the **Build** tab of the NightBench Builder window to get to the **Build** page.
- Select the error from the **Errors and Alerts** window located at the bottom of the **Build** page by clicking on the text of the error.
- Press the **Show Reference** button.

This opens the HyperHelp viewer to the section of the *MAXAda Reference Manual* related to the selected error.



**Figure 1-10. HyperHelp viewer displaying a reference**

The *MAXAda Reference Manual* is opened in the HyperHelp viewer showing the topic associated with pragma `TASK_CPU_BIAS` since the error we have inserted in `sim.ada` (see “sim.ada” on page A-2) is associated with that pragma.

## Correcting an error

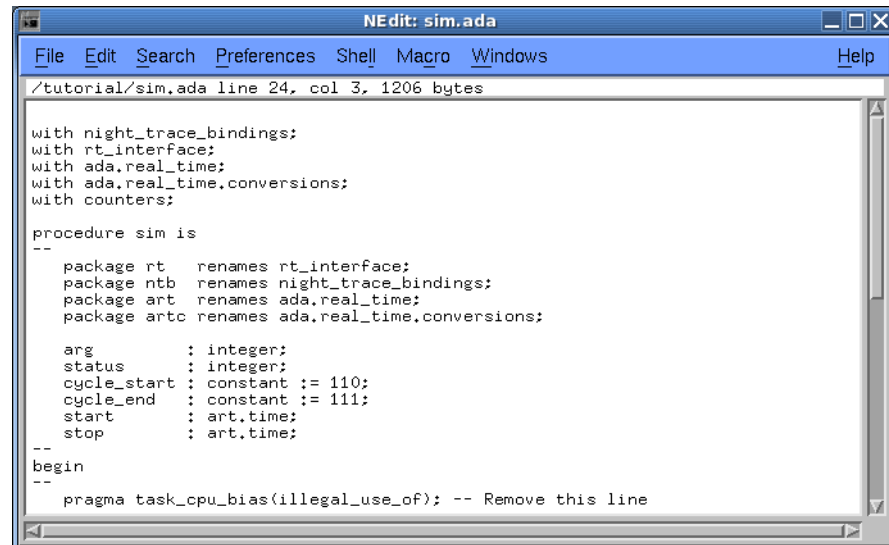
We must correct the error in order for our build to complete successfully. We can do that using the editing features of NightBench.

### To correct an error

- Click on the Build tab of the NightBench Builder window to get to the Build page.
- Select the error from the Errors and Alerts window located at the bottom of the Build page by clicking on the text of the error.
- Press the Edit Selected button.

The source file associated with the selected error is opened in the editor configured for NightBench (see the section titled “Preferences - Editor” in the *NightBench User’s Guide* (0890514)).

In our case, the source file **sim.ada** (see “sim.ada” on page A-2) will be brought up in the default editor, NEdit. NEdit is a part of the PowerWorks Linux Development Environment.



```

NEdit: sim.ada
File Edit Search Preferences Shell Macro Windows Help
/tutorial/sim.ada line 24, col 3, 1206 bytes
with night_trace_bindings;
with rt_interface;
with ada.real_time;
with ada.real_time.conversions;
with counters;

procedure sim is
--
  package rt renames rt_interface;
  package ntb renames night_trace_bindings;
  package art renames ada.real_time;
  package artc renames ada.real_time.conversions;

  arg      : integer;
  status   : integer;
  cycle_start : constant := 110;
  cycle_end : constant := 111;
  start    : art.time;
  stop     : art.time;
--
begin
--
  pragma task_cpu_bias(illegal_use_of); -- Remove this line

```

**Figure 1-11. Editing the source file `sim.ada` in NEdit**

The problem with the offending line is that it does not belong in our program at all. It was only included to demonstrate the error handling capabilities of NightBench. We need to delete the line, save the file, and rebuild our program.

- Select the line

```
pragma task_cpu_bias(illegal_use_of); -- Remove this line
```

in the editor configured for NightBench. (In NEdit, you may triple-click on a line to select it.)

- Press the **Backspace** key to delete the line.
- Select **Save** from the **File** menu to save the changes.
- Select **Exit** from the **File** menu to close NEdit.

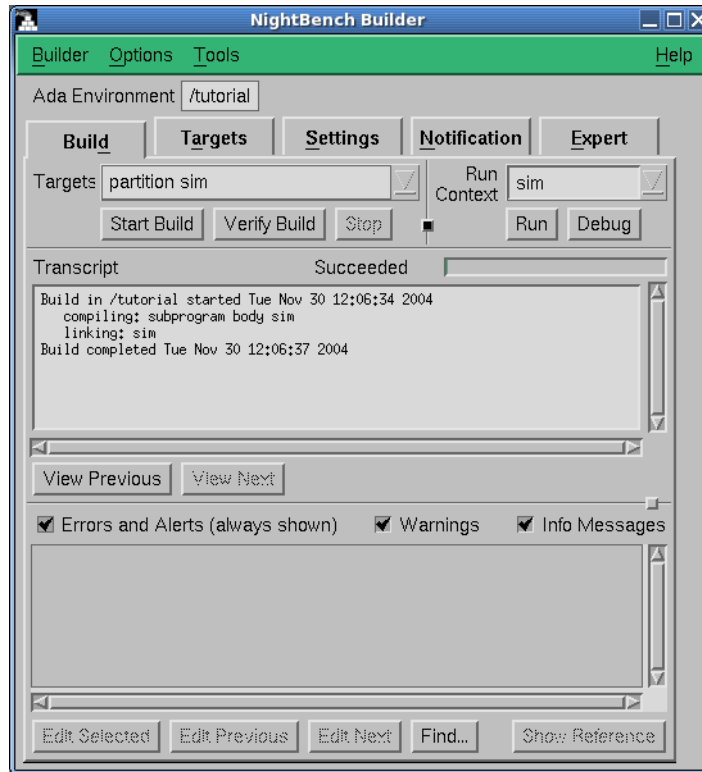
## Rebuilding a partition

Now that we have corrected the error, we are ready to rebuild our partition.

### To rebuild a partition

- Click on the Build tab of the NightBench Builder window to get to the Build page.
- Press Start Build.

Since we only modified the body of the unit `sim` to correct our error and since none of the other units in our partition have any dependencies on `sim`, NightBench will recompile only the body of that unit and relink the partition.



**Figure 1-12. Build completed with no errors**

When the build finishes, a Build Completed dialog notifies us that the build has completed with no errors.

- Press OK to dismiss the Build Completed dialog.

### Before you continue

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

However, if you do not have a PowerMAX OS system networked to your Linux system, you may jump to the section “Using NightTrace” on page 1-46 and continue with the tutorial.

## Using NightSim

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

## Invoking NightSim

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

NightBench allows the user to invoke the NightSim Scheduler directly.

### To invoke NightSim from NightBench

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

## Configuring the Scheduler

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



Figure 1-13. NightSim Scheduler



### To configure a NightSim Scheduler

- Specify a **Scheduler key**. The key is a user-chosen numeric identifier with which the scheduler will be associated. For our example, we will use 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.
- Enter the name of a PowerMAX OS system which will act as the **Timing host** for the simulation. You may use the drop down list associated with this field for the names of systems previously used as timing hosts. For our example, we will enter **yoshi** a Power Hawk 640 system.

#### NOTE

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

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

#### NOTE

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

- Specify **Clock period**.

For our simulation, we would like the real-time clock to “fire” every .001 seconds (or 100000 microseconds).

For our example, we will specify 100000 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.

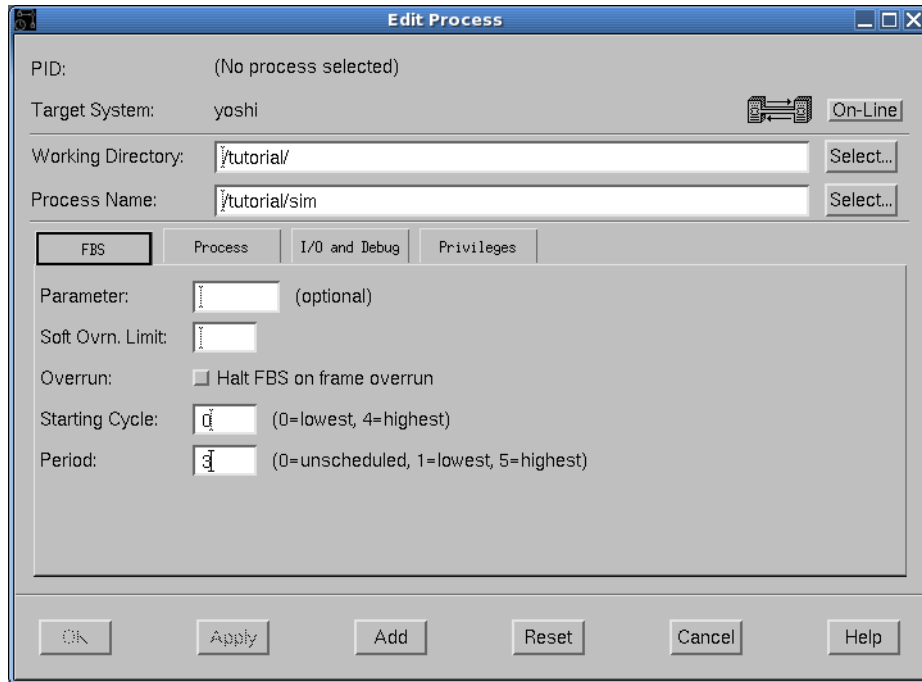


Figure 1-14. NightSim Edit Process

### 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.

### NOTE

The Select a Program dialog operates from the target PowerMAX OS system. If the working directory is not exported to and NFS mounted on the PowerMAX OS system, then the dialog will not be able to show the files. See “Getting Started” on page 1-5 for details on setting up your working directory.

- 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.

- 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.

## **Setting up the scheduler**

### **To set up the scheduler**

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

This action:

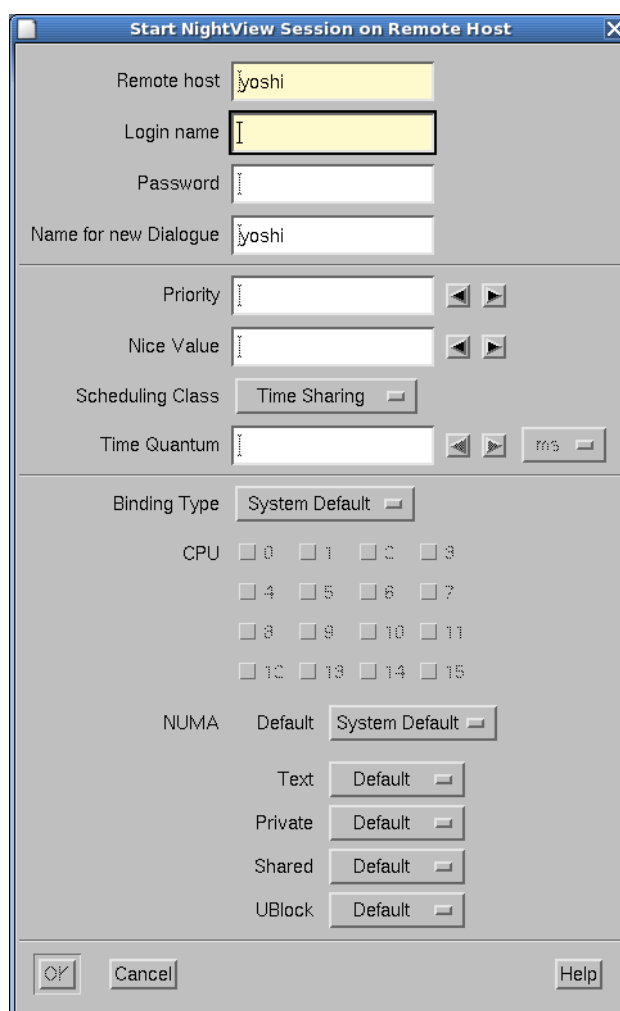
- creates a scheduler that is configured according to the parameters we specified
- schedules the processes that we have added to the NightSim Scheduler window and starts them running up to the first `fb_wait` call, and
- attaches the timing source to the scheduler.

Because we have specified the Schedule program within a NightView dialogue option when we added this process to the frequency-based scheduler (see “To add a process to the frequency-based scheduler” on page 1-26), the NightView Source Level Debugger will be started.

## Using NightView

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

Because we have specified the Schedule program within a NightView dialogue option when we added this process to the frequency-based scheduler (see “To add a process to the frequency-based scheduler” on page 1-26), we are presented with a NightView dialog allowing us to log into the target system on which we will be running our program.



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

### To start a NightView session on a remote host

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

#### NOTE

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

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

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

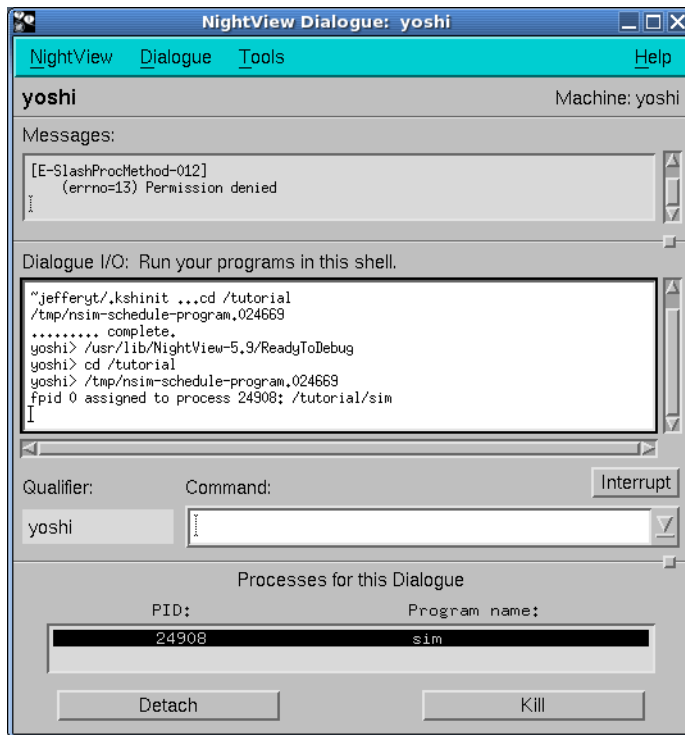


Figure 1-16. NightView Dialogue

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

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

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

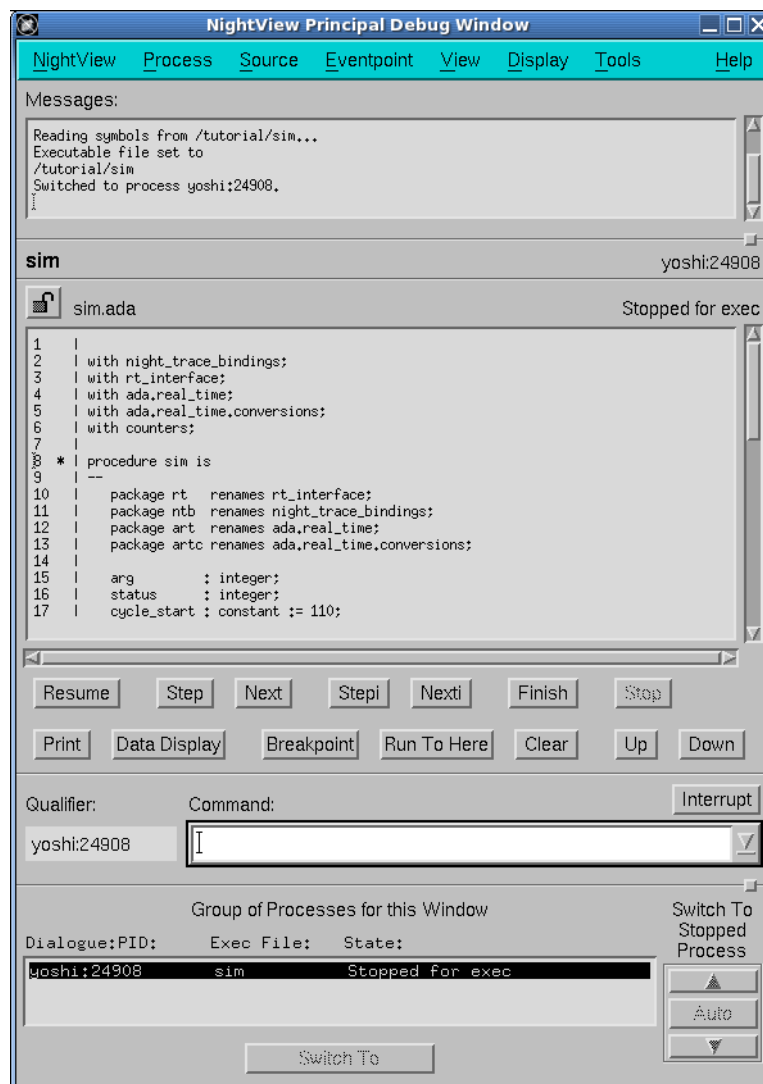


Figure 1-17. NightView Principal Debug Window

## 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 loca-

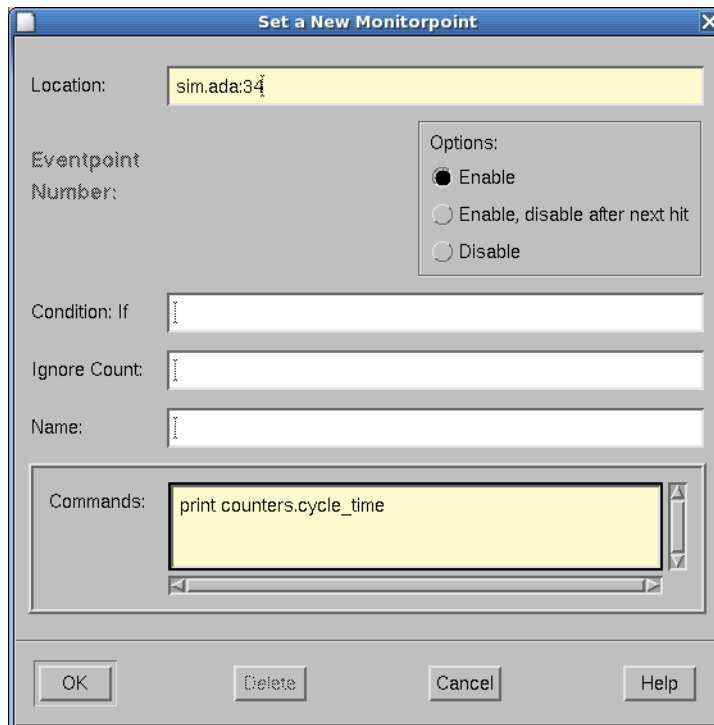
tion that will save the value of one or more expressions, which you specify. The saved values are then periodically displayed by NightView in a Monitor Window.

### To set a monitorpoint

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

```
rt.fbs_wait(status);
```

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



**Figure 1-18. Setting a new monitorpoint**

- Enter the following

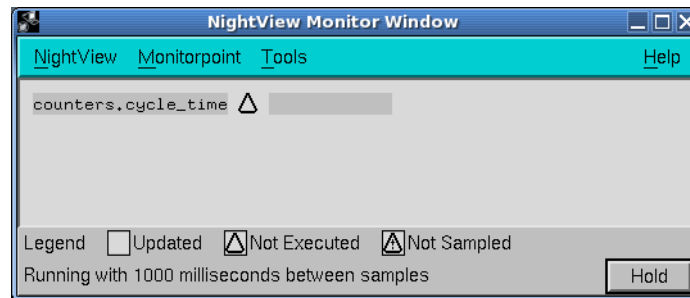
```
print counters.cycle_time
```

in the **Commands** text box.

- Press **OK**.

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





**Figure 1-19. 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:

```
rt.fbs_wait(status);
```

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.

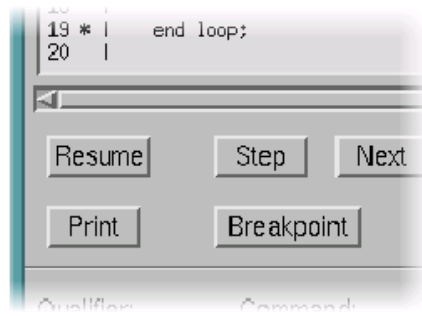


Figure 1-20. Resuming execution

## 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.

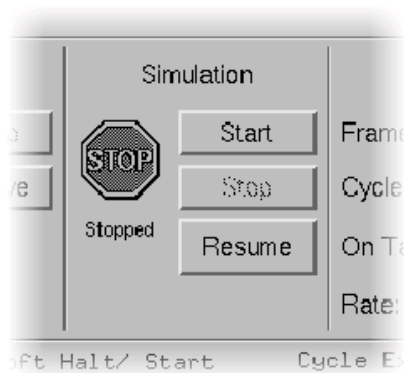
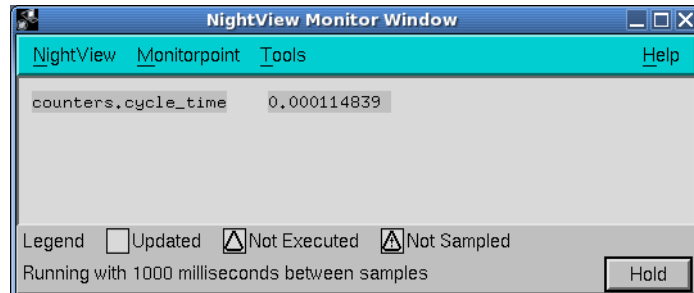


Figure 1-21. Starting the simulation

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

Also, note that the value for `counters.cycle_time` in the NightView Monitor Window changes as the program runs.



**Figure 1-22. Monitoring `cycle_time` in the NightView Monitor Window**

## Monitoring the simulation

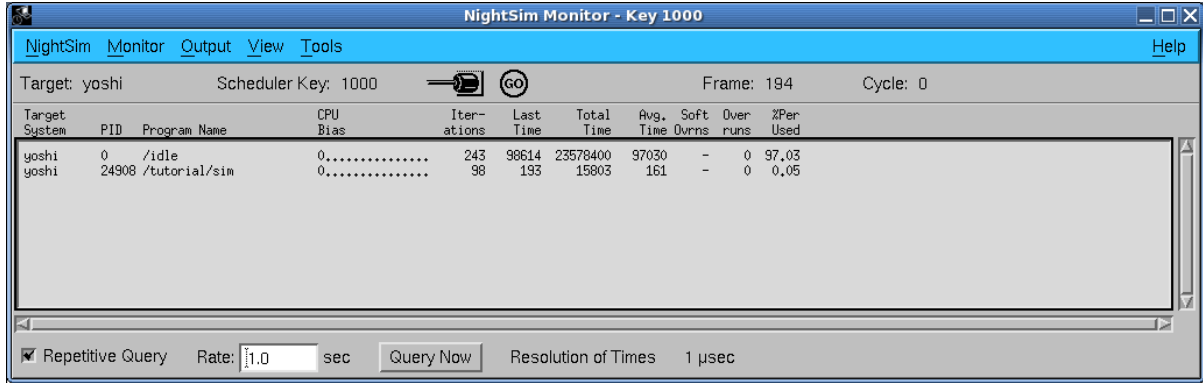
The performance monitor is a mechanism that enables you to monitor FBS-scheduled processes' utilization of a CPU.

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 1-23. 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 `fbs_wait`.

The value under the **Last Time** column for the **/idle** process shows the time during which the CPU was not busy.

## Using NightProbe

The NightProbe Data Monitoring Tool is a real-time graphical tool for monitoring, recording, and altering program data within one or more executing programs without intrusion. It can be used in a development environment as a tool for debugging, or in a production environment to create a “control panel” for program input and output.

NightProbe utilizes a non-intrusive technique of mapping the application’s address space into its own. Subsequent direct reads and writes by NightProbe allow it to sample and modify user data without interrupting or otherwise affecting the user process.

There is no API for NightProbe. Applications need only ensure that their debug information is generated with the DWARF format by using the `-g` compilation option. Even without symbols, however, NightProbe can probe processes based on virtual addresses alone.

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

## Invoking NightProbe

### To invoke the NightProbe from NightSim

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

The NightProbe Main window is opened.

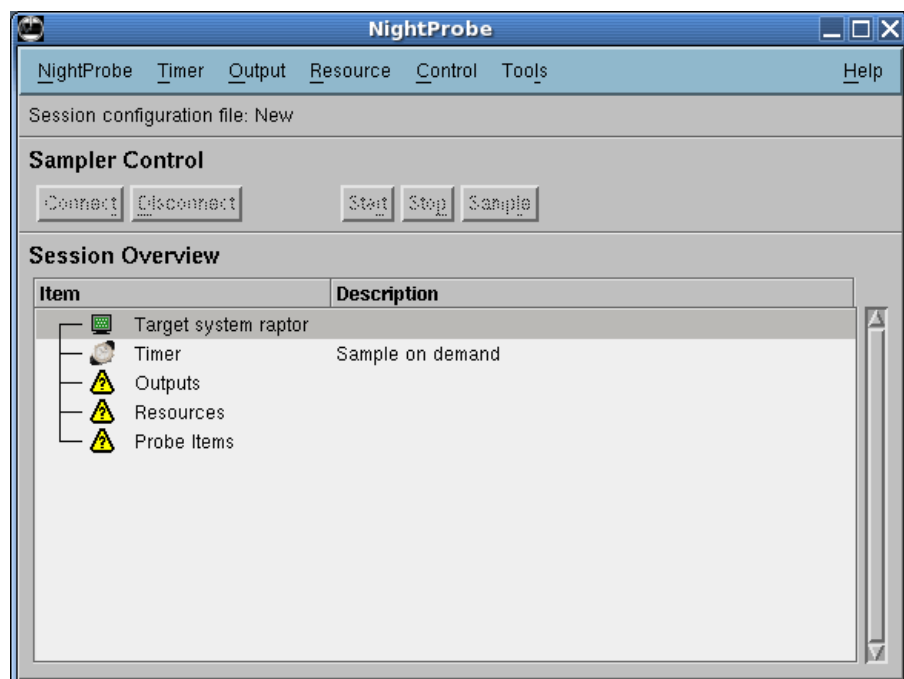
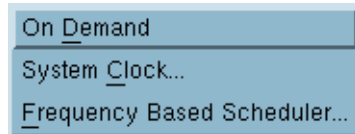


Figure 1-24. NightProbe Main window

## Configuring NightProbe

The user may configure NightProbe by selecting a timing source and output method.

The timing source is selected from the **Timer** menu:



**Figure 1-25. NightProbe Timer menu**

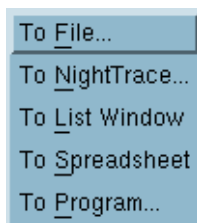
You may instruct NightProbe to sample when the user requests (**On Demand**), to use the system clock as the timing source using a frequency that the user selects (**System Clock**), or to take samples as directed by a frequency-based scheduler (**Frequency-Based Scheduler**).

Press the ESC key to exit this menu.

### NOTE

You can also select the timing source by right-clicking on the **Timing Source** entry in the **Session Overview** area. Each item in the **Session Overview** area has a menu associated with it that can be brought up by right-clicking on that item.

The output method is selected from the **Output** menu:



**Figure 1-26. NightProbe Output menu**

NightProbe can write the output to a file that the user specifies (**To File**), to a file where the sampled data is written in a format usable as input to NightTrace (**To NightTrace**), to a window that will display a scrolling list of the values of the monitored variables as the program runs (**To List Window**), or to a window displaying the values of the monitored variables in a spreadsheet format (**To Spreadsheet**).

Press the ESC key to exit this menu.

### NOTE

You can also select the output method by right-clicking on the **Outputs** entry in the **Session Overview** area. Each item in the **Session Overview** area has a menu associated with it that can be brought up by right-clicking on that item.

Our example will use a configuration file shipped with the PowerWorks Linux Development Environment Installation CD which specifies the timing source to use the system clock sampling at an asynchronous rate of once per second and which displays the output in a spreadsheet format. This file, **nprobe.config**, was copied to our working directory earlier in the step “Getting Started” on page 1-5.

### To configure the NightProbe Data Monitoring Tool

- Select **Open Session...** from the **NightProbe** menu of the **NightProbe Data Recording** window.

You will be presented with a file selection dialog.

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

You should see the following members of the `counters` class listed in the **Variable List** area of the **NightProbe Data Recording** window:

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

We will be probing and modifying these variables.

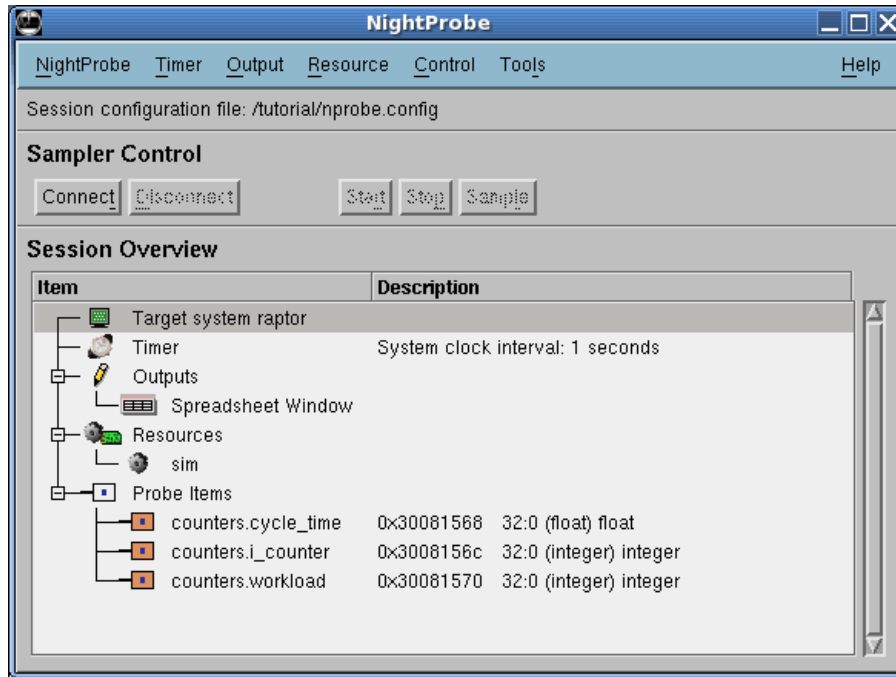


Figure 1-27. Configured NightProbe Data Recording window

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.

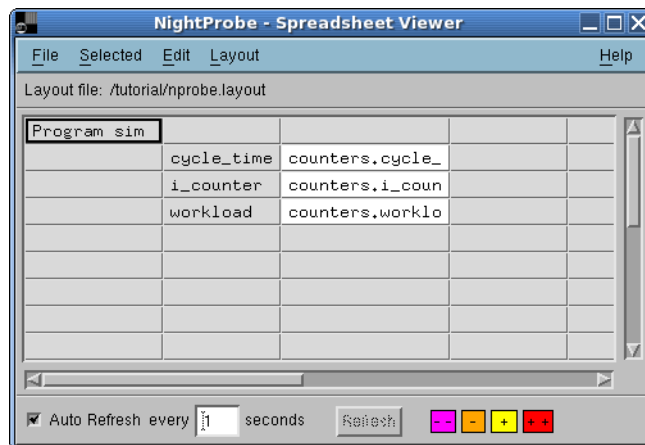


Figure 1-28. NightProbe Spreadsheet Viewer window



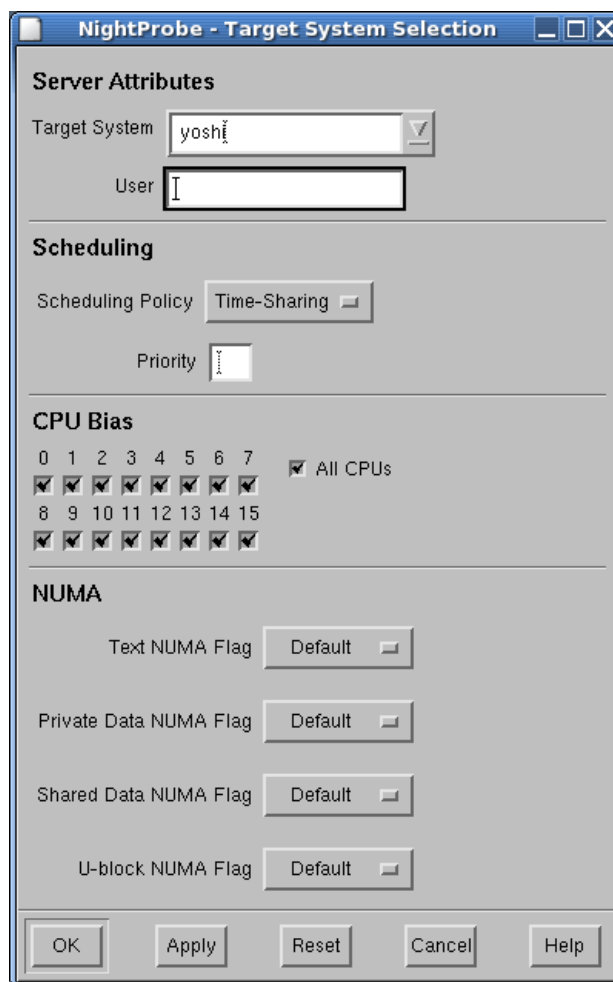
## Selecting the target system

We need to tell NightProbe on which system our **sim** application is running so that it will be able to connect to it.

### To select the target system

- Double-click on the **Target system** item in the Session Overview area of the NightProbe Main window.

You will be presented with the Target System Selection dialog.



**Figure 1-29. Target System Selection dialog**

- Enter the name of the system on which the **sim** application is running in the **Target System** field.
- Enter the login name of the user on the target system in the **User** field.
- Press **OK** to dismiss the dialog.

## Connecting to the target program

When you are ready to perform data recording or monitoring, you must first connect NightProbe to a real-time NightProbe server on the target system.

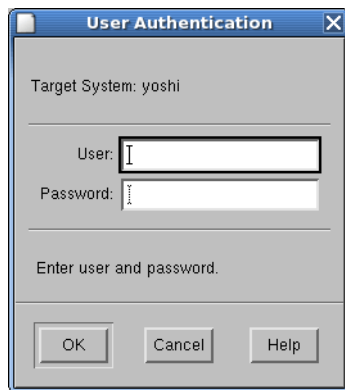
The real-time NightProbe server performs initialization during the connection phase - opening output devices, verifying target processes, and mapping target process variable addresses.

The probed applications are not affected by this operation.

The real-time NightProbe server is the actual process that will read and write values from and to the user application's address space.

### To connect to the target program

- Press the **Connect** button under the **Sampler Control** area of the NightProbe Main window.
- When presented with the **User Authentication** dialog, enter the login name of the user in the **User** field along with the corresponding password in the **Password** field.
- Press the **OK** button to continue.



**Figure 1-30. 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 1-38), 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 under 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.

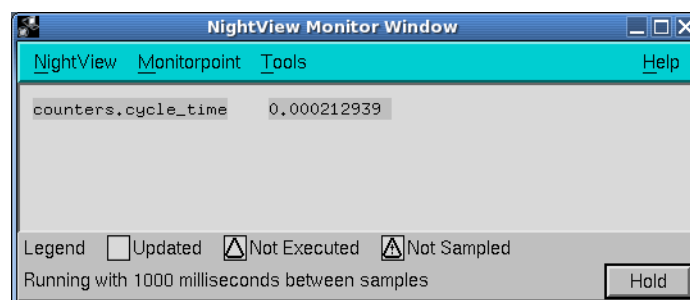
## Modifying program data

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 1000.

Notice that the value for `cycle_time` has increased significantly. In our example, it is now approximately 0.00028 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.)

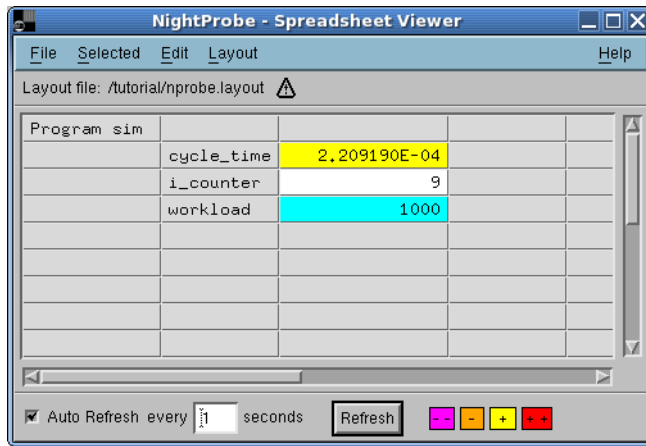


**Figure 1-31. Modified values in NightView Monitor Window**

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 1-38) specify that when the value exceeds 0.0002, the color of the cell will change to yellow signifying a state of high caution.

**NOTE**

Depending on the speed of your machine, the color of the cell may not have changed at all or may have changed to some other color. Experiment with values for `workload` until the `cycle_time` is in the range 0.0002 .. 0.0004999.



**Figure 1-32. Modified values in NightProbe Spreadsheet Viewer**

- Change the value of `workload` to 10000. Notice the color of the cell containing the `cycle_time` value changes to red, signifying a state of high danger.

**NOTE**

Depending on the speed of your machine, the color of the cell may not have changed at all or may have changed to some other color. Experiment with values for `workload` until the `cycle_time` is greater than 0.0005.

- Change the value of `workload` to 100. Notice the color of the cell containing the `cycle_time` value changes back to white.

**NOTE**

Depending on the speed of your machine, the color of the cell may not have changed at all or may have changed to some other color. Experiment with values for `workload` until the `cycle_time` is less than 0.0002.

## Using NightTrace

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

NightTrace allows users to manage user and kernel NightTrace daemons, providing the user with the ability to start, stop, pause, and resume execution of any of the daemons under its management.

## Invoking NightTrace

### IMPORTANT!!!

If you do not have a system running PowerMAX OS networked to your Linux system, you may use NightTrace to analyze the kernel and user trace data files that were copied from the installation CD to the working directory you created in “Getting Started” on page 1-5. Follow the instructions under “Invoking NightTrace from the command line” on page 1-46.

Otherwise, continue with the section titled “Invoking NightTrace from NightProbe” on page 1-51.

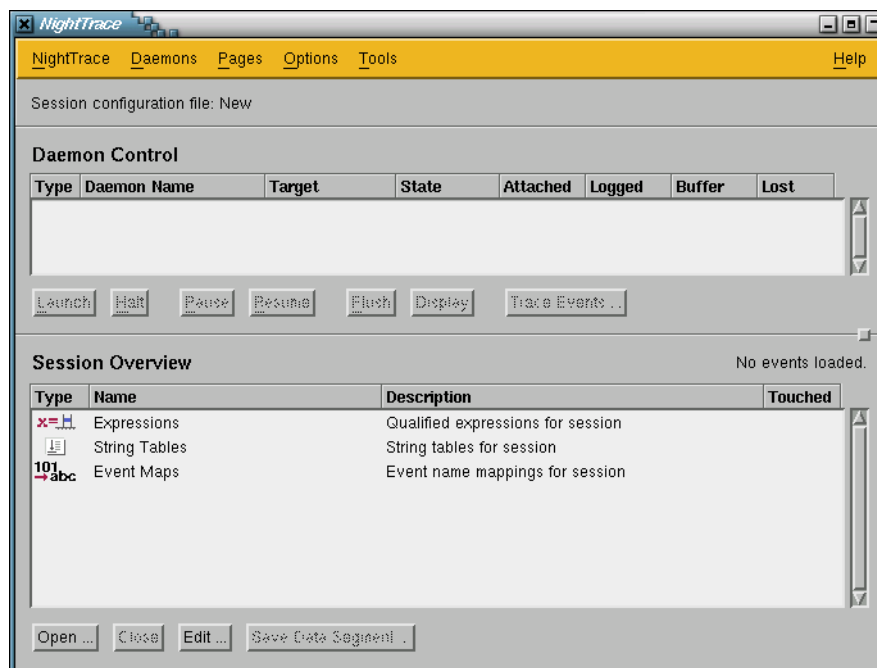
## Invoking NightTrace from the command line

### To invoke NightTrace from the command line

- From the working directory you created in “Getting Started” on page 1-5 on the Linux system, enter the following command

```
ntrace
```

This will open the NightTrace Main Window. (See the section titled “ntrace Arguments” in the *NightTrace User’s Guide* (0890398) for more information about invoking NightTrace.)



**Figure 1-33. 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).

## Loading data and configuration files

Since we do not have a PowerMAX OS system connected to our Linux system, we will load the kernel trace and user trace data as well as a NightTrace configuration file from those files that we copied in “Copying tutorial-related files from the installation CD” on page 1-7.

Those files are:

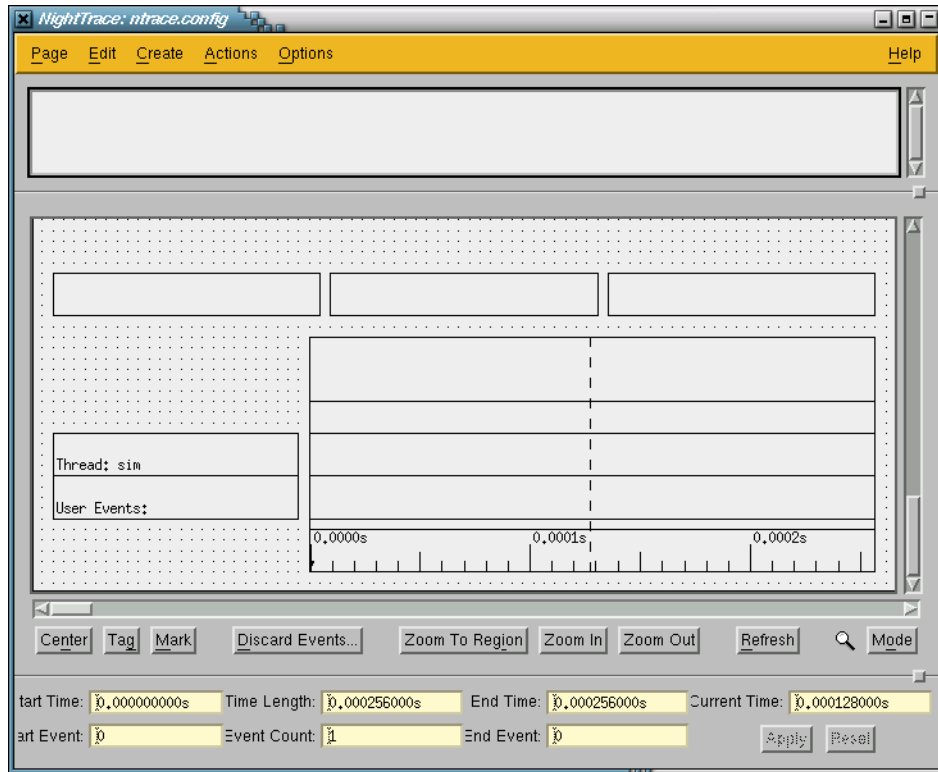
<code>ntrace.kernel-data</code>	a file containing the kernel trace data
<code>ntrace.user-data</code>	a file containing the user trace data
<code>ntrace.config</code>	a file containing string tables, format tables, and a NightTrace display page, including descriptions of NightTrace display objects for the application’s user trace events

### To load data and configuration files

- 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.

NightTrace will present a user trace display page configured using the **ntrace.config** file as shown below:



**Figure 1-34. NightTrace user trace display page**

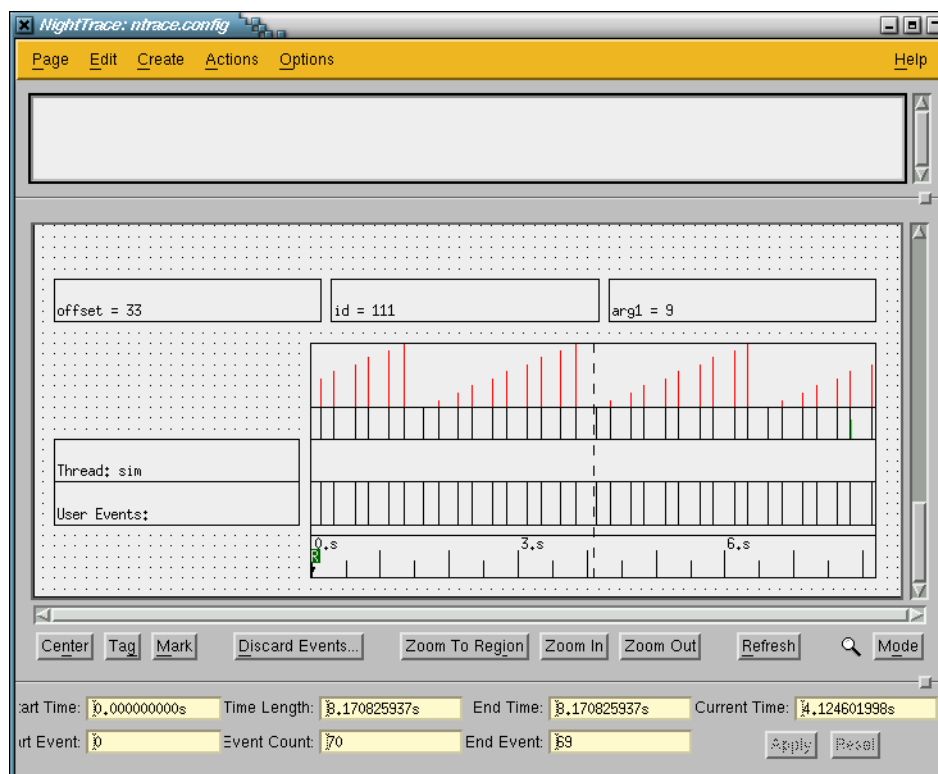
- Press the Open... button at the bottom of the NightTrace Main Window.

You will be presented with an Open Display File dialog.

- Select the file **ntrace.user-trace** from the list of Files.
- Press the OK button to load the user trace data.
- Press the Scroll Out button multiple times until the horizontal scroll bar fills the entire display

NightTrace will load the user trace data into the display page we created in the previous step:





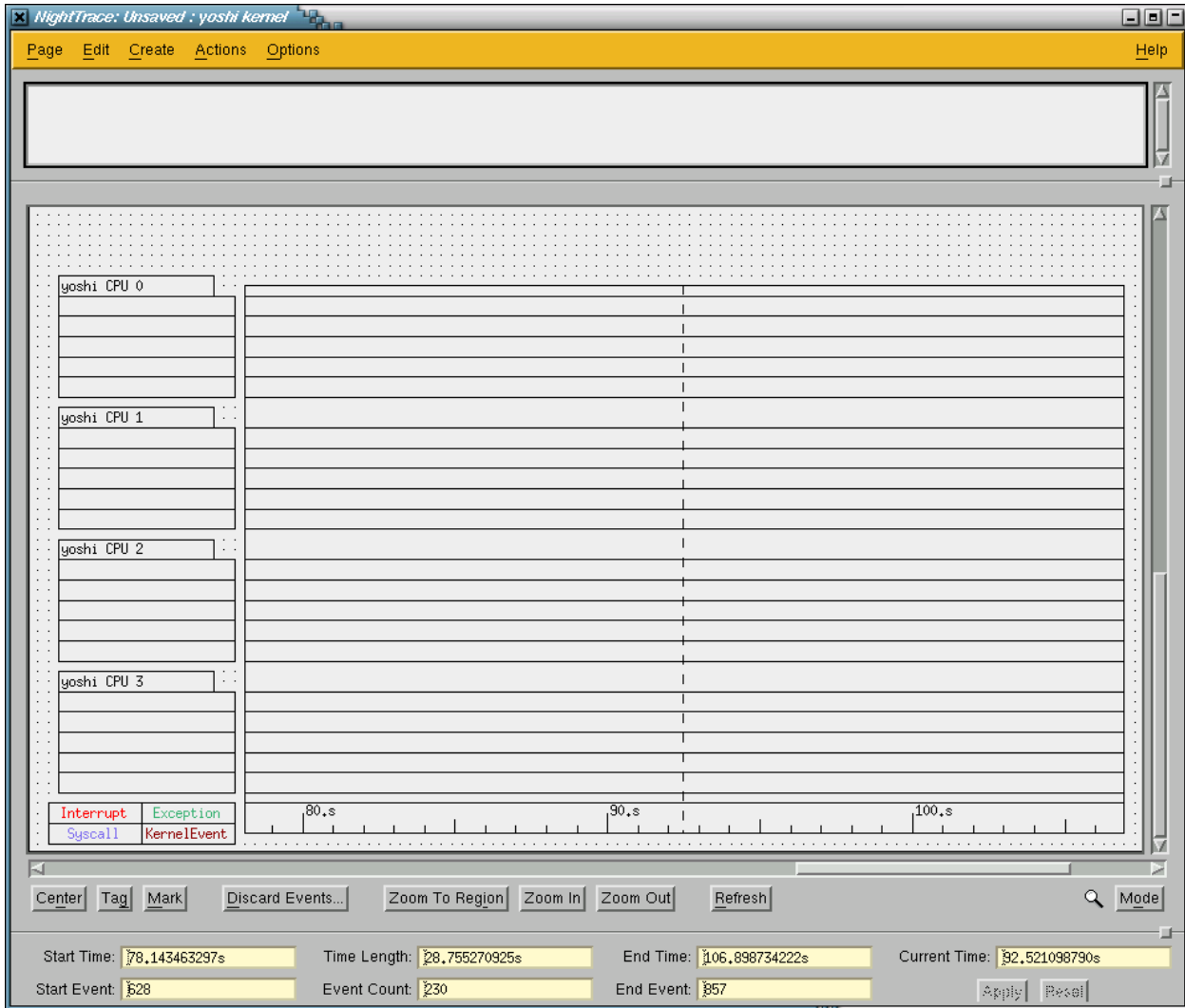
**Figure 1-35. NightTrace user trace display page with data loaded**

- 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.kernel-trace** from the list of Files.
- Press the OK button to load the kernel trace data.
- Press the Scroll Out button multiple times until the horizontal scroll bar fills the entire display area.

NightTrace will create a kernel display page and load the kernel trace data into it as shown below:



**Figure 1-36. NightTrace kernel display page**

For more information on display pages, see The Display Page in the *NightTrace User's Guide* (0890398).

**IMPORTANT!!!**

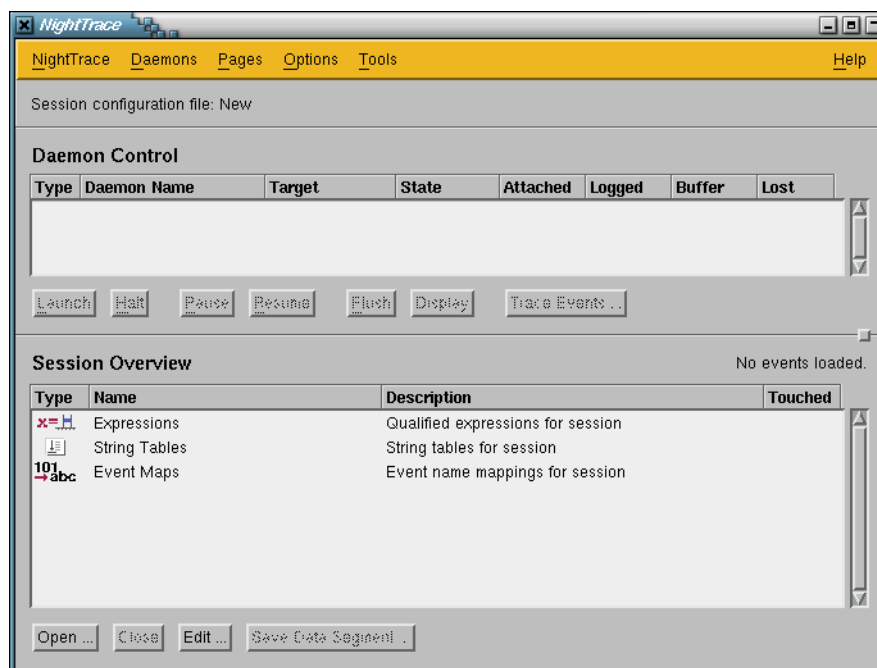
Proceed to the section titled “Positioning the current time line” on page 1-64.

## Invoking NightTrace from NightProbe

### To invoke NightTrace from NightProbe

- From the Tools menu of the NightProbe Data Recording window, select the NightTrace System Tracing and Analysis menu item.

The NightTrace Main Window is opened.



**Figure 1-37. 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).

NightTrace allows users to manage user and kernel NightTrace daemons. It provides users with the ability to define a session consisting of one or more daemon definitions which can be saved for future use. 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.

Using NightTrace, users can manage multiple daemons simultaneously on multiple target systems from a central location.

NightTrace offers the user the ability to start, stop, pause, and resume execution of any of the daemons under its management. The user may also view statistics as trace data is being gathered as well as dynamically enable and disable events while a particular daemon is executing.

## Configuring a user daemon

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

User trace events are generated by:

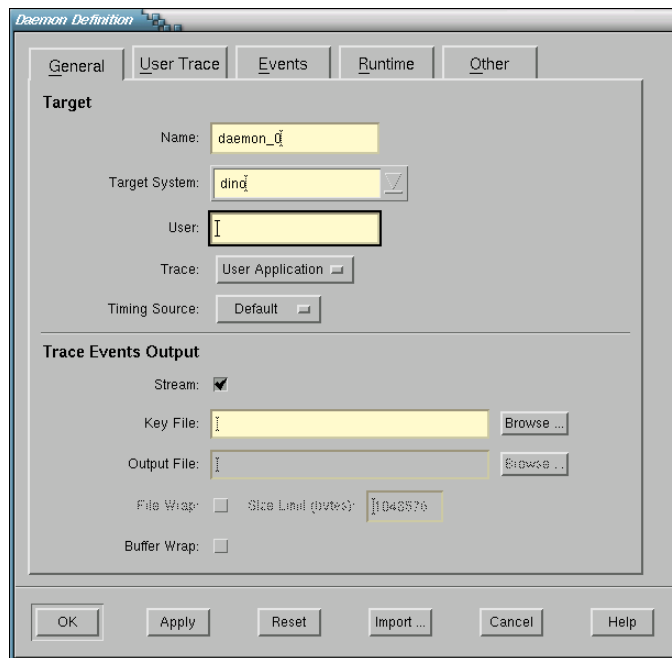
- user applications that use the NightTrace API
- NightProbe (see the description of the **To NightTrace** menu item in the chapter titled “Using the Data Recording Window” in the *NightProbe User’s Guide* (0890480).

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

### To configure a user daemon

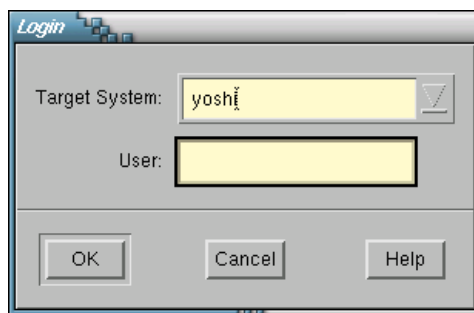
- From the **Daemons** menu on the NightTrace Main Window, select the **New...** menu item.

The Daemon Definition Dialog is displayed.



**Figure 1-38. Daemon Definition dialog**

- Press the **Import...** button at the bottom of the Daemon Definition Dialog.  
You will be presented with a Login dialog.



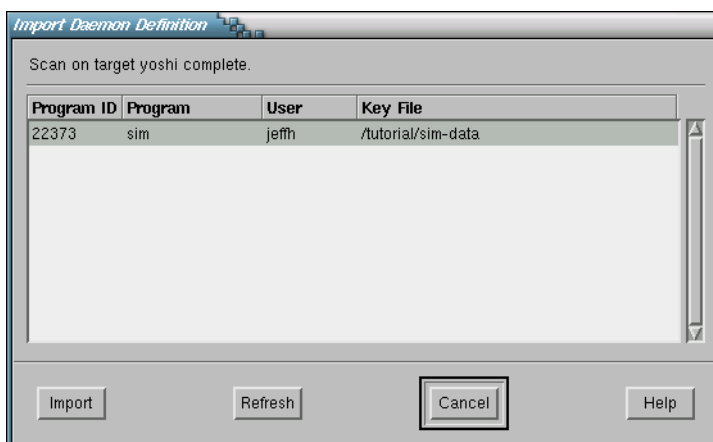
**Figure 1-39. Login dialog**

- 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.

You will then be presented with the Enter Password dialog.

- Enter the password for the user specified in the Login dialog.

When user authentication completes, the Import Daemon Definition dialog is presented.



**Figure 1-40. 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 OK button.

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

- Press OK 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 PowerWorks Linux Development Environment Installation CD. This file, named **ntrace.config**, was copied to our local **tutorial** directory earlier in the step “Getting Started” on page 1-5.

### **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.

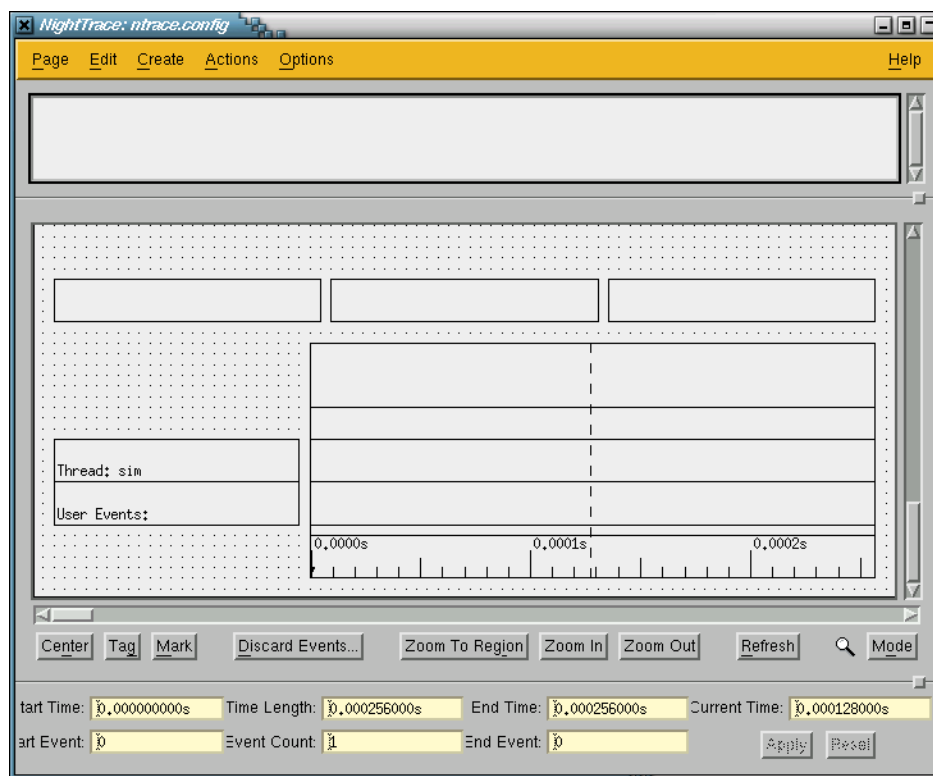


Figure 1-41. Customized NightTrace display page

## 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 Details 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

Launching 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 Details 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.



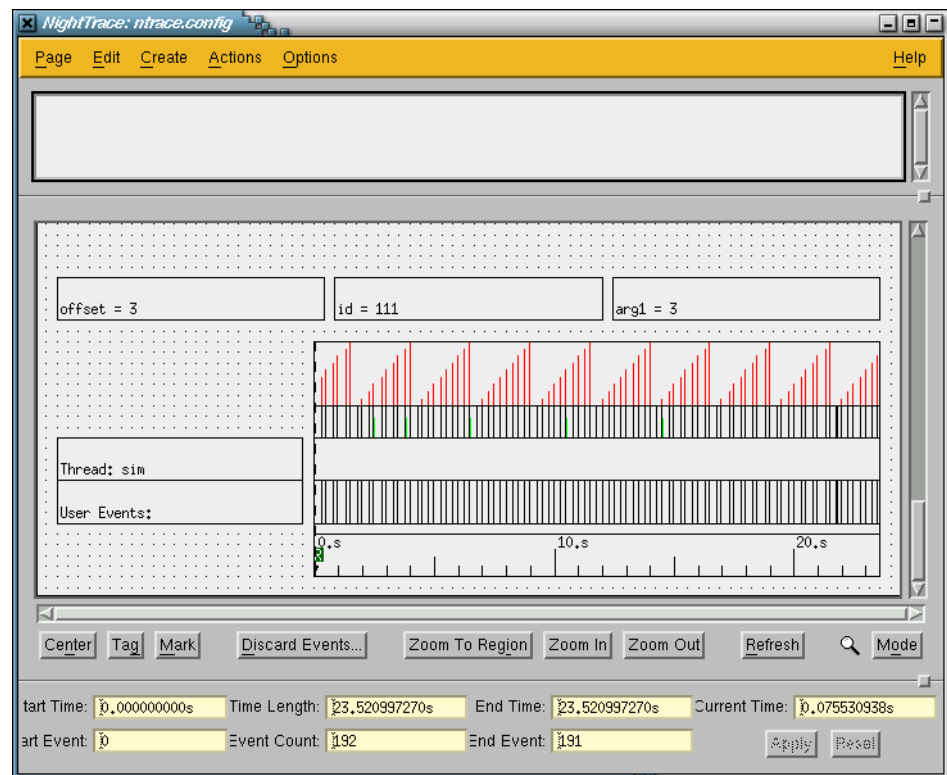


Figure 1-42. User trace data in customized NightTrace display page

#### 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 mod 10;
ntb.trace_event(cycle_end, arg);
```

### To insert a patchpoint in a program

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

```
ntb.trace_event(cycle_end, arg);
```

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

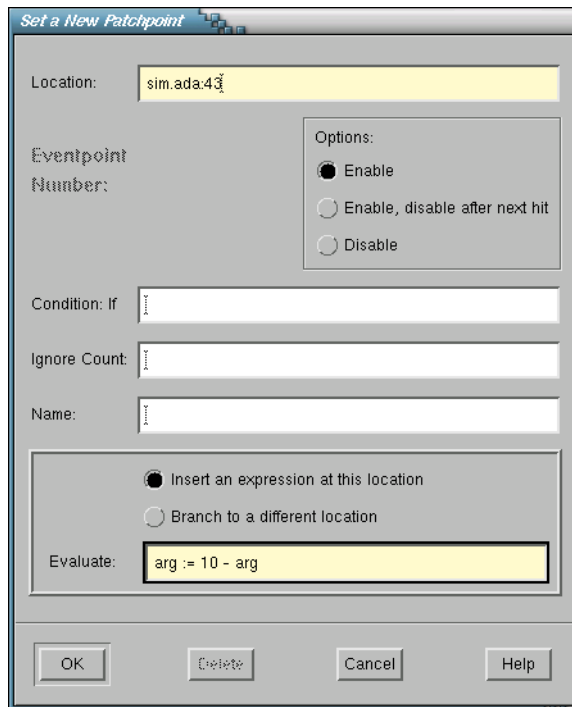


Figure 1-43. Setting a new patchpoint

- 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 1-57), 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 NightTrace display buffer, we may view it immediately even while additional trace data is being collected.

### To view streaming data

- On the Interval Control Bar under the grid on the NightTrace display page, press the right arrowhead 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).

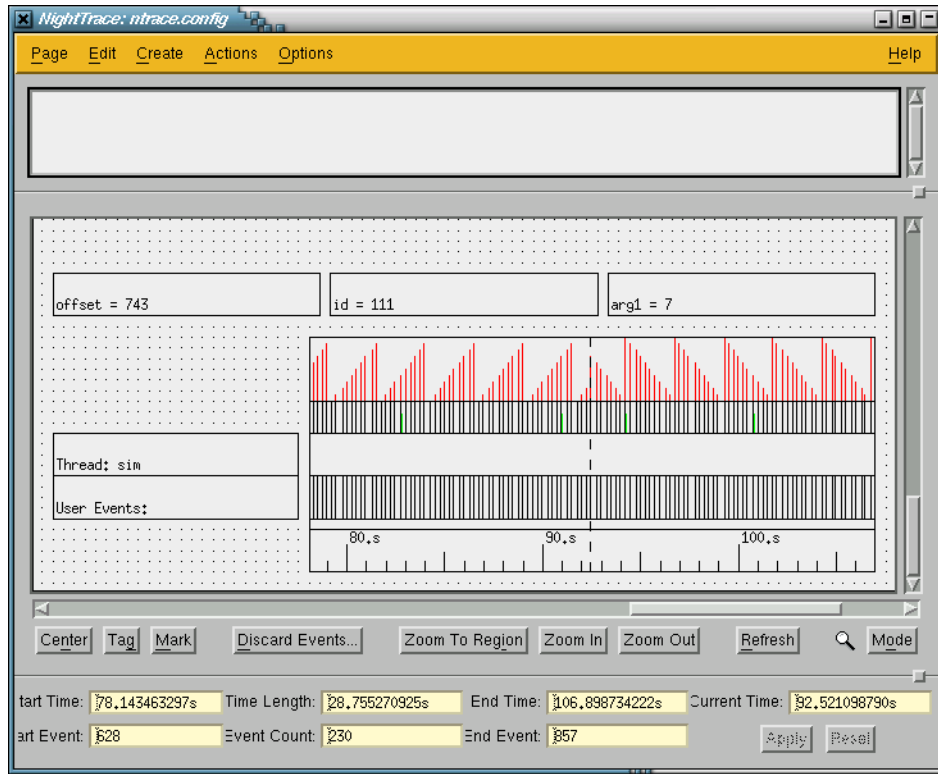


Figure 1-44. User trace data after patchpoint inserted

**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!

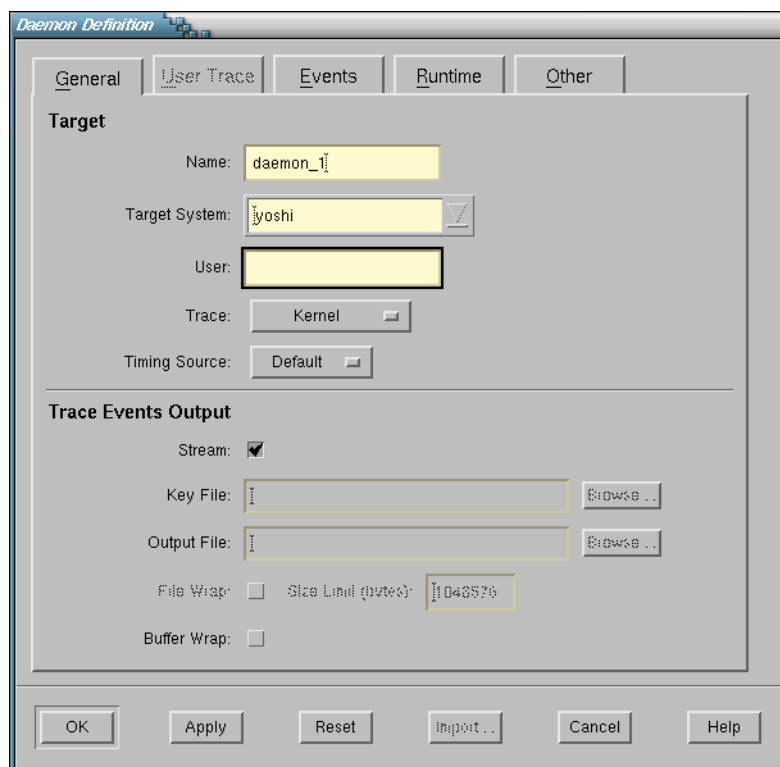
## Configuring a kernel daemon

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 1-45. Daemon Definition dialog**

- Select the **Kernel** radiobutton 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

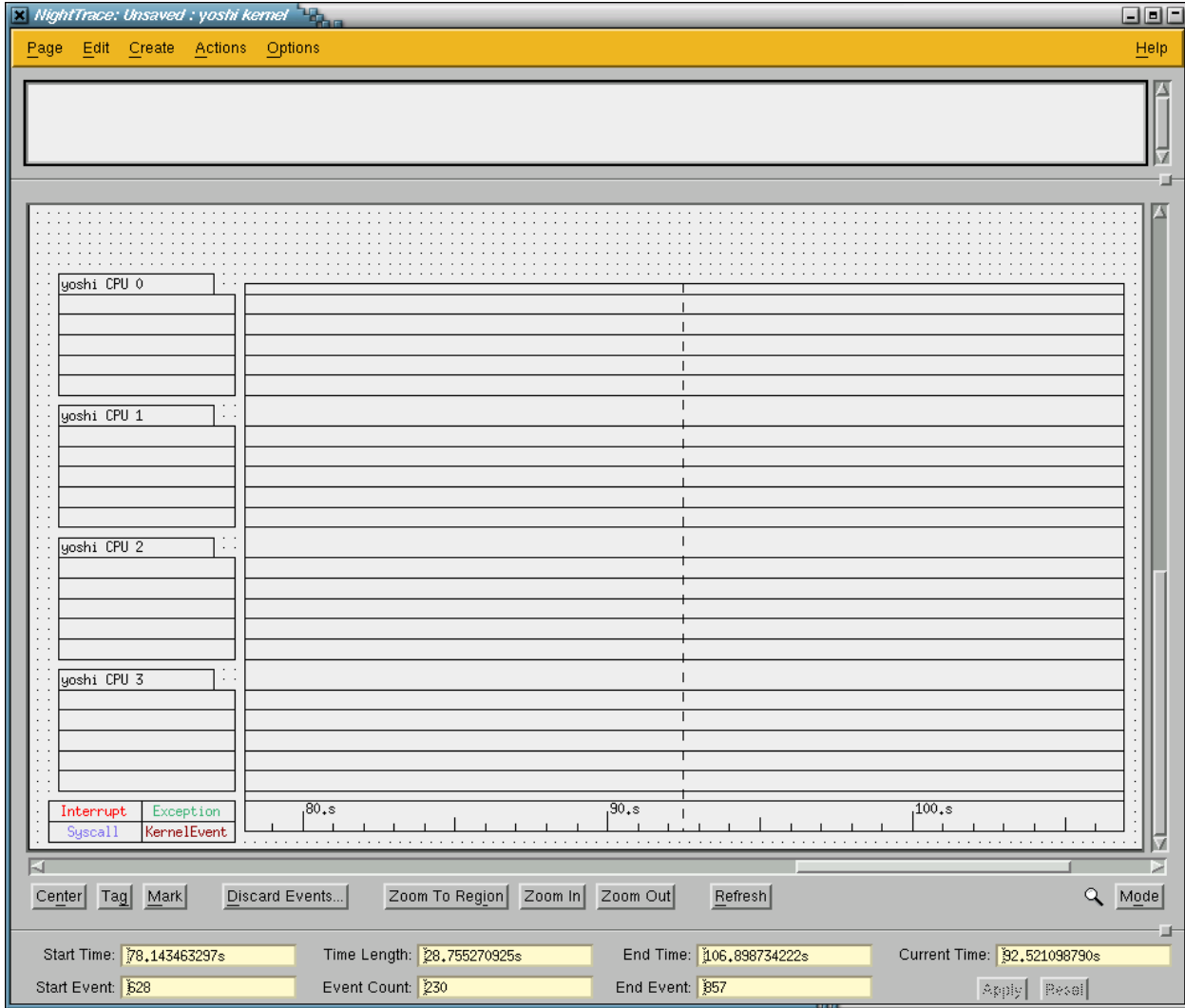
Once the daemons are configured, they must be created before they can begin collecting events.

### To create the daemons

- Select the kernel daemon in the **Daemon Details 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.



**Figure 1-46. NightTrace kernel display page**

**NOTE**

Launching 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 Details 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.

### 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.

### 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.

## Displaying the kernel trace data

As we are collecting trace data from the PowerMAX OS 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 Details 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 the setting of the **Stream** checkbox on the **General** page of the Daemon Definition dialog), 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 Details Area of the NightTrace Main Window).
- 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

- Select *both* daemons from the Daemon Details Area of the NightTrace Main Window).
- Press Halt.

The state of both daemons change from Running to Halted.

## Positioning the current time line

For those users without a PowerMAX OS system networked to their Linux system, we will continue with an analysis of the trace data that we loaded in “Loading data and configuration files” on page 1-47.

For those users who have generated live data from a PowerMAX OS system networked to their Linux system, we will analyze that data in the following sections.

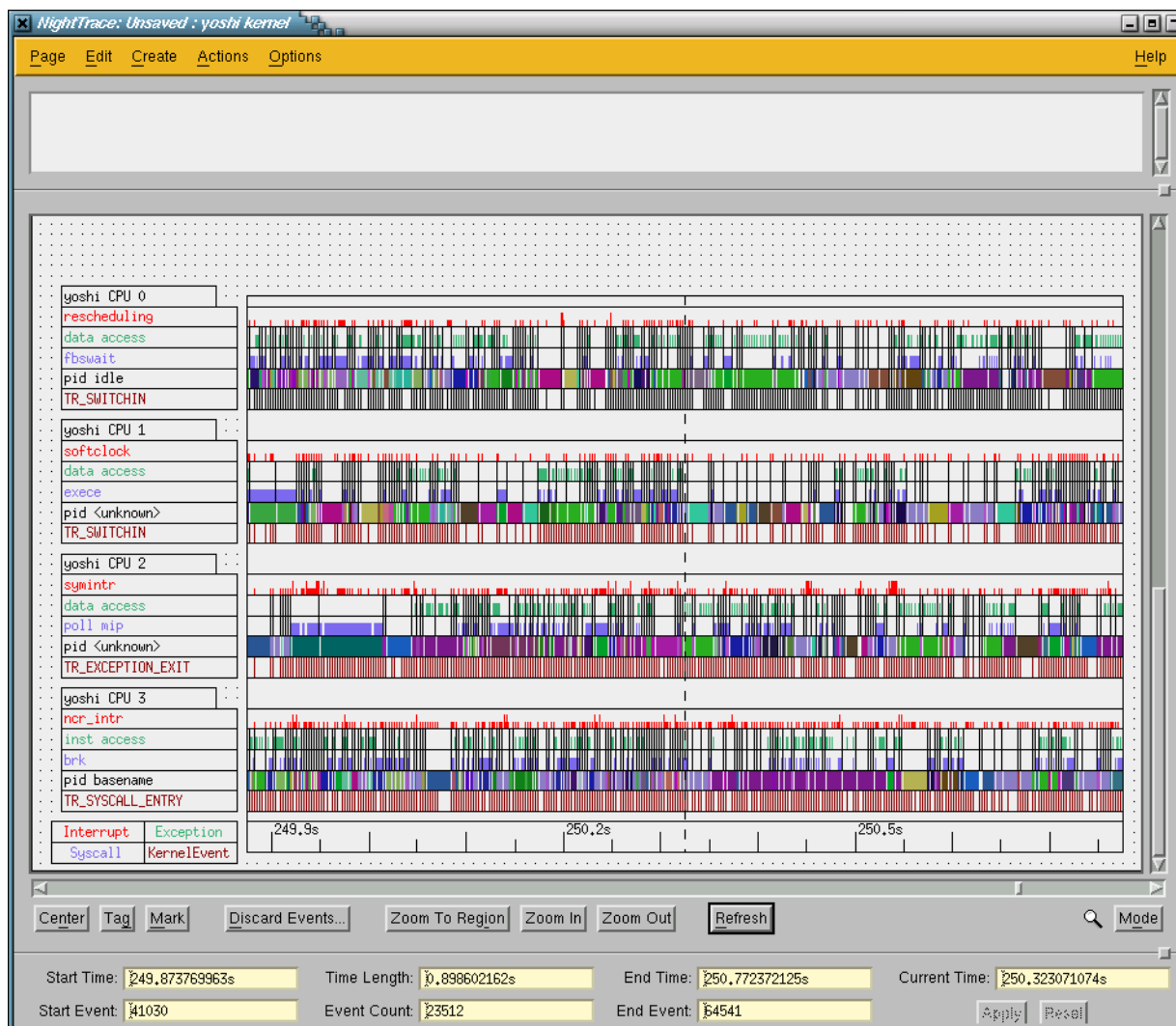
We will position the current time line to a point somewhere after the kernel trace data started being generated.

### To position the current time line

- On the Interval Control Bar under the grid on the *kernel display page*, press the right arrowhead 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.



**Figure 1-47. NightTrace kernel trace data**

The DataBoxes are updated based on the current position of the current timeline and indicate the last value of each data item that occurred on or before the timeline 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 PowerWorks Linux Development Environment Installation CD to our working directory in the step “Getting Started” on page 1-5. 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.

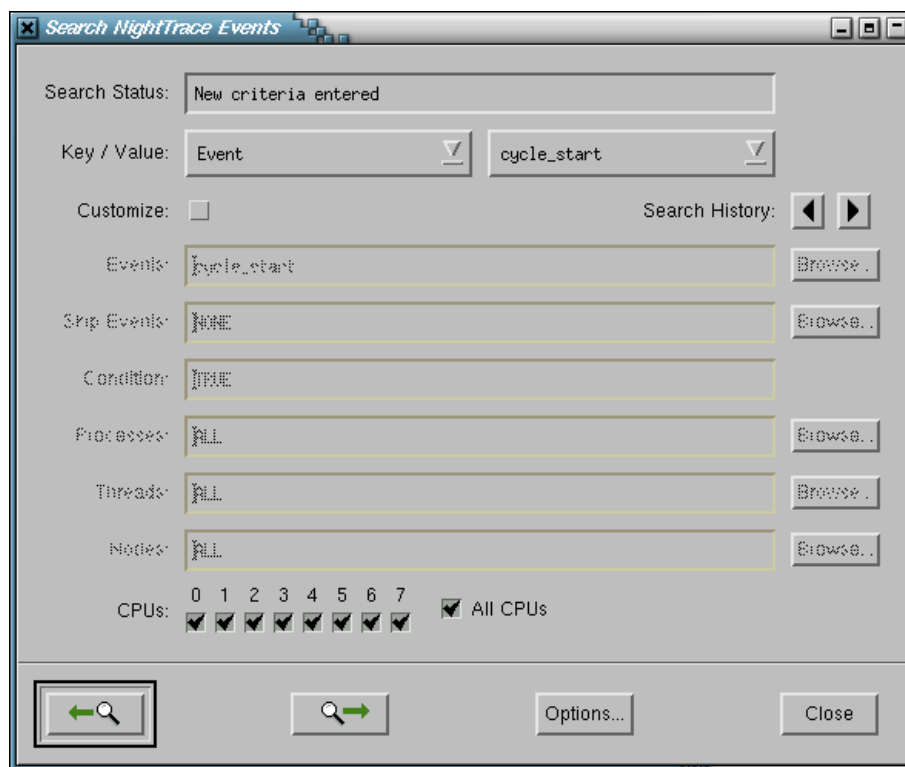
- Select the file **ntrace.eventmap** from the list of **Files**.
- Press the **OK** button to load the eventmap file.

## Searching for a user trace event

### To search for a user trace event

- Select the **Search...** menu item from the **Tools** menu on the NightTrace display page containing the user trace data.

The NightTrace **Search** dialog is presented.



**Figure 1-48. NightTrace Search dialog**

- Enter `cycle_start` in the Event List field.

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

```
ntb.trace_event(cycle_start, counters.get);
```

#### NOTE

Because we loaded the `ntrace.eventmap` file (“Loading an eventmap file” on page 1-66), 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 **Search** button.
- 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.

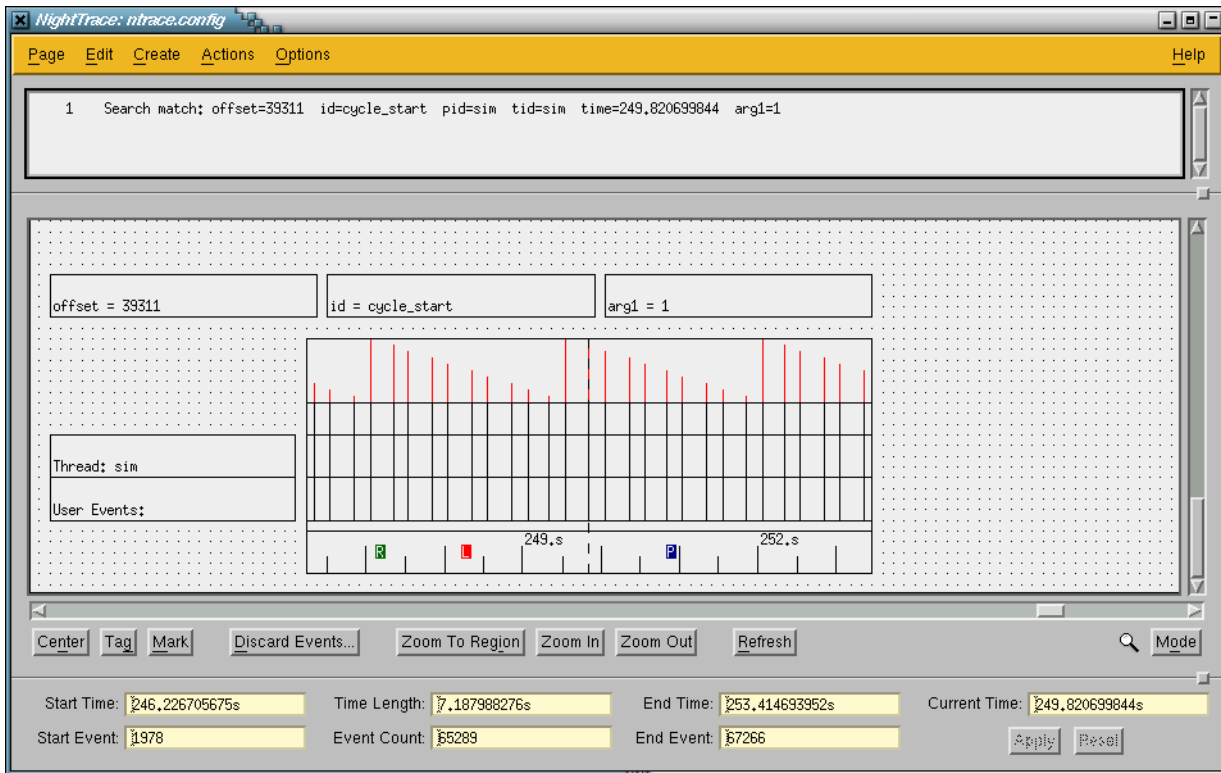


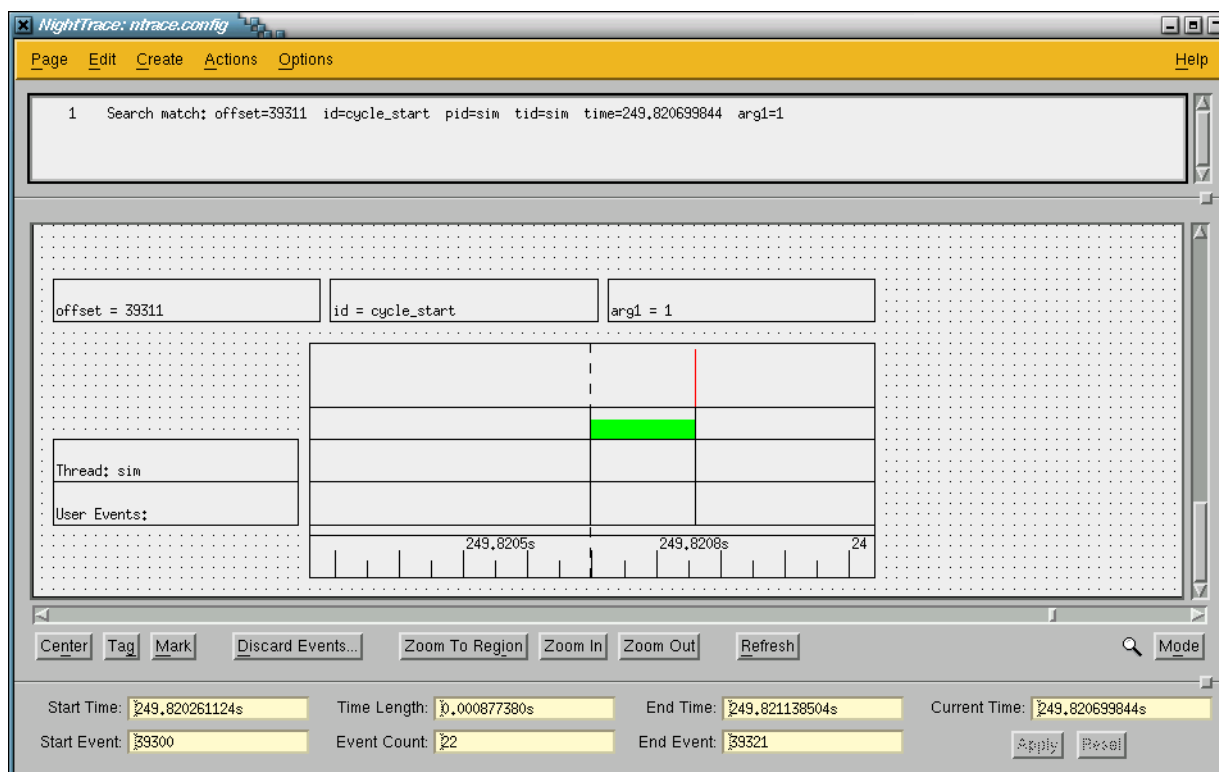
Figure 1-49. User trace data after search

## 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 1-50. Zoomed in view of user trace data**

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

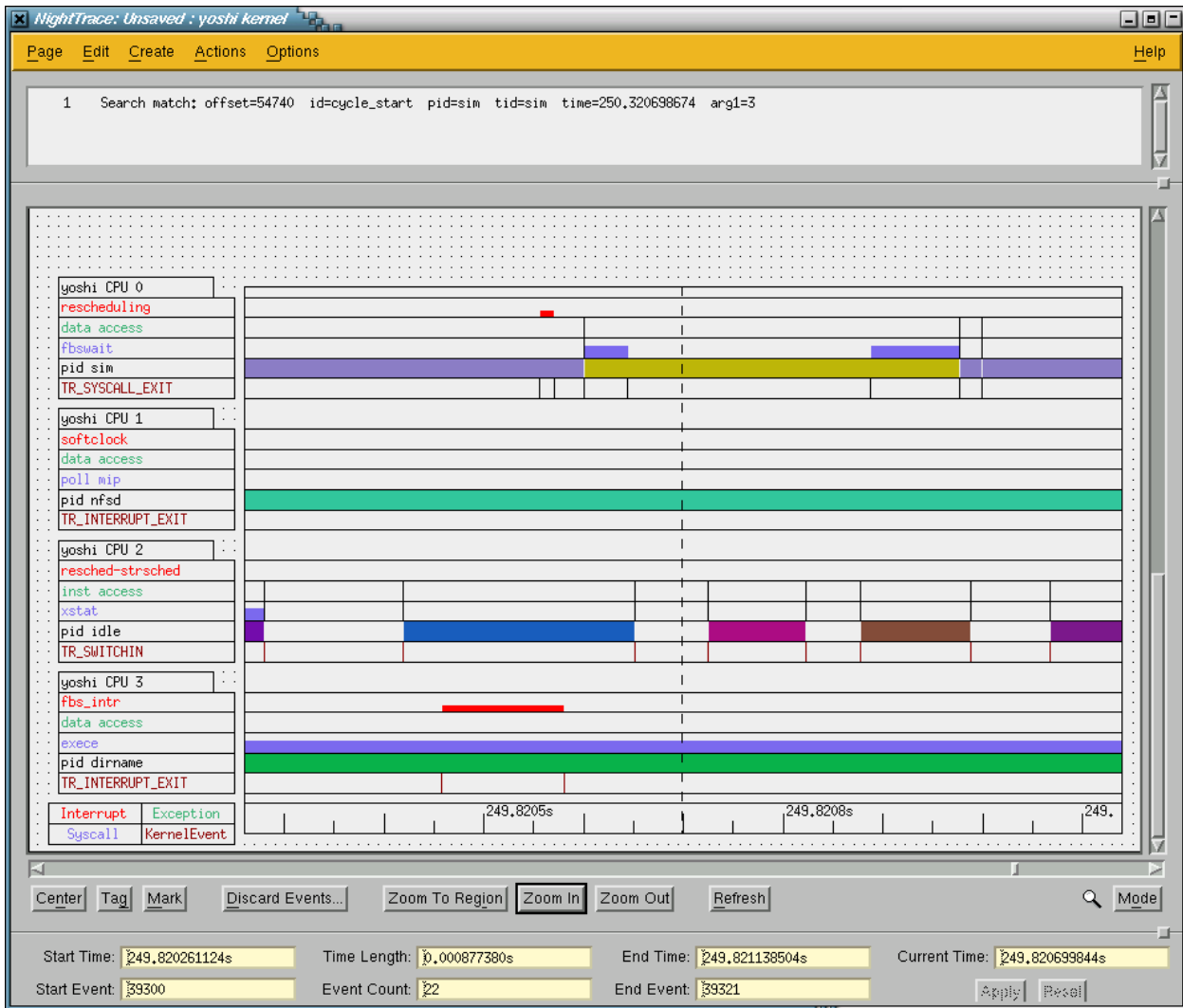


Figure 1-51. Zoomed in view of kernel display page

**NOTE**

The following analysis of the kernel trace data is based on Figure 1-51. If you are analyzing live data, your kernel display page may look different. You may see additional activity, most likely red interrupt activity, between the exit and reentry to `fb_wait`. Additionally, the colors in the PID rows will differ.

The boxes with textual information on the left hand side of the window change when the current time line changes. The current time line is the vertical dashed lined that runs the entire length of the display grid. The text in the boxes to the left describes the most recent item to the left of the time line. Each row describes a different event or entity. Items in the Red row are interrupts, items in the green row are machine exceptions, items in the blue row are system calls, items in the next row are process identifiers and items in the last row individual kernel trace events.

In Figure 1-51, the first red bar displayed on the grid for CPU 3 indicates the interrupt from the RCIM device. (If you collected your own kernel data, the CPU where the interrupt occurred could be on any CPU.) Note that the red text box to the left for CPU 3 indicates it was an `rcimintr` interrupt.

**NOTE**

In Figure 1-51, the description of the interrupt name may be either `rcimintr` or `fb_intr`, depending on your system configuration.

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

The application then performs its calculations (as indicated by the goldenrod-colored bar in the PID row) before it comes back to the `fbwait` call (the second blue bar).

The lack of any activity in the white space in the top three rows for that CPU indicates that the user application did not make any intervening system calls, received no machine exceptions, and was not disturbed by some other interrupt. The PID graphs shows a colored box when a process is context-switched onto a CPU. The color of each process is automatically generated (and in this case is goldenrod) so that context switches are readily identified. Each process retains its assigned color throughout the data set.

In a real-life scenario, we would tune and shield the system for optimal real-time performance.

## Exiting the tools

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

## Exiting NightTrace

### To exit NightTrace

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

## Exiting NightProbe

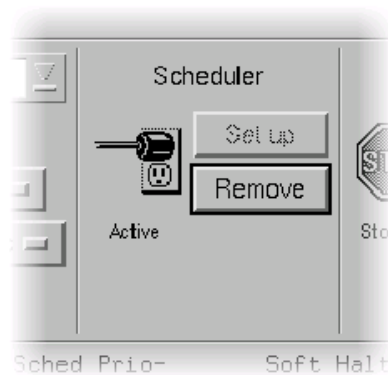
### To exit NightProbe

- From the NightProbe Data Recording window, press the Stop button to stop sampling data.
- Press the Disconnect button to disconnect from the application.
- From the File 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

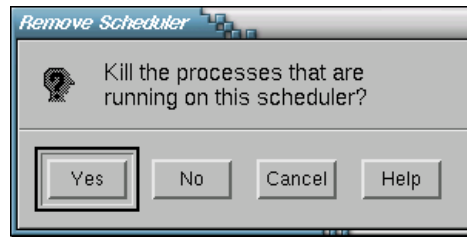
- In the NightSim Scheduler window, press the Stop button.
- Press the Remove button.



**Figure 1-52. Removing the scheduler**

You will be presented with the following dialog:





**Figure 1-53. 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.

## Exiting NightBench

### To exit NightBench

- From the NightBench Development window, select **Exit NightBench Session** from the Development menu.

## Conclusion

This concludes our tutorial for the PowerWorks Linux Development Environment.



# **A**

## **Tutorial Files**

---

The following sections show the source listings for the files used in the *PowerWorks Linux Development Environment Tutorial*.

## sim.ada

```
with night_trace_bindings;
with rt_interface;
with ada.real_time;
with ada.real_time.conversions;
with counters;

procedure sim is
--
  package rt   renames rt_interface;
  package ntb  renames night_trace_bindings;
  package art  renames ada.real_time;
  package artc renames ada.real_time.conversions;

  arg      : integer;
  status   : integer;
  cycle_start : constant := 110;
  cycle_end  : constant := 111;
  start     : art.time;
  stop      : art.time;
--
begin
--
  pragma task_cpu_bias(illegal_use_of); -- Remove this line

  ntb.trace_begin (trace_file => "sim-data",
                  use_spl     => false,
                  use_resched => false,
                  lock_pages  => false);
  ntb.trace_open_thread ("sim");

  counters.set_workload(0);

  loop
    rt.fbs_wait(status);
    exit when status /= 0;
    start := art.clock;
    counters.increment(1);
    ntb.trace_event(cycle_start, counters.get);
    counters.work;
    stop := art.clock;
    arg := counters.get mod 10;
    ntb.trace_event(cycle_end, arg);
    counters.cycle_time := float(artc.to_seconds(art."-"(stop,start)));
  end loop;

  ntb.trace_end;
--
end sim;
```

**counters.ada**

```

package counters is
--
  procedure increment(i : integer);
  procedure set_workload (workload : integer);

  function calculate return integer;
  function get return integer;

  procedure work;
  cycle_time : float;
--
private
--
  i_counter : integer := 0;
  workload : integer := 10000;
--
end counters;

with ada.real_time;
with ada.real_time.conversions;
package body counters is
--
  procedure spin (seconds : long_float) is
    use ada.real_time;
    mark : time := clock;
  begin
    loop
      exit when
        ada.real_time.conversions.to_seconds(clock-mark) >= seconds;
    end loop;
  end spin;

  procedure increment (i : integer) is
  begin
    i_counter := (i_counter + i) mod 10;
  end increment;

  procedure work is
    x : long_float := 0.0;
    pragma volatile (x);
  begin
    for i in 1..workload loop
      x := x * long_float(calculate);
    end loop;
    spin (0.000100);
  end work;

  function calculate return integer is
  begin
    return i_counter * 2;
  end calculate;

  function get return integer is
  begin
    return i_counter;
  end get ;

```

```
procedure set_workload (workload : integer) is
begin
    counters.workload := workload;
end set_workload;
--
end counters;
```

## art.conversions.ada

```
package ada.real_time.conversions is
--
--   function to_seconds (t : time_span) return long_float ;
--
end ada.real_time.conversions ;

package body ada.real_time.conversions is
--
--   function to_seconds (t : time_span) return long_float is
--     sc : seconds_count;
--     ts : time_span;
--     res : long_float ;
--   begin
--     split(time'(time_of(0,t)), sc, ts);
--     res := long_float (sc) ;
--     ts := ts * 1_000_000_000 ;
--     split(time'(time_of(0, ts)), sc, ts);
--     return res + (long_float(sc)/1_000_000_000.0) ;
--   end to_seconds ;
--
end ada.real_time.conversions ;
```









**Spine for 1/2" Binder**

**Product Name: 0.5" from  
top of spine, Helvetica,  
36 pt, Bold**

**Volume Number (if any):  
Helvetica, 24 pt, Bold**

**Volume Name (if any):  
Helvetica, 18 pt, Bold**

**Manual Title(s):  
Helvetica, 10 pt, Bold,  
centered vertically  
within space above bar,  
double space between  
each title**

**Bar: 1" x 1/8" beginning  
1/4" in from either side**

**Part Number: Helvetica,  
6 pt, centered, 1/8" up**

**PowerWorks Linux  
Development Environment**

**Tutorial**

**0898100**

