POWER8 Functional Simulator

User's Guide

Advance

6 June 2014

© Copyright International Business Machines Corporation 2014

Printed in the United States of America June 2014

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, and service names may be trademarks or service marks of others.

All information contained in this document is subject to change without notice. The products described in this document are NOT intended for use in applications such as implantation, life support, or other hazardous uses where malfunction could result in death, bodily injury, or catastrophic property damage. The information contained in this document does not affect or change IBM product specifications or warranties. Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of IBM or third parties. All information contained in this document was obtained in specific environments, and is presented as an illustration. The results obtained in other operating environments may vary.

While the information contained herein is believed to be accurate, such information is preliminary, and should not be relied upon for accuracy or completeness, and no representations or warranties of accuracy or completeness are made.

Note: This document contains information on products in the design, sampling and/or initial production phases of development. This information is subject to change without notice. Verify with your IBM field applications engineer that you have the latest version of this document before finalizing a design.

THE INFORMATION CONTAINED IN THIS DOCUMENT IS PROVIDED ON AN “AS IS” BASIS. In no event will IBM be liable for damages arising directly or indirectly from any use of the information contained in this document.

IBM Systems and Technology Group
2070 Route 52, Bldg. 330
Hopewell Junction, NY 12533-6351

The IBM home page can be found at **ibm.com®**.

Version 1.0
6 June 2014

Contents

List of Figures

List of Tables

Revision Log

Each release of this document supersedes all previously released versions. The revision log lists all significant changes made to the document since its initial release. In the rest of the document, change bars in the margin indicate that the adjacent text was modified from the previous release of this document.

Revision Date	Pages	Description
6 June 2014	—	Initial release (1.0).

About this Document

The IBM® POWER8™ Functional Simulator has been developed and refined in conjunction with several design projects built upon the IBM Power architecture. The POWER8 Functional Simulator enables hardware and software developers to simulate a POWER8 processor-based system to develop and enhance application support for this platform.

The *IBM POWER8 Functional Simulator User's Guide* describes the basic structure and operation of the POWER8 Functional Simulator and its command-line user interface.

Intended Audience

This document is intended for designers and programmers who are developing and testing applications that are designed to run on systems based on the POWER8 processor. Potential users include:

- System and software designers
- Hardware and software tool developers
- Application and product engineers

Using this Guide

The guide is organized into topics that cover concepts and procedures for initiating and running a functional simulation of a POWER8 system. This book includes the following chapters:

- *Section 1 Introduction to the POWER8 Functional Simulator* describes the POWER8 Functional Simulator and introduces the POWER8 platform modeled by the POWER8 Functional Simulator.
- *Section 2 Command Syntax and Usage* describes the POWER8 Functional Simulator command framework and introduces the structure, format, and usage of simulator commands.
- *Section 3 Accessing the Host Environment* describes several mechanisms that are provided to allow interactions between the host and simulated systems.

Conventions

This guide provides screen captures to illustrate example interface elements and uses code samples to represent example implementations. Your software interface or development environment might vary from these examples depending on your system and product environment.

The typographical conventions shown in the following table are used to indicate the command syntax and to clarify meaning.

Typographical Conventions

Convention	Description
Bold typeface	Represents information and controls displayed on screen, including menu options, application pages, windows, dialogs, and field names.
Italics typeface	Used to emphasize new concepts and terms, and to stress important ideas.
`Bookmaster Gothic` typeface	Represents example code, such as XML output or C/C++ code examples. When referenced in the main text, it also represents information such as: • Commands, file names, and directories • In-line programming elements, such as function names and XML elements *`Italic Bookmaster Gothic`* in code and commands represent values for variables that you must supply, such as arguments to commands or path names for your particular system. For example: `cd /users/`*`your_name`*
... . . . (Horizontal or Vertical ellipsis)	In format and syntax descriptions, an ellipsis indicates that some material has been omitted to simplify a discussion.
{ } (Braces)	Enclose a list from which you must choose an item or information in syntax descriptions.
[ ] (Brackets)	Enclose optional items in format and syntax descriptions. For example, in the statement SELECT [DISTINCT], DISTINCT is an optional keyword.
| (Vertical rule)	Separates items in a list of choices enclosed in { } (braces) in format and syntax descriptions.
UPPERCASE	Indicates keys or key combinations that you can use. For example, press CTRL + ALT + DEL.
Hyperlink	URLs are displayed in blue text to denote a virtual link to an external site. For example: http://www.ibm.com
Note: This is note text.	"**Note:**" denotes information that emphasizes a concept or provides peripheral information.

Related Documents

The simulator's command interface is implemented as an extension of the Tool Control Language (Tcl). Information about Tcl syntax and features can be found in:

- *Practical Programming in Tcl and Tk* by Brent B. Welch. Prentice Hall, Inc.

Among the documents available in OpenPOWER Connect, an IBM online technical library, the following are particularly helpful in understanding the operation of the POWER8 Functional Simulator:

- *POWER8 Processor User's Manual for the Single-Chip Module*
- *POWER8 Processor Datasheet for the Single-Chip Module*
- *POWER8 Processor SCM and Memory Buffer Hardware Errata Notice*
- *Power ISA User Instruction Set Architecture - Book I (Version 2.07)*
- *Power ISA Virtual Environment Architecture - Book II (Version 2.07)*
- *Power ISA Operating Environment Architecture (Server Environment) - Book III-S (Version 2.07)*

Help and Support

For questions or to request technical support, contact OpenPOWER@us.ibm.com

1. Introduction to the POWER8 Functional Simulator

This chapter provides an overview of the POWER8 Functional Simulator, also referred to in this document as systemsim-p8. It provides concepts and procedures for using the simulator for the POWER8 Processor. It also describes configuration parameters for setting up and running the simulation environment in standalone and Linux mode. Topics in this chapter include:

- Simulator Overview
- Installing and Invoking the Simulator
- Simulator Basics

1.1 Simulator Overview

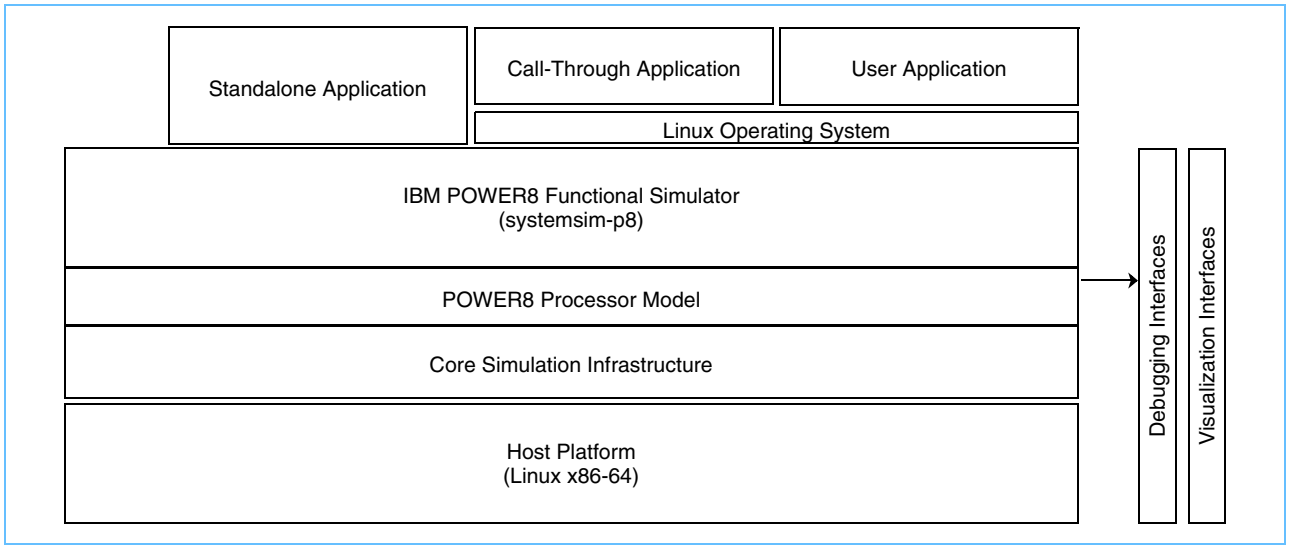
The POWER8 Functional Simulator for the POWER8 Processor is a software enablement tool that runs on a x86_64 Linux host system and allows users to develop and debug software for POWER8 processors. It simulates the architectural behavior of the system to test the features and functions of a software program developed for, or ported to, the POWER8 platform. It includes generalized simulation of the memory, disk, network, and system console. Some simulator configurations are extensible. They can be modified using Tool Command Language (Tcl) commands to produce customized run scripts for booting operating systems and loading and running user application code for debug or analysis.

Figure 1-1 shows the simulation stack. The simulator can be run as part of the software development kit (SDK) for PowerLinux, which is available at:

http://www-304.ibm.com/webapp/set2/sas/f/lopdiags/sdklop.html

It can also be run outside of this environment with the user interacting directly with the simulator. This document focuses on the latter usage method.

Figure 1-1. Simulator Stack for the POWER8 Processor



1.2 Installing and Invoking the Simulator

To install the simulator, first download the installation package from http://www-304.ibm.com/webapp/set2/sas/f/pwrfs/home.

After the package is downloaded, use the `rpm` command to install it:

```
rpm -ivh systemsim-p8-fc19-<version>.86_64.rpm
```

By default, the simulator is installed in the `/opt/ibm/systemsim-p8` directory. This directory is used in all the examples shown in this document.

The simulator is invoked by using the `power8` shell script, which is located in the `/opt/ibm/systemsim-p8/run/pegasus` directory.

When the simulator starts, it loads an initial run script, which typically configures and initializes the simulated machine. The name of the initial run script can be passed to the `power8` script with the `-f` option. When not specified on the command line, the simulator uses the `lib/pegasus/systemsim.tcl file`, which is provided as part of the systemsim-p8 release.

When specified using the `-f` option, the name of the initial run script can contain an absolute or relative path. The simulator searches for initial run scripts with a relative path by first looking in the current directory, and then in the `lib/pegasus` directory of the systemsim-p8 release. If the simulator fails to find the initial run script specified with the `-f` option, it issues an error message and exits.

It is generally the task of the initial run script to locate the operating system and filesystem images to be used by the simulated machine. For the POWER8 simulator, the default initial run script is `boot-linux.tcl`, which is installed in the `/opt/ibm/systemsim-p8/run/pegasus/linux` directory. The script searches for a Linux kernel named `vmlinux` and a filesystem image named `disk.img`. The script looks in the current directory. If it fails to find either of these images, it prints an error message and terminates the simulator.

Users who are working with systemsim-p8 in the SDK Eclipse development environment do not need to be concerned about this because the SDK handles it for them. Users running the simulator without the SDK must follow the instructions in `/opt/ibm/systemsim-p8/examples/kernel/README` to obtain vmlinux and disk.img. They then edit the `boot-linux.tcl` script to point to the location of these two files.

The following examples illustrate various ways to invoke the simulator. These examples assume that the simulator was installed into `/opt/ibm/systemsim-p8`.

Note: By default, the Tcl shell and command line interpreter do not support history recall. To add this feature, install the "rlwrap" package.

1. To run the simulator and boot Linux, change to the `run/pegasus/linux` directory and issue:

   ```
   ../power8 -f boot-linux.tcl
   ```

2. To run the simulator without a console window (`-n`) using the `run/pegasus/linux` directory and the user-created script, `myrun.tcl`, issue:

   ```
   ../power8 -n -f myrun.tcl
   ```

When the simulator starts, the window in which it was started becomes the simulator command window where you can enter simulator commands. The simulator also creates the console window (unless this was disabled with `-n`) which is initially labeled UART0 in the window's title bar.

1.3 Simulator Basics

1.3.1 Interacting with the Simulator

There are two ways to interact with the simulator:

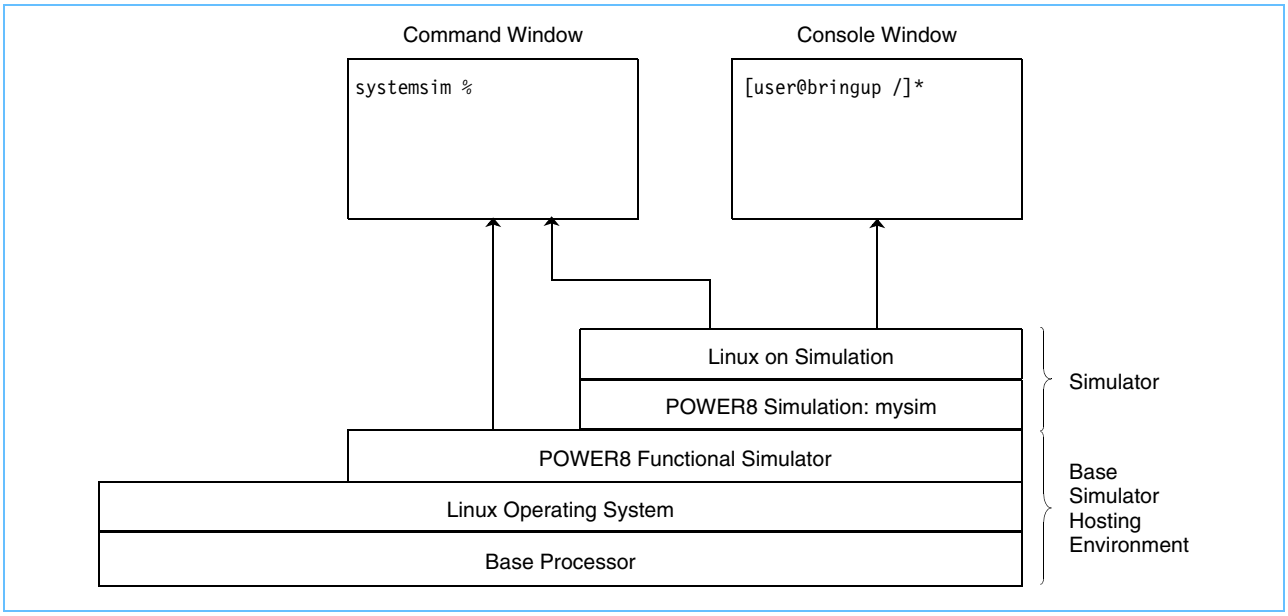
- Issuing commands to the simulated system
- Issuing commands to the simulator

The simulated system is the Linux environment on top of the simulated POWER8, where you run and debug programs. You interact with it by entering commands at the Linux command prompt in the console window. The console window is a Linux shell of the simulated Linux operating system.

You can also control the simulator itself, configuring it to do such tasks as setting breakpoints in code. These commands are entered at the simulator command line in the simulator command window.

Figure 1-2 on page 15 shows the simulator windows and the layers with which they communicate.

Figure 1-2. Simulator Structure



All simulator commands must be entered at the prompt in the command window (that is, in the window in which the simulator was started). *Table 1-1* shows some important commands.

Table 1-1. Basic Top-Level Commands for the POWER8 Functional Simulator (Page 1 of 2)

Simulator Command	Description
quit	Closes the simulation and exits the simulator.
help	Displays a list of the available simulator commands.
mysim go	Starts or continues the simulation. The first time the command is issued, the simulator boots the Linux operating system on the simulation.
mysim display memorymap	Displays the configured system memory regions, which can be adjusted with Tcl commands in the startup script.

Table 1-1. Basic Top-Level Commands for the POWER8 Functional Simulator (Page 2 of 2)

Simulator Command	Description
mysim cpu 0 help	Shows a list of the simulator commands available for each CPU. For example: `mysim cpu 0 step (number of instructions)`

The simulator prompt is displayed in the command window when the simulation is stopped or paused. When the simulation is running, the command window also displays a copy of the output to the console window and simulation-cycle information every few seconds, and the prompt is not available. To stop the simulation and get back the prompt, use the Ctrl-C key sequence. This stops the simulation, and the prompt reappears.

1.3.2 Operating-System Modes

A key attribute of the POWER8 Functional Simulator is its ability to boot and run a complete POWER8 system. By booting an operating system, such as Linux, the POWER8 Functional Simulator can execute many typical application programs that use standard operating system functions. Alternatively, applications can be run in standalone mode, in which all operating system functions are supplied by the simulator and normal operating system effects, such as paging and scheduling, do not occur. These two approaches to running applications on the simulator are referred to as Linux mode and standalone mode.

Linux Mode

In Linux mode, after the simulator is configured and loaded, the simulator boots the Linux operating system on the simulated system. At runtime, the operating system is simulated along with the running programs. The simulated operating system takes care of all the system calls, just as it would in a nonsimulation (real) environment.

Standalone Mode

In standalone mode, the application is loaded without an operating system. Standalone applications are user-mode applications that are normally run on an operating system. On a real system, these applications rely on the operating system to perform certain tasks, including loading the program, address translation, and system-call support. In standalone mode, the simulator provides some of this support, allowing applications to run without having to first boot an operating system on the simulator.

There are limitations that apply when building an application to be loaded and run by the simulator without an operating system. For example, applications must be linked statically with any libraries they require because the standard operating system shared libraries are not available in standalone mode. Another example is support for virtual memory address translation. Typically, the operating system provides address-translation support. Since an operating system is not present in standalone mode, the simulator loads executables without address translation, so that the effective address is the same as the real address. Therefore, all addresses referenced in the executable must be valid real addresses. If the simulator has been configured with 64 MB of memory, all addresses must fit in the range of x'0' to x'3FFFFFF'.

1.3.3 Obtaining vmlinux and Modifying the Root File System

The details of creating a sysroot disk file can be rather complicated. The sysroot disk provided for use with the IBM SDK for PowerLinux contains a minimal set of build tools and libraries to limit the size of the disk image file. This disk should be sufficient for running most simple applications, but in some cases users will want additional packages installed into the sysroot disk. To modify the contents of the sysroot disk:

1. Start the simulator with the `boot-linux.tcl` startup script.
2. Wait for the shell prompt to appear in the console window.
3. To copy normal files to the sysroot, use the `callthru` source command from the console window to copy the files into place from the host filesystem.
4. To install rpms, first copy the rpm file to the simulator file system with the `callthru` source. Then install the package with `rpm -Uvh <package>.rpm`. After the package is installed, you can delete the rpm file to preserve space in the simulated file system.
5. When all packages are installed, issue two consecutive `sync` commands to make sure that all changes are written to the sysroot disk file. For example:

   ```
   $ sync; sync
   ```

6. Exit the simulator.

2. Command Syntax and Usage

This chapter describes the POWER8 Functional Simulator command framework and introduces the structure, format, and usage of simulator commands. Topics in this chapter include:

- Understanding and Using Simulator Commands
- Defining and Managing a Simulated Machine
- Summary of Top-Level Simulator Commands

2.1 Understanding and Using Simulator Commands

The POWER8 Functional Simulator provides a unified, cross-platform application programming interface that enables users to easily set up the simulation environment, manage simulated architecture components, and write debugging and user application routines. The POWER8 Functional Simulator harnesses the power of Tcl/Tk to develop a simple and programmable text-oriented syntax that is easily extended. It minimizes the need for proprietary and difficult programming grammar and usage. By extending Tcl with exported functions, data types, and numerous predefined interfaces that are used for all interobject communication, the simulator provides a rapid, cross-platform development environment that enables users to quickly start working in the simulation environment.

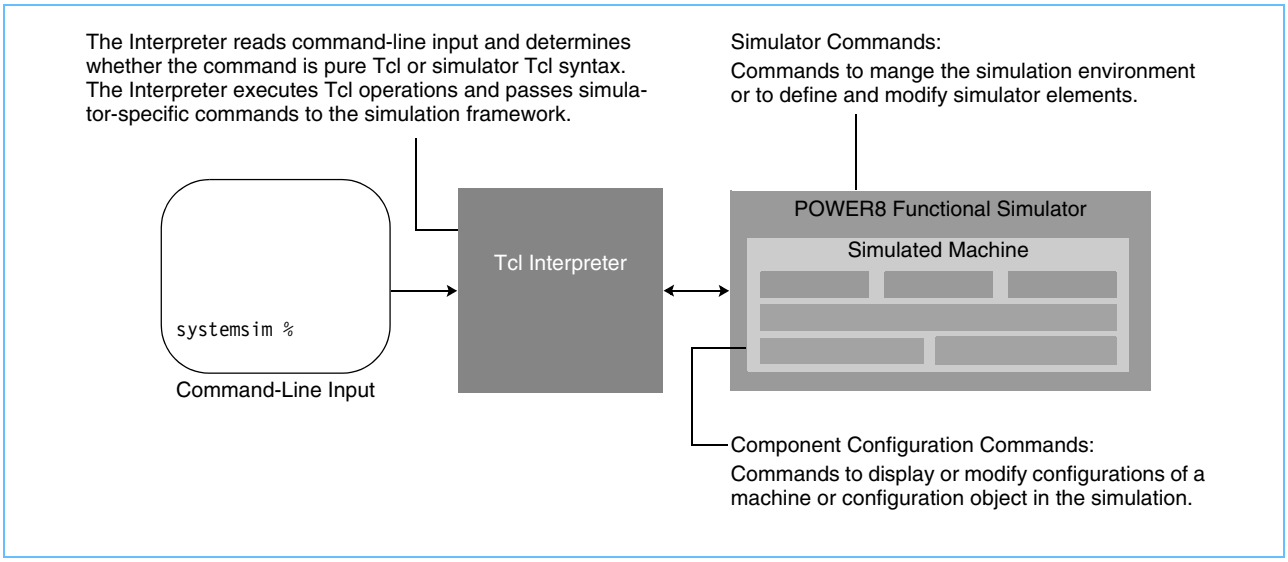
The POWER8 Functional Simulator command framework provides a set of commands for simulating processors. Each component in a system is configured by using commands that not only define the component's run-time behavior and characteristics, but govern its relationships and interactions with surrounding components in the system. The *POWER8 Functional Simulator Command Reference* provides syntax and usage information for Tcl/Tk commands that are used in the simulator environment.

Commands in the POWER8 Functional Simulator are organized into a hierarchy of operations based on the command function. At the top level, commands perform general sets of operations in the simulation environment, such as defining and displaying machine properties and system configurations, modifying configurable parameters, or managing the simulation environment.

The command-line interface can also be used to perform a number of operations on the simulator itself, such as to control a simulation, start debugger tools, and define and load virtual devices and disk images.

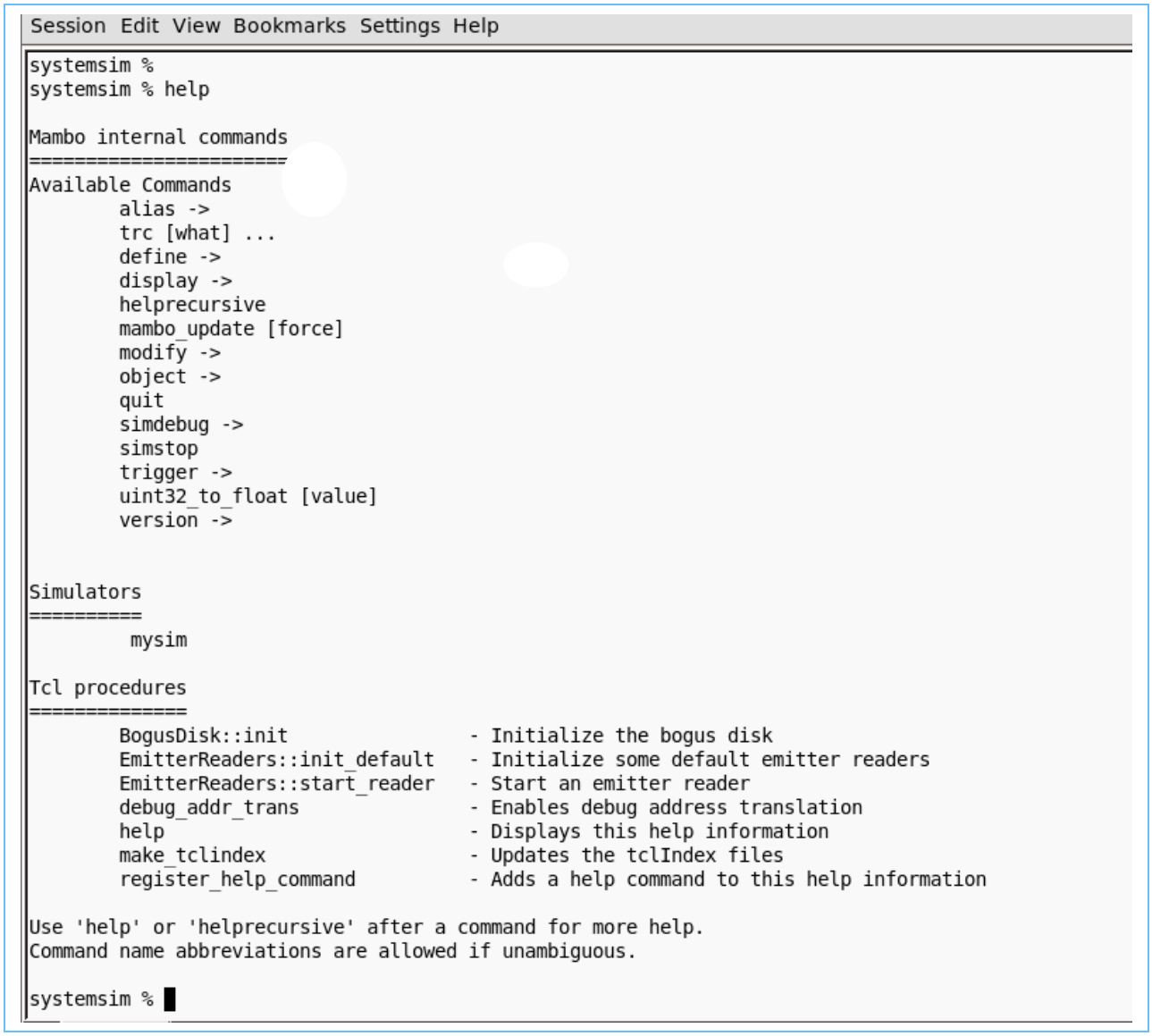
Figure 2-1 on page 20 illustrates how commands are processed in the simulation environment and describes the different categories of commands that are available.

Figure 2-1. Categories of Simulator Commands



Once the simulator is started, commands can be entered at the simulator command line or through simulation Tcl scripts. *Figure 2-2* illustrates the simulator command line at startup and the output that is displayed in response to typing "help" at the command prompt.

Figure 2-2. POWER8 Functional Simulator Command Line

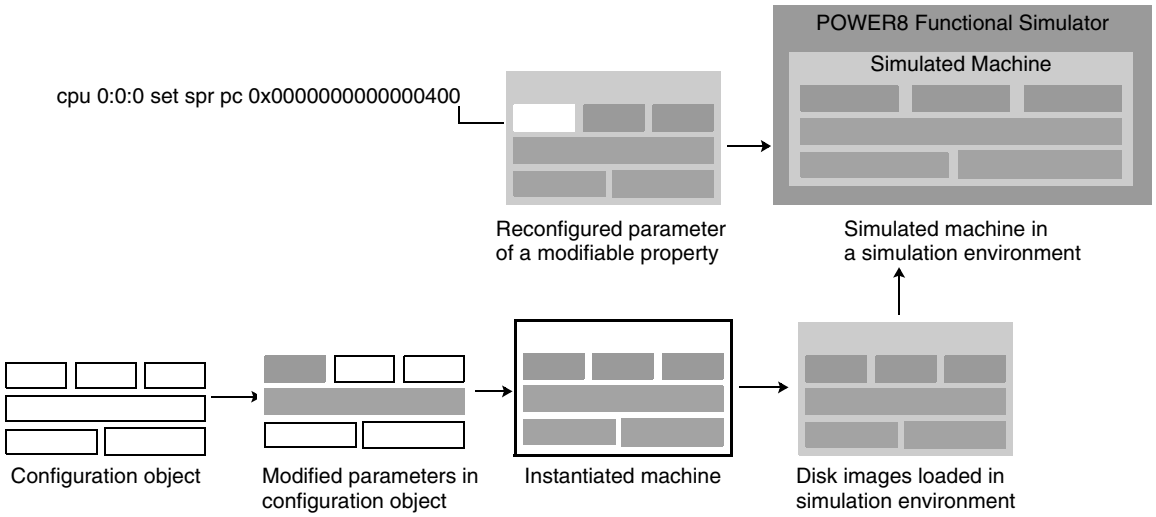


2.2 Defining and Managing a Simulated Machine

The POWER8 Functional Simulator delivers default configurations for the POWER8 processor that it is modeling. Using this configuration, users can instantiate a simulated machine based on a default configuration object to simulate the functionality of a baseline system.

Figure 2-3 on page 22 describes the general sequence of commands that is used to define a machine in the simulation environment.

Figure 2-3. Defining, Creating, and Starting a Simulated Machine



1. Create a configuration object for a machine type. The `define config` command is used to define a new machine type for which an empty configuration template is created, or the `define dup` command is used to duplicate an existing configuration based on predefined settings.
2. Modify configurable settings for the configuration object. The [*configuration_name*] `config` command is used to customize the default machine configuration by modifying mutable configuration properties.
3. Instantiate a machine based on the configuration object. The `define machine` [*machine_name*] command is used to create an instance of the machine type. At this point, most machine properties are fixed, although a subset of properties can be changed for the simulated machine (see step 6 below).
4. Load the Linux kernel and rootdisk image used by the simulated machine. The [*machine_name*] `load` and [*machine_name*} `bogus` commands are used to load disk images. Booting an operating system enables the execution of typical application programs that use standard operating system functions. The Linux operating system (running in the simulated environment) loads the application and is responsible for all operating system calls.
5. Start the simulation. The [*machine_name*] `go` command launches the simulator console window, which displays output of the simulated machine and allows users to configure and interact with the simulation.
6. Reconfigure a modifiable machine parameter. The [*machine_name*] `set` command is used to change values of machine properties that can be modified. When the simulation is started, the revised value is used in the simulation.

Commands to configure and initialize a simulated machine are typically provided to the simulator with a Tcl configuration and start-up file, called an initial run script, that is loaded when the simulator starts. The initial run script specifies commands to:

- Create a machine configuration and a machine instance using this configuration,
- Locate and load the operating system and file system image files,
- Prepare the machine to begin execution.

The name of the initial run script can be passed to systemsim-p8 with the `-f` option. If there is no initial run script specified when the simulator is started, the simulator uses the default initial run script provided with the release.

A typical initial run script

The specific commands in an initial run script vary slightly for various machine configurations, but all follow the basic procedure described previously. The following command sequence might be found in a typical initial run script:

1. Set up configuration and utility routines to be used later for booting an OS. These utilities reside in the `/opt/ibm/systemsim-p8/lib/common` directory. They are brought in with the `tcl` source command.

   ```
   source $env(LIB_DIR)/common/openfirmware_utils.tcl
   ```

2. The `lib/common` and `lib/pegasus` directories also contain standard configuration files that can be used in run scripts to create a simulator named `mysim` for a machine with the configuration `myconf`, based on the default POWER8 configuration.

   ```
   proc config_hook { conf } {
       $conf config processor/number_of_threads 1
       $conf config memory_size 2048M

   }

   source $env(LIB_DIR)/pegasus/systemsim.tcl

   source $env(RUN_DIR)/pegasus/p8-devtree.tcl
   ```

3. Specify a file containing the root filesystem (sysroot) image:

   ```
   # bogus disk
   mysim bogus disk init 0 disk.img rw
   ```

   The initial run script typically uses a standard search order to locate the `sysroot_disk` file starting with the current directory. The `rw` parameter indicates that the disk image has an access type of rw (for read-write), which indicates that modifications to the root filesystem during the simulation must be stored back into the `sysroot_disk` file. When the `sysroot_disk` is accessed read-write, the user must issue the `sync` command before exiting the simulator to ensure consistency of the filesystem image. If the parameter used is `cow` (copy on write), it indicates that changes will be stored in `<cowfile>`. This method treats the contents of the `sysroot_disk` file as read-only, so that subsequent simulations can be performed with repeatable results.

4. Set up the simulated network (for more information, see *Section 3 Accessing the Host Environment* on page 27):

   ```
   # networking with IRQ off
   # need this locally:
   #   sudo tunctl -u $USER -t tap0
   #   sudo ifconfig tap0 172.19.98.108 netmask 255.255.255.254
   # in sim this:
   #   ifconfig eth0 172.19.98.109 netmask 255.255.255.254
   mysim bogus net init 0 d0:d0:d0:da:da:da tap0 0 0
   ```

5. Load the operating system kernel into memory:

   ```
   mysim load vmlinux <path_to_vmlinux_file> 0x0
   ```

In hardware, this is generally performed by the system firmware. However, the simulator is typically configured without firmware installed. Therefore, a simulator command is used to load the kernel into memory.

In many cases, the initial run script uses a standard search order to locate the vmlinux file, starting with the current directory and then the images directory under the simulator root directory. The 0x0 parameter in this command specifies the address at which to load the kernel file:

```
# Load vmlinux
mysim load vmlinux vmlinux 0x0
```

6. Start the simulation with the `mysim go` command:

```
mysim mode fastest
mysim go
```

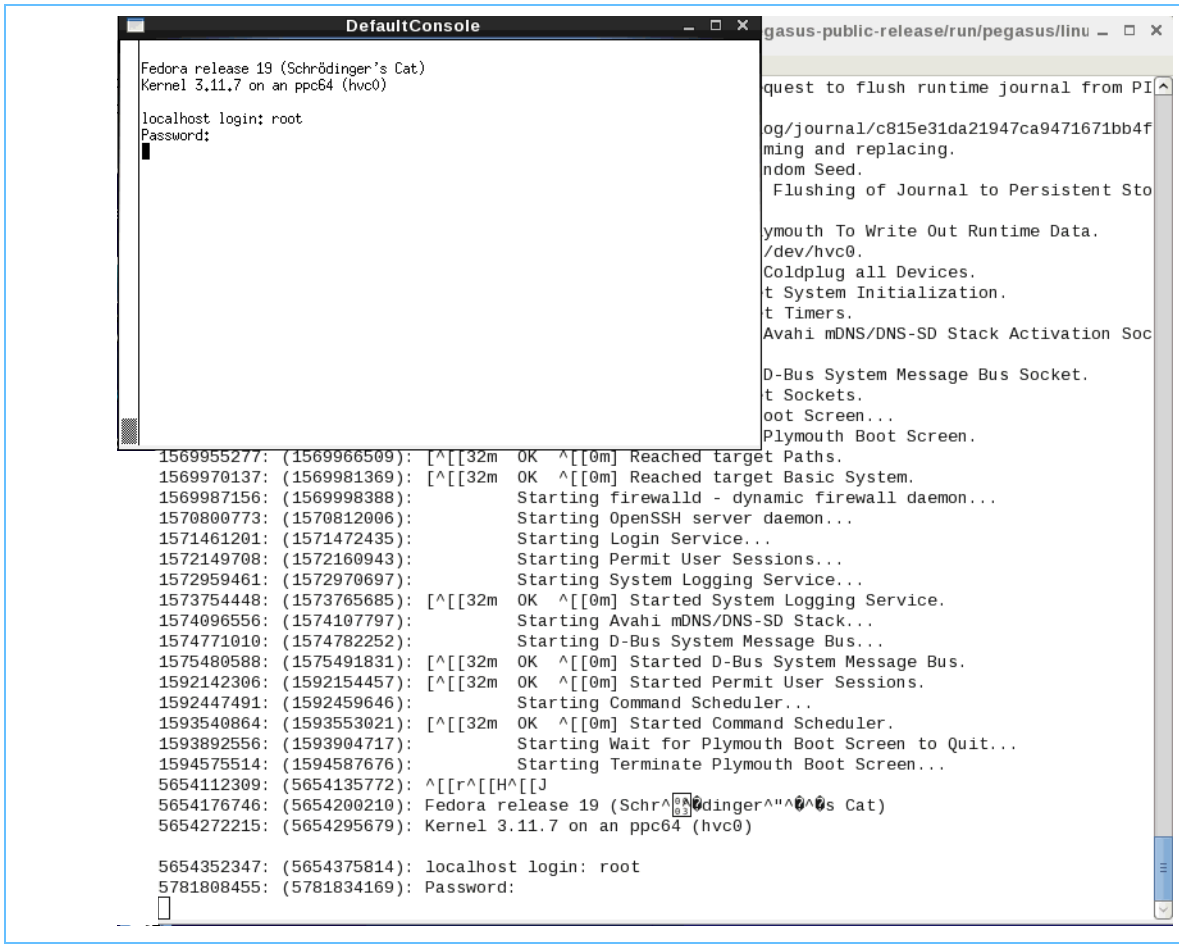
7. After the operating system has completed boot, you can execute POWER8 applications by entering commands at the Linux console (see *Figure 2-4* on page 25). To automate console input from within a script, use the `mysim console create` command. This command automates the interactions that are typically performed by manually typing commands in the simulator console window:

```
mysim console create input in string <console_input>
```

where `console_input` specifies a string containing console commands to execute. The string contents are identical to any commands that are typed in the console window, including new lines (which can be entered with the escape sequence `\n`). Typically, the last command of the console input is `callthru exit` to return control to the simulation Tcl command script. For example:

```
mysim console create UART0 in string "\n"
mysim console create UART0 in string "ls -l /"
mysim console create UART0 in string "cat /proc/cpuinfo"
mysim console create UART0 in string "callthru exit"
```

Figure 2-4. Linux Console and Command Line Screen



2.3 Summary of Top-Level Simulator Commands

Table 2-1 summarizes the functionality of selected top-level commands that are used to define, modify, and use the simulator. The *POWER8 Functional Simulator Command Reference* provides the complete command-line syntax and usage of each command or class of commands.

Table 2-1. Top-Level Commands for the POWER8 Functional Simulator (Page 1 of 2)

Command	Command Summary
alias	Assigns a user-specified personal shorthand for a command string. The `alias` command allows users to call a small, more familiar, command or name to execute long or complex command strings.
define	Defines settings for a configuration object. The `define` command also provides the ability to duplicate configurations from a predefined machine type, instantiate a machine based on a configuration object, and enumerate a list of machines that are active in the simulation.
display	Displays system-wide information about configurations, machines, instruction settings, and warning levels. The `display` command is especially useful to determine properties that are configured for machines that are currently available in a simulation.

Table 2-1. Top-Level Commands for the POWER8 Functional Simulator (Page 2 of 2)

Command	Command Summary
help or helprecursive	Displays a listing the POWER8 Functional Simulator commands. The `helprecursive` command displays a comprehensive command tree that hierarchically lists syntax and input parameters for all available commands.
object	Provides the ability to interrogate information from one or more executable files to examine low-level execution details.
quit	Ends the current simulation and exits to the operating-system command line.
simdebug	Provides low-level tracing capabilities that are useful for debugging functionality or performance issues in the simulated system. **Note:** The `simdebug` command might be disabled for certain distributions.
version	Displays the version number of simulation system components, the date and timestamp of the installed POWER8 Functional Simulator build, and compile-time flags that are enabled in the build.

mysim	Simulator commands are prefaced with mysim. To see what commands are available for the simulator, type:
	`mysim help`.
	Examples include:
	`cycle (number of cycles to execute)`
	`load (load binary files)`
	`memory (display/write memory)`
	To learn more, type help after the command. For example:
	`mysim load help`

At any time, users can type the help command at the command line to retrieve a list of command choices that are available from that point in the syntax statement. In most cases, you can also just type a partial command sequence and hit return. For example, at the top level, help displays a list of top-level commands. An arrow indicates that a subsequent level of command functionality is available for this command.

3. Accessing the Host Environment

This chapter describes several mechanisms that are provided to allow interactions between the host and simulated systems. Topics in this chapter include:

- The Callthru Utility
- Bogus Network Support
- Bogus Disk Driver Support

3.1 The Callthru Utility

The callthru utility allows you to copy files between the host system and the simulated system while it is running. This utility runs within the simulated system and accesses files in the host system using special call-through functions of the simulator. The callthru utility is installed as a binary application in the simulator system in the `/opt/ibm/systemsim-p8/examples` directory. The callthru utility supports the following options:

- To write standard input into `<filename>` on the host system, issue:

  ```
  callthru sink <filename>
  ```

- To write the contents of `<filename>` on the host system to standard output, issue:

  ```
  callthru source <filename>
  ```

Redirecting appropriately lets you copy files between the host and the simulated system. For example, to copy the `/tmp/ matrix_mul` application from the host into the simulated system and then run it, issue the following commands in the console window of the simulated system:

```
callthru source /tmp/matrix_mul > matrix_mul
chmod +x matrix_mul
./matrix_mul
```

Another commonly used feature of the callthru utility is the exit option, which stops the simulation and is initiated by the callthru utility inside the simulator. This is especially useful for constructing "scripted" executions of the simulator that involve alternating steps in the simulator and the simulated system.

- To stop the simulator and return control back to the currently active run script, issue:

  ```
  callthru sink <filename>
  ```

3.2 Bogus Network Support

Bogus network support was developed to enable network communications with reasonable performance between the simulated system and other systems. This is accomplished by using a special Ethernet device that uses call-through functions of the simulator to send and receive network packets to the host system. To enable communication with other systems, the host system must be configured to relay packets from the simulated system out to the real network.

The bogus network facility can be configured and used in a variety of ways. A detailed description of the Linux and systemsim-p8 commands to set up and manage bogus network communications follows. For user convenience, the most common approach to using the bogus network has been automated using Tcl procedures. These are described first because most users should find these sufficient for simple network communication between the host and the simulated system.

3.2.1 Extended Description of Bogus Network Support

There are three key components to bogus network communications:

1. A facility on the host system that provides systemsim-p8 with a path to the network. The TUN/TAP support available for Linux is a good choice for this component, and we assume TUN/TAP in the remainder of this description.
2. The systemsim-p8 support for the bogus network. This support is not enabled by default. Simulator commands are used to enable the bogus network support.
3. An operating system (OS) kernel with a bogus network driver.

3.2.2 Setting up TUN/TAP on the Host System

You must have root privileges on your system to set up bogus network operation. Execute the following commands:

```
sudo tunctl -u $USER -t tap0
sudo ifconfig tap0 172.19.98.108 netmask 255.255.255.254
```

3.2.3 Configuring systemsim-p8 Support for the Bogus Network

To enable bogus network support, issue simulator commands that configure and initialize the bogus network. These commands must be issued before booting the Linux kernel on the simulator so that Linux recognizes the bogus network device during its boot process. The general form of the command to initialize the bogus network is:

```
mysim bogus net init 0 <mac address> <interface name> <irq>
```

The `<mac address>` parameter is the media access control (MAC) hardware address that you want the emulated Ethernet to use. It must be unique on your network (that is, not used by any other emulated hosts or by any host network adapter). The `<interface name>` parameter is the name of the interface to be used, typically "tap0". The `<irq>` parameter specifies the interrupt request queue ID to be used by the bogus network device; use 0 0 for the POWER8 Functional Simulator.

3.2.4 The Bogus Network Device Driver

The final component required for bogus network communication is an OS kernel with the bogus network device driver. Binary patches for the bogus network and disk device drivers can be downloaded from the unicamp file FTP site. Instructions for downloading patches, along with pointers for kernel binary download, can be found in the readme file in the `/opt/ibm/systemsim-p8/examples/kernel` directory.

3.2.5 Connecting to the Simulation Host

Once these components are in place, you are ready to use the bogus network for network communications with the simulated system. Start the simulation. At the Linux prompt, enter the following commands on the simulated console (UART):

```
% mount /proc
% ifconfig eth0 172.19.98.109 netmask 255.255.254.0
```

You can then ping the host system from the simulated system (and vice versa)

```
% ping -c 1 172.19.98.108
```

3.2.6 Troubleshooting the Bogus Network

Ping of system host from simulated host results in ping icmp open socket: Operation not permitted:

First, check to see if you can ping the simulated host from the system host. If this is working, the bogus network traffic is flowing in both directions and the problem is probably with the kernel or root disk. most likely the latter.

operations seems to hang:

Beware of firewalls, ipchains, and iptables. A firewall of some sort is probably blocking the port. Either disable the firewall or open up the specific port.

3.3 Bogus Disk Driver Support

To enable bogus disk support, issue simulator commands that configure and initialize the bogus disk system. These commands must be issued before booting the Linux kernel on the simulator so that Linux recognizes the bogus disk device during its boot process. The general form of the command to initialize bogus disk is:

```
mysim bogus disk init 0 disk.img rw
```

This configures the disk support for read/write operations.

You can find examples of how to configure both bogus disk and bogus network support in the example Linux boot script, `boot-linux.tcl`.