EXAM PREP

Your Complete Certification Solution

Exam CX-310-200

Solaris 10 System Administration Part I

CD Features ExamGear Practice Questions!

Bill Calkins

Solaris 10 System Administration Exam Prep (Exam CX-310-200), Part I

ISBN-13: 978-0-7897-3790-8

ISBN-10: 0-7897-3790-6

Library of Congress Cataloging-in-Publication Data:

Calkins, Bill.

Solaris 10 system administration exam prep (Exam CX-310-200) / Bill

Calkins.

p. cm.

ISBN 978-0-7897-3790-8 (pbk. w/cd)

1. Electronic data processing personnel--Certification. 2. Operating

systems (Computers)--Examinations--Study guides. 3. Solaris (Computer

file) I. Title.

QA76.3.C34346 2008

005.4'32--dc22

2008031592

Printed in the United States on America

First Printing: September 2008

Introduction

Bill Calkins has been training Solaris system administrators for more than 15 years. This book contains the training material he uses in his basic and advanced Solaris administration courses. Over the years, this material has helped thousands of Solaris administrators become certified. This is our first edition of the *Solaris 10 System Administration Exam Prep*. It began with the *Training Guide* for Solaris 2.6, 7, 8, and 9 and is now the *Exam Prep* for Solaris 10. Instructors from universities and training organizations around the world have used this book as courseware in their Solaris administration courses. In addition, administrators from around the world have used this book as a self-study guide when instruction from a Sun training center is either unavailable or not within their budget. Many people have written with their success stories, suggestions, and comments. Their suggestions are what keep making this guide more valuable.

The SCSA Solaris 10 OS CX-310-200 and CX-310-202 *Exam Prep* books provide training materials for anyone who is interested in becoming a Sun Certified System Administrator for Solaris 10. When used as a study guide, these two books will save you a great deal of time and effort searching for information you will need to know when taking the exam. Each book covers the exam objectives in enough detail for the inexperienced administrator to learn the objectives and apply the knowledge to real-life scenarios. Experienced readers will find the material in these books to be complete and concise, making it a valuable study guide for the Sun Certified System Administrator (SCSA) exams.

This book is not a cheat sheet or cram session for the exam; it is a training manual. In other words, it does not merely give answers to the questions you will be asked on the exam. We have made certain that this book addresses the exam objectives in detail, from start to finish. If you are unsure about the objectives on the exams, this book teaches you what you need to know. After reading each chapter, assess your knowledge of the material covered using the review questions at the end of the chapter. When you have completed reading a section, use the practice exam at the end of the book and the ExamGear test engine on the CD-ROM to assess your knowledge of the objectives covered on each exam. This CD-ROM contains sample questions similar to what you are likely to see on the real exam. More sample questions are available at http://www.UnixEd.com, so be sure to visit this site to find additional training and study materials.

How This Book Helps You

This book teaches you how to administer the Solaris 10 operating system. It offers you a self-guided training course for all the areas covered on the CX-310-200 certification exam by showing you how to install, configure, and administer the Solaris 10 operating environment. You will learn the specific skills that are required to administer a system and to pass the first part of the Sun Certified System Administrator exam for Solaris 10 (CX-210-200). Experienced administrators who are upgrading an existing Solaris certification will find in-depth coverage of the new topics that they will need to learn for the CX-310-203 upgrade exam in both the SCSA Solaris 10 OS CX-310-200 and CX-310-202 *Exam Prep* books.

Throughout the book, we provide helpful tips and real-world examples that we have encountered as system administrators. In addition, we provide useful real-world exercises to help you practice the material you have learned. The following list describes this book's setup:

- **Organization**: This book is organized according to the individual exam objectives. Every objective you need to know for installing, configuring, and administering a Solaris 10 system is in this book. We have attempted to present the objectives in an order that is as close as possible to that listed by Sun. However, we have not hesitated to reorganize them as needed to make the material as easy as possible for you to learn. We have also attempted to make the information accessible in the following ways:
  - This Introduction includes the full list of exam topics and objectives.
  - Read the "Study and Exam Prep Tips" element early on to help you develop study strategies. This element provides valuable exam-day tips and information on exam/question formats.
  - Each chapter begins with a list of the objectives to be covered, exactly as they are defined by Sun. Throughout each section, material that is directly related to the exam objectives is identified.
  - Each chapter also begins with an outline that provides you with an overview of the material and the page numbers where particular topics can be found.
- **Instructional features**: This book is designed to provide you with multiple ways to learn and reinforce the exam material. The following are some of the helpful methods:
  - **Objective explanations**: As mentioned, each chapter begins with a list of the objectives covered in the chapter. In addition, immediately following each objective is an explanation in a context that defines it more meaningfully.
  - **Study Strategies**: The beginning of each chapter also includes strategies for studying and retaining the material in the chapter, particularly as it is addressed on the exam.

  - **Exam Alerts**: Throughout each chapter you'll find exam tips that help you prepare for exam day. These tips were written by those who have already taken the Solaris 10 certification exams.
  - **Notes**: Notes contain various types of useful information, such as tips on technology or administrative practices, historical background on terms and technologies, or side commentary on industry issues.
  - **Cautions**: When you use sophisticated information technology, mistakes or even catastrophes are always possible because of improper application of the technology. Cautions alert you to such potential problems.
  - **Step by Steps**: These are hands-on lab excercises that walk you through a particular task or function relevant to the exam objectives.
  - **Key Terms**: A list of key terms appears near the end of each chapter.
  - **Exercises**: Found at the end of the chapters in the "Summary" section, the exercises are performance-based opportunities for you to learn and assess your knowledge.
  - **Suggested Readings and Resources**: At the end of each chapter is a list of additional resources that you can use if you are interested in going beyond the objectives and learning more about the topics presented in the chapter.
- **Extensive practice test options**: The book provides numerous opportunities for you to assess your knowledge and practice for the exam. The practice options include the following:
  - **Exam Questions**: These questions appear in the "Summary" section at the end of each chapter. They allow you to quickly assess your comprehension of what you just read. Answers to the questions are provided in the section "Answers to Exam Questions."
  - **Practice Exam**: A practice exam is included in Part II of this book, "Final Review," (as discussed in a moment).
  - **ExamGear**: The ExamGear software included on the CD-ROM provides further practice questions.

NOTE

ExamGear software For a complete description of the ExamGear test engine, see Appendix A, "What's on the CD-ROM."

- **Final Review**: This part of the book gives you two valuable tools for preparing for the exam:
  - **Fast Facts**: This condensed version of the information contained in the book will prove extremely useful for last-minute review.
  - **Practice Exam**: A full practice exam is included, with questions written in styles similar to those used on the actual exam. Use the practice exam to assess your readiness for the real exam.
- **Appendix and glossary**: This book also contains a glossary and a description of what is on the CD-ROM (Appendix A).

These and all the other book features will enable you to thoroughly prepare for the exam.

Conventions Used in This Book

- **Commands**: In the text, steps, and examples, the commands you type are displayed in a special `monospace` font.
- **Arguments and options**: In command syntax, command options and arguments are enclosed in < >. (The italicized words within the < > symbols stand for what you actually type. You don't type the < >.)

  ```
  lp -d<printer name> <filename> <return>
  ```

- **Using the mouse**: When using menus and windows, you select items using the mouse. Here is the default mapping for a three-button mouse:

  Left button: Select

  Middle button: Transfer/adjust

  Right button: Menu

  The Select button is used to select objects and activate controls. The middle mouse button is configured for either Transfer or Adjust. By default, it is set up for Transfer, which means that it is used to drag or drop list or text items. You use the left mouse button to highlight text, and then you use the middle button to move the text to another window or to reissue a command. The middle button can also be used to move around windows on the screen. The right mouse button, the Menu button, is used to display and choose options from pop-up menus.

- **Menu options**: The names of menus and the options that appear on them are separated by a comma. For example, "Select File, Open" means to pull down the File menu and choose the Open option.
- **Code-continuation character**: When a line of code is too long to fit on one line of a page, it is broken and continued to the next line. The continuation is preceded by a backslash.

Audience

This book is designed for anyone who has a basic understanding of UNIX and wants to learn more about Solaris system administration. Whether or not you plan to become certified, this book is the starting point to becoming a Solaris system administrator. It's the same training material that Bill uses in his Solaris 10 Intermediate System Administration course. This book covers the basics as well as intermediate system administration topics you need to know before you begin administering the Solaris operating system. Our goal was to present the material in an easy-to-follow format, with text that is easy to read and understand. The only prerequisite is that you have used UNIX, that you have attended a fundamentals of UNIX class for users, or that you have studied equivalent material so that you understand basic UNIX commands and syntax. Before you begin administering Solaris, it's important that you have actually used UNIX.

This book is also intended for experienced system administrators who want to become certified, update their current Solaris certification, or simply learn about the features of the Solaris 10 operating environment. To pass the certification exams, you need a solid understanding of the fundamentals of administering Solaris. This book will help you review the fundamentals required to pass the certification exams.

The Sun Certified System Administrator Exams

To become a Sun Certified System Administrator, you need to pass two exams: CX-310-200 (Part I) and CX-310-202 (Part II). This book covers the Part I exam, which is a prerequisite for Part II. You will not receive a certificate until you have passed both examinations. Also, if you are already certified in Solaris 2.6, 7, 8, or 9, you need to use materials found in volumes 1 and 2 of this series to take the upgrade exam, CX-310-203, to become certified on Solaris 10.

Beware of fakes. We have seen some websites that promote their own certification programs, so be sure to evaluate them carefully. Certification programs promoted by these sites are not the same as the Sun certification program, and you will not receive a certificate from Sun until you pass Sun's exams from a certified Sun testing center. Go to my website (http://www.UnixEd.com) for links to the real exams and information on Sun's certification program if you are in doubt. In addition, feel free to visit our online Solaris certification discussion forum at http://www.UnixEd.com, where you can ask me questions directly.

Exam CX-310-200

The following sections cover the Exam CX-310-200 objectives.

Manage File Systems

List the different types of file systems and file types in the Solaris operating environment. Understand how to add disk devices to a system and the device files associated with each disk on each Solaris platform (SPARC and x86/x64). Understand how to use the `format` and `fdisk` utilities. Understand how to create, mount, and repair file systems. Understand all the configuration files associated with managing file systems.

Install Software

Describe the methods used and the sequence of steps required to perform the Solaris 10 operating environment software installation on SPARC-, x64-, and x86-based systems. Identify the function of the package administration commands. Understand how to install, verify, and remove operating system patches.

Perform System Boot Procedures

Understand the entire boot process, with knowledge of the various configuration files and startup scripts on SPARC-, x64-, and x86-based systems. Understand how to use and execute boot PROM commands. Understand the role of the Solaris Management Facility (SMF) in the boot process, and become familiar with SMF-related commands. Understand the function of the files or directories accessed during the boot process. Understand the commands used to change the run level of a system to a specified state.

Perform User and Security Administration

Understand all aspects of administering users and groups. Understand how to set and verify file and directory permissions.

Manage Network Printers and System Processes

Describe the purpose, features, and functionality of the print management tools available in the Solaris operating environment. Understand the LP print service directory structure and the Solaris operating environment printing process. Understand the commands that display information for all active processes on the system. Understand the effect of sending a specified signal to a process. Understand the various methods used to terminate an active process.

Perform System Backups and Restores

Understand the functional capabilities of the various backup, archive, and restore utilities in Solaris 10. Identify the commands and steps required to back up and restore a file system. Given a specific scenario, be prepared to develop a strategy for scheduled backups and back up an unmounted file system using the appropriate commands.

Exam CX-310-203 (Solaris 10 Upgrade Exam)

If you're already certified on Solaris 2.6, 7, 8, or 9, you only need to take the CX-310-203 upgrade exam to update your certification. Here are the current objectives for that exam:

- Install Software
- Manage File Systems
- Perform System Boot and Shutdown Procedures for SPARC-, x64-, and x86-Based Systems
- Perform User and Security Administration
- Perform Advanced Installation Procedures

Summary

It’s not uncommon for Sun to change the exam objectives or to shift them around after the exams have been published. I highly recommend that before you start this book, you visit my website at http://www.UnixEd.com to get the most up-to-date list of exam objectives, the errata for this book, up-to-date sample exam questions, and any other last-minute notes about these exams. We will provide all the information you need to pass the exam; all you need to do is devote the time. Learning the objectives is the first step; the next step is to practice. You need to have access to both SPARC- and x86/x64-based systems running Solaris 10 so that you can practice what you have learned. Unless you have a supernatural memory, it will be difficult to pass the exams without practice.

In the back of this book is the ExamGear software test CD that prepares you for the questions you might see on the exam. The CD-ROM-based test engine was designed by educational experts to help you learn as you test. It is a preview of the types of questions to expect on the exams, and it tests your knowledge on all the exam objectives. If you are weak in any area, the sample questions will help you identify that area so that you can go back to the appropriate chapter and study the topic. Each question on the CD-ROM has a flash card to help you in case you get stuck. This flash card contains brief, concise, textbook excerpts that explain why each answer is correct so that you can learn as you test.

Also, for an additional cost, you can purchase more questions for the ExamGear test engine from our website. You’ll obtain hundreds of questions that will take you deep into each exam objective, providing a comprehensive skills assessment and helping you evaluate your readiness and retention of the materials.

Advice on Taking the Exam

More extensive tips are found in the “Study and Exam Prep Tips” element and throughout the book, but keep in mind the following advice as you study for the exam:

- **Read all the material**: This book includes information not reflected in the exam objectives to better prepare you for the exam and for real-world experiences. Read all the material to benefit from this.
- **Do the step-by-step lab exercises, and complete the exercises in each chapter**: This will help you gain experience and prepare for the scenario-type questions that you will encounter.
- **Use the questions to assess your knowledge**: Each chapter contains exam questions. Use these to asses your knowledge and determine where you need to review material.

- **Review the exam objectives**: Develop your own questions and examples for each topic listed. If you can develop and answer several questions for each topic, you should not find it difficult to pass the exam.
- **Relax and sleep before taking the exam**: Your time for taking the exam is limited. However, if you have prepared and you know Solaris network administration, you will find plenty of time to answer all the questions. Be sure to rest well the night before so that you can handle the stress that time limitations put on you as you take the exam.
- **Review all the material in the "Fast Facts" element the night before or the morning you take the exam.**
- **If you don't know the answer to a question, just skip it and don't waste much time**: You need to complete the exam in the time allotted. Don't be lazy during the exam; answer all the questions as quickly as possible. Any unfinished questions will be marked as incorrect.
- **Visit my website, http://www.UnixEd.com. It contains the following:**
    - Late-breaking changes that Sun might make to the exam or the objectives. You can expect Sun to change the exams frequently. Make sure you check my website before taking the exam.
    - A FAQs page with frequently asked questions and errata about this book or the exams.
    - Links to other informative websites.
    - Additional practice questions and sample exams for the ExamGear test engine. The ExamGear test engine has hundreds of questions that you can use to further assess your retention of the material presented in the book. The exams feature electronic flash cards that take the place of those sticky notes that you've used as bookmarks throughout the book. Don't attempt the real exam until you can pass every section of the practice exams with a 95% or better score.
    - An online forum where you can discuss certification-related issues with me and other system administrators, including some who have already taken the exam.
    - Additional study materials, training programs, and online seminars related to Solaris certification.
    - You can also email me directly from this website with questions or comments about this book. I always try to answer each one.

When you feel confident, take the real exams and become certified. Don't forget to drop me an email and let me know how you did (guru@UnixEd.com).

5

CHAPTER FIVE

Managing System Processes

Objectives

The following test objectives for Exam CX-310-200 are covered in this chapter:

Explain how to view system processes and clear frozen processes.

- ▶ Managing system processes is a common task for any system administrator. You should know how to use the commands that display information for all active processes on the system, and how to terminate an active or deadlocked process.

Explain how to schedule an automatic one-time execution of a command and the automatic recurring execution of a command.

- ▶ Many processes compete for execution time so scheduling jobs to run at off-peak hours can dramatically improve system performance. The system administrator needs to understand how to use the Solaris batch processor to schedule execution of commands.

Outline

Study Strategies

The following study strategies will help you prepare for the test:

- ▶ Understand each of the commands in this chapter enough so that you can match the command and option with a description. Practice them on a Solaris system so that you can become familiar with the output they produce.
- ▶ Know all the commands used to display information about a process. When viewing processes, understand each of the fields that are displayed in the output.
- ▶ Finally, understand how to schedule commands via the Solaris batch-processing facilities. Become familiar with all of the associated configuration files: what they do and how they are formatted.

EXAM ALERT

Managing System Processes As of this writing, the topic of managing system processes is covered lightly on the CX-310-200 exam. This could change in the future as Sun keeps updating and changing its exams. The best approach is to be prepared and learn the material thoroughly. After all, it's a topic every system administrator needs to know to effectively perform the job.

Introduction

This chapter covers Solaris processes—how to view processes, understand the effects signals have on processes, and how to manage processes.

Viewing a Process

Objective:

Explain how to view system processes.

Solaris is a multitasking environment in which a number of programs run at the same time. This means that many users can be active on the system at the same time, running many jobs (processes) simultaneously. Each Solaris program can start and stop multiple processes while it is running, but only one job is active per processor at any given time while the other jobs wait in a job queue. Because each process takes its turn running in very short time slices (much less than a second each), multitasking operating systems give the appearance that multiple processes are running at the same time. A parent process forks a child process, which, in turn, can fork other processes.

NOTE

Forks The term *fork* is used to describe a process started from another process. As with a fork in the road, one process turns into two. You'll also see the term *spawn* used—the two words are interchangeable for the purposes of this subject.

A program can be made up of many processes. A *process* is part of a program running in its own address space. A process under Solaris consists of an *address space* and a set of data structures in the *kernel* to keep track of that process. The address space is divided into various sections that include the instructions that the process may execute, memory allocated during the execution of the process, the *stack*, and memory-mapped files. The kernel must keep track of the following data for each process on the system:

- Address space
- Current status of the process
- Execution priority of the process
- Resource usage of the process
- Current signal mask
- Ownership of the process

A process is distinct from a job, command, or program that can be composed of many processes working together to perform a specific task. For example, a computer-aided design application is a single program. When this program starts, it spawns other processes as it runs. When a user logs in to the program, it spawns yet other processes. Each process has a process ID associated with it and is referred to as a *PID*. You can monitor processes that are currently executing by using one of the commands listed in Table 5.1.

TABLE 5.1 Commands to Display Processes

Command	Description
`mpstat`	Executed from the command line, `mpstat` reports processor statistics in tabular form. Each row of the table represents the activity of one processor.
`ps`	Executed from the command line to display information about active processes.
`pgrep`	Executed from the command line to find processes by a specific name or attribute.
`prstat`	Executed from the command line to display information about active processes on the system.
`ptree`	Prints the process trees containing the specified pids or users, with child processes indented from their respective parent processes.
`sdtprocess`	A GUI used to display and control processes on a system. This utility requires the X Window System (also known as X Windows).
`SMC process tool`	A GUI available in the Solaris Management Console used to monitor and manage processes on a system.
`pargs`	Executed from the command line to examine the arguments and environment variables of a process.
`svcs`	With the `-p` option, this Service Management Facility command will list processes associated with each service instance.
`time`	Time a simple command.

Before getting into the commands used to monitor processes, you first need to become familiar with process attributes. A process has certain attributes that directly affect execution. These are listed in Table 5.2.

TABLE 5.2 Process Attributes

Attribute	Description
`PID`	The process identification (a unique number that defines the process within the kernel)
`PPID`	The parent PID (the parent of the process)
`UID`	The user ID number of the user who owns the process
`EUID`	The effective user ID of the process
`GID`	The group ID of the user who owns the process
`EGID`	The effective group ID that owns the process
`Priority`	The priority at which the process runs

Use the `ps` command to view processes currently running on the system. Use the `ps` command when you're on a character-based terminal and don't have access to a graphical display. Adding the `-l` option to the `ps` command displays a variety of other information about the processes currently running, including the state of each process (listed under `S`). The codes used to show the various process states are listed in Table 5.3.

TABLE 5.3 Process States

Code	Process State	Description
O	Running	The process is running on a processor.
S	Sleeping	The process is waiting for an event to complete.
R	Runnable	The process is on the run queue.
Z	Zombie state	The process was terminated and the parent is not waiting.
T	Traced	The process was stopped by a signal because the parent is tracing it.
W	Waiting	The process is waiting for CPU usage to drop to the CPU-caps enforced limits

To see all the processes that are running on a system, type the following:

```
ps -el
```

The system responds with the following output:

```
# ps -el

F  S  UID   PID  PPID C PRI NI  ADDR SZ   WCHAN TTY  TIME CMD
19 T   0      0     0  0   0 SY   ?     0         ?   0:18 sched
 8 S   0      1     0  0  40 20   ?   150       ? ?   0:00 init
19 S   0      2     0  0   0 SY   ?     0       ? ?   0:00 pageout
19 S   0      3     0  0   0 SY   ?     0       ? ?   0:01 fsflush
 8 S   0    309     1  0  40 20   ?   217       ? ?   0:00 sac
 8 S   0    315     1  0  40 20   ?   331       ? ?   0:00 sshd
 8 S   0    143     1  0  40 20   ?   273       ? ?   0:00 rpcbind
 8 S   0     51     1  0  40 20   ?   268       ? ?   0:00 sysevent
 8 S   0     61     1  0  40 20   ?   343       ? ?   0:01 picld
 8 S   0    453   403  0  50 20   ?  1106       ? ?   0:00 dtfile
 8 S   0    189     1  0  40 20   ?   509       ? ?   0:00 automoun
 8 S   0    165     1  0  40 20   ?   292       ? ?   0:00 inetd
 8 S   0    200     1  0  40 20   ?   415       ? ?   0:00 syslogd
 8 S   0    180     1  0  40 20   ?   266       ? ?   0:00 lockd
 8 S   0    219     1  0  40 20   ?   391       ? ?   0:00 lpsched
 8 S   1    184     1  0  40 20   ?   306       ? ?   0:00 statd
 8 S   0    214     1  0  40 20   ?   365       ? ?   0:00 nscd
 8 S   0    204     1  0  40 20   ?   254       ? ?   0:00 cron
 8 S   0    232     1  0  40 20   ?   173       ? ?   0:00 powerd
 8 S   0    255   254  0  40 20   ?   215       ? ?   0:00 smcboot
 8 S   0    258     1  0  40 20   ?   356       ? ?   0:02 vold
```

The manual page for the `ps` command describes all the fields displayed with the `ps` command, as well as all the command options. Table 5.4 lists some important fields.

TABLE 5.4 Process Fields

Field	Description
`F`	Flags associated with the process.
`S`	The state of the process.
`UID`	The user ID of the process owner. For many processes, this is `0` because they run `setuid`.
`PID`	The process ID of each process. This value should be unique. Generally, PIDs are allocated lowest to highest, but they wrap at some point. This value is necessary for you to send a signal, such as the `kill` signal, to a process.
`PPID`	The parent process ID. This identifies the parent process that started the process. Using the PPID enables you to trace the sequence of process creation that took place.
`PRI`	The priority of the process. Without the `-c` option, higher numbers mean lower priority. With the `-c` option, higher numbers mean higher priority.
`NI`	The `nice` value, used in priority computation. This is not printed when the `-c` option is used. The process's `nice` number contributes to its scheduling priority. Making a process nicer means lowering its priority.
`ADDR`	The memory address of the process.
`SZ`	The `SIZE` field. This is the total number of pages in the process. Page sizes are 8192 bytes on sun4u systems, but vary on different hardware platforms. Issue the `/usr/bin/pagesize` command to display the page size on your system.
`WCHAN`	The address of an event for which the process is sleeping (if it's `-`, the process is running).
`STIME`	The starting time of the process (in hours, minutes, and seconds).
`TTY`	The terminal assigned to your process.
`TIME`	The cumulative CPU time used by the process in minutes and seconds.
`CMD`	The command that generated the process.

You often want to look at all processes. You can do this using the command `ps -el`. A number of options available with the `ps` command control what information gets printed. A few of them are listed in Table 5.5.

TABLE 5.5 `ps` Command Options

Option	Description
`-A`	Lists information for all processes. Identical to the `-e` option.
`-a`	Lists information about all the most frequently requested processes. Processes not associated with a terminal will not be listed.
`-e`	Lists information about every process on the system.

(continues)

TABLE 5.5 *Continued*

Option	Description
`-f`	Generates a full listing.
`-l`	Generates a long listing.
`-P`	Prints the number of the processor to which the process is bound, if any, under an additional column header `PSR`. This is a useful option on systems that have multiple processors.
`-u <username>`	Lists only process data for a particular user. In the listing, the numerical user ID is printed unless you give the `-f` option, which prints the login name.

For a complete list of options to the `ps` command, refer to the Solaris online manual pages.

NOTE

`sort` Command The `sort` command is useful when you're looking at system processes. Use the `sort` command as the pipe output to sort by size or PID. For example, to sort by the `SZ` field, use the command `ps -el ¦ sort +9` (remember, `sort` starts numbering fields with 0).

pgrep

The `pgrep` command replaces the combination of the `ps`, `grep`, `egrep`, and `awk` commands that were used to manage processes in earlier releases of Solaris. The `pgrep` command examines the active processes on the system and reports the process IDs of the processes whose attributes match the criteria you specify on the command line. The command syntax for the `pgrep` command is shown here:

```
pgrep <options> <pattern>
```

`pgrep` options are described in Table 5.6.

TABLE 5.6 `pgrep` Options

Option	Description
`-d <delim>`	Specifies the output delimiter string to be printed between each matching process ID. If no `-d` option is specified, the default is a newline character.
`-f`	The regular expression pattern should be matched against the full process argument string. If no `-f` option is specified, the expression is matched only against the name of the executable file.
`-g <pgrplist>`	Matches only processes whose process group ID is in the given list.
`-G <gidlist>`	Matches only processes whose real group ID is in the given list. Each group ID may be specified as either a group name or a numerical group ID.

TABLE 5.6 *Continued*

Option	Description
`-l`	Long output format. Prints the process name along with the process ID of each matching process.
`-n`	Matches only the newest (most recently created) process that meets all other specified matching criteria.
`-P` *`<ppidlist>`*	Matches only processes whose parent process ID is in the given list.
`-s` *`<sidlist>`*	Matches only processes whose process session ID is in the given list.
`-t` *`<termlist>`*	Matches only processes that are associated with a terminal in the given list. Each terminal is specified as the suffix following `/dev/` of the terminal's device pathname in `/dev` (for example, `term/a` or `pts/0`).
`-u` *`<euidlist>`*	Matches only processes whose effective user ID is in the given list. Each user ID may be specified as either a login name or a numerical user ID.
`-U` *`<uidlist>`*	Matches only processes whose real user ID is in the given list. Each user ID may be specified as either a login name or a numerical user ID.
`-v`	Matches all processes except those that meet the specified matching criteria.
`-x`	Considers only processes whose argument string or executable filename exactly matches the specified pattern.
`<pattern>`	A pattern to match against either the executable filename or full process argument string.

For example, the following `pgrep` command finds all processes that have `"dt"` in the process argument string:

```
# pgrep -l -f "dt"
```

The system responds with this:

```
  500 /usr/dt/bin/dtlogin -daemon
16224 ./dtterm
  438 /usr/dt/bin/dtlogin -daemon
  448 /usr/openwin/bin/Xsun :0 -defdepth 24 -nobanner -auth /var/dt/A:0-p_aW2a
  520 dtgreet -display :0
```

To find the process ID for the `lpsched` process, issue this command:

```
# pgrep -l lpsched
```

The system responds with this:

```
6899 lpsched
```

`prstat`

Use the `prstat` command from the command line to monitor system processes. Again, like the `ps` command, it provides information on active processes. The difference is that you can specify whether you want information on specific processes, UIDs, CPU IDs, or processor sets. By default, `prstat` displays information about all processes sorted by CPU usage. Another nice feature with `prstat` is that the information remains on the screen and is updated periodically. The information displayed by the `prstat` command is described in Table 5.7.

TABLE 5.7 Column Headings for the `prstat` Command

Column Heading	Description
`PID`	The process identification (a unique number that defines the process within the kernel)
`USERNAME`	The login ID name of the owner of the process
`SIZE`	The total virtual memory size of the process in kilobytes (K), megabytes (M), or gigabytes (G)
`RSS`	The resident set size of the process in kilobytes, megabytes, or gigabytes
`STATE`	The state of the process:
	`cpu<`*n*`>`—Process is running on CPU.
	`sleep`—Process is waiting for an event to complete.
	`run`—Process is in the run queue.
	`zombie`—Process has terminated and parent is not waiting.
	`stop`—Process is stopped.
`PRI`	The priority of the process
`NICE`	The value used in priority computation
`TIME`	The cumulative execution time for the process
`CPU`	The percentage of recent CPU time used by the process
`PROCESS`	The name of the process
`NLWP`	The number of lightweight processes (LWPs) in the process

This section will introduce some new terminology, so Table 5.8 defines a few terms related to processing in general.

TABLE 5.8 Process Terminology

Term	Description
Multitasking	A technique used in an operating system for sharing a single processor among several independent jobs. Multitasking introduces overhead because the processor spends some time choosing the next job to run and saving and restoring tasks' state. However, it reduces the worst-case time from job submission to completion compared with a simple batch system, in which each job must finish before the next one starts. Multitasking also means that while one task is waiting for some external event, the CPU is free to do useful work on other tasks. A multitasking operating system should provide some degree of protection of one task from another to prevent tasks from interacting in unexpected ways, such as accidentally modifying the contents of each other's memory areas. The jobs in a multitasking system may belong to one or many users. This is distinct from parallel processing, in which one user runs several tasks on several processors. Time sharing is almost synonymous with multitasking, but it implies that there is more than one user.
Parallel processing	The simultaneous use of more than one CPU to solve a problem. The processors either may communicate to cooperate in solving a problem or may run completely independently, possibly under the control of another processor that distributes work to the others and collects results from them.
Multithreaded	Multithreaded is a process that has multiple flows (threads) of control. The traditional Unix process contained, and still contains, a single thread of control. Multithreading (MT) separates a process into many execution threads, each of which runs independently. For more information, see the Multithreaded Programming Guide at `http://docs.sun.com/` Part number 816-5137-10.
Lightweight process (LWP)	A single-threaded subprocess. LWPs are scheduled by the kernel to use available CPU resources based on their scheduling class and priority. LWPs include a kernel thread, which contains information that must be in memory all the time, and a LWP, which contains information that is swappable. A process can consist of multiple LWPs and multiple application threads. A lightweight process is somewhere between a thread and a full process.
Application thread	A series of instructions with a separate stack that can execute independently in a user's address space. The threads can be multiplexed on top of LWPs.
Address space	The range of addresses that a processor or process can access, or at which a device can be accessed. The term may refer to either a physical address or a virtual address. The size of a processor's address space depends on the width of the processor's address bus and address registers. Processes running in 32-bit mode have a 4 gigabyte address space (2^{32} bytes) and processes running in 64-bit mode have a 16 terabyte (2^{64} bytes) address space.
Shared memory	Usually refers to RAM, which can be accessed by more than one process in a multitasking operating system with memory protection.

The syntax for the `prstat` command is as follows:

```
prstat [options] <count> <interval>
```

Table 5.9 describes a few of the `prstat` command options and arguments.

TABLE 5.9 `prstat` Options and Arguments

Option	Description
`prstat` Options	
`-a`	Displays separate reports about processes and users at the same time.
`-c`	Continuously prints new reports beneath previous reports instead of overwriting them.
`-j <projlist>`	Reports only processes or LWPs whose project ID is in the given list. Each project ID can be specified as either a project name or a numerical project ID.
`-J`	Reports information about processes and projects.
`-k <tasklist>`	Reports only processes or LWPs whose task ID is in *tasklist*.
`-m`	Reports microstate process accounting information. In addition to all fields listed in `-v` mode, this mode also includes the percentage of time the process has spent processing system traps, text page faults, and data page faults, and waiting for user locks and waiting for CPU (latency time).
`-n <nproc>`	Restricts the number of output lines. The `<nproc>` argument specifies how many lines of process or LWP statistics are reported.
`-p <pidlist>`	Reports only processes that have a PID in the given list.
`-P <cpulist>`	Reports only processes or LWPs that have most recently executed on a CPU in the given list. The `<cpulist>` argument identifies each CPU by an integer as reported by `psrinfo`.
`-S <key>`	Sorts output lines by `<key>` in descending order. Values for `<key>` can be `cpu`—Sorts by process CPU usage. This is the default. `time`—Sorts by process execution time. `size`—Sorts by size of process image. `rss`—Sorts by resident set size. `pri`—Sorts by process priority.
`-s <key>`	Sorts output lines by `<key>` in ascending order. See the `-S` option for a list of valid *`keys`* to use.
`-t`	Reports total usage summary for each user.
`-u <uidlist>`	Reports only processes whose effective user ID is in the given list. The value for `<uidlist>` may be specified as either a login name or a numerical user ID.

TABLE 5.9 *Continued*

Option	Description
-U *<uidlist>*	Reports only processes whose real user ID is in the given list. The value for *<uidlist>* may be specified as either a login name or a numerical user ID.
prstat Arguments	
<count>	Specifies the number of times that the statistics are repeated. By default, prstat reports statistics until a termination signal is received.
<interval>	Specifies the sampling interval in seconds; the default interval is 5 seconds.

NOTE

psrinfo Command psrinfo displays one line for each configured processor, displaying whether it is online, non-interruptible, offline, or powered off, as well as when that status last changed.

The following example uses the prstat command to view the four most active root processes running. The -n option is used here to restrict the output to the top four processes. The next number, 5, specifies the sampling interval in seconds, and the last number, 3, runs the command three times:

```
# prstat -u root -n 4 5 3
```

The system displays the following output:

```
PID USERNAME  SIZE   RSS  STATE  PRI  NICE  TIME    CPU  PROCESS/NLWP
4375 root     4568K 4344K cpu0    59    0   0:00:00 0.4% prstat/1
4298 root     7088K 5144K sleep   59    0   0:00:02 0.2% dtterm/1
 304 root     2304K 1904K sleep   59    0   0:02:35 0.0% mibiisa/7
 427 root     1832K 1304K sleep   59    0   0:00:00 0.0% rpc.rstatd/1
Total: 53 processes, 111 lwps, load averages: 0.02, 0.01, 0.01
```

The output updates on your display five times every three seconds.

I described projects in Chapter 4, "User and Security Administration," where user accounts can be assigned to project groups. These projects can also be used to label workloads and separate projects and a project's related processes from one another.

The project provides a networkwide administrative identifier for related work. A project consists of tasks, which collect a group of processes into a manageable entity that represents a workload component.

You can use the `prstat` command with the `-J` option to monitor the CPU usage of projects and the `-k` option to monitor tasks across your system. Therefore, you can have `prstat` report on the processes related to a project rather than just list all system processes. In addition, the system administrator can set processing limits on the project, such as setting a limit on the total amount of physical memory, in bytes, that is available to processes in the project. For more information on projects and resource capping, read the man pages on the following commands: `rcapd(1M)`, `project(4)`, `rcapstat(1)`, and `rcapadm(1M)`.

mpstat

Use the `mpstat` command to report processor statistics on a multi-processor system. When executing the `mpstat` command, we'll usually want to see more than one result, so we specify the number of seconds between each mpstat as follows:

```
mpstat 30
```

The argument **30**, specifies that I want to get a report every 30 seconds. The system displays the following information every 30 seconds:

```
CPU minf mjf xcal  intr ithr  csw icsw migr smtx  srw syscl  usr sys  wt idl
  0    6   0    0   114   14   25    0    6    3    0    48    1   2  25  72
  1    6   0    0    86   85   50    0    6    3    0    66    1   4  24  71
  2    7   0    0    42   42   31    0    6    3    0    54    1   3  24  72
  3    8   0    0     0    0   33    0    6    4    0    54    1   3  24  72
```

The results are from a system with four processors. Typically, a system administrator will use the `mpstat` command to check CPU utilization. In this example, I look at the `idl` column (percent idle time) and see that the server's CPUs are approx 28% used. For more information on the other columns of information, refer to the `mpstat` man pages.

ptree

The `ptree` command will display the process tree. The parent process is displayed with the respective child processes indented beneath it. Here is an example showing the processes that belong to the `inetd` process (PID 270):

```
# ptree 270<cr>
```

The system displays:

```
270   /usr/lib/inet/inetd start
  780   /usr/sbin/in.telnetd
    783   -sh
      1250  ptree 270
```

With no arguments, the `ptree` command will display every process along with the associated child processes.

time

The `time` command is used to display the time that the system has spent executing a command. It's a useful command for benchmarking performance. Use this command to time a command on a particular system configuration and compare to another system. In the following example, I'll check the system processing time for a script I wrote named "`longtime`":

```
# time ./longtime<cr>
```

The system displays:

```
real       14.7
user        9.9
sys         2.3
```

The `real` time is the total time that has elapsed between invoking the script and its termination. The `user` time is the time the processor spends executing your user code. Finally, the `system` time is the time the processor spends executing Operating System code on behalf of your process.

Process Manager

In the Desktop Environment (CDE & JAVA Desktop) you have access to the Process Manager GUI, `sdtprocess`, a graphical tool that provides a process manager window for monitoring and controlling system processes.

EXAM ALERT

The exam will most likely ask you about the command-line tools used to manage system processes, such as `kill`, `pkill`, `pargs`, and `pgrep`. You only need to understand that GUI tools can be used to manage processes and you should be prepared to identify these GUI tools.

The advantage of using the Process Manager is that you can view and control processes without knowing all the complex options associated with the `ps` and `kill` commands. For example, you can display processes that contain specific character strings, and you can sort the process list alphabetically or numerically. You can initiate a search using the `find` command, or you can terminate a process simply by highlighting it and clicking `kill`.

To open the Process Manager, you need to log into the Desktop windowing environment. You can start the GUI by executing the command `sdtprocess`, as follows:

```
# sdtprocess &
```

Or, you can click Find Process on the Tools subpanel, as shown in Figure 5.1.

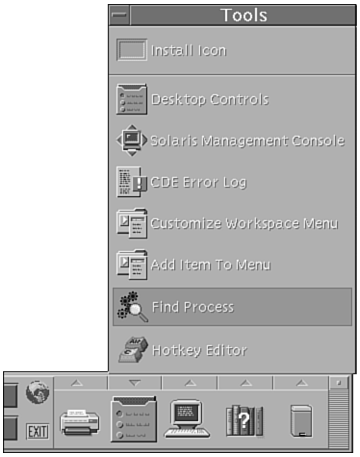
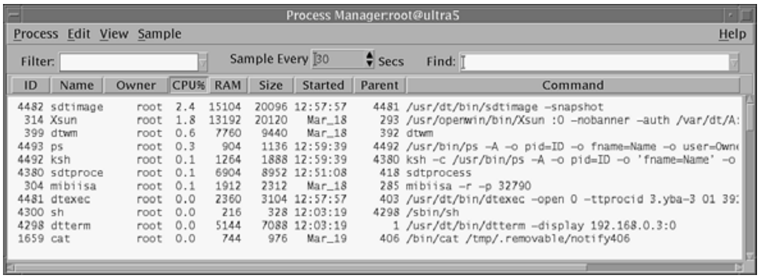


FIGURE 5.1 Front panel.

The Process Manager window opens, as shown in Figure 5.2.



```
  ID   Name       Owner  CPU%  RAM    Size  Started   Parent  Command
4482 sdtimage     root   2.4  15104  20096 12:57:57    4481 /usr/dt/bin/sdtimage -snapshot
 314 Xsun         root   1.8  13192  20120   Mar_18     293 /usr/openwin/bin/Xsun :0 -nobanner -auth /var/dt/A:
 399 dtwm         root   0.6   7760   9440   Mar_18     392 dtwm
4493 ps           root   0.3    904   1136 12:59:39    4492 /usr/bin/ps -A -o pid=ID -o fname=Name -o user=Own
4492 ksh          root   0.1   1264   1888 12:59:39    4380 ksh -c /usr/bin/ps -A -o pid=ID -o 'fname=Name' -o
4380 sdtproce     root   0.1   6904   8952 12:51:08     418 sdtprocess
 304 mibiisa      root   0.1   1912   2312   Mar_18     285 mibiisa -r -p 32790
4481 dtexec       root   0.0   2360   3104 12:57:57     403 /usr/dt/bin/dtexec -open 0 -ttprocid 3.yba-3 01 39:
4300 sh           root   0.0    216    328 12:03:19    4298 /sbin/sh
4298 dtterm       root   0.0   5144   7088 12:03:19       1 /usr/dt/bin/dtterm -display 192.168.0.3:0
1659 cat          root   0.0    744    976   Mar_19     406 /bin/cat /tmp/.removable/notify406
```

Figure 5.2 Process Manager window.

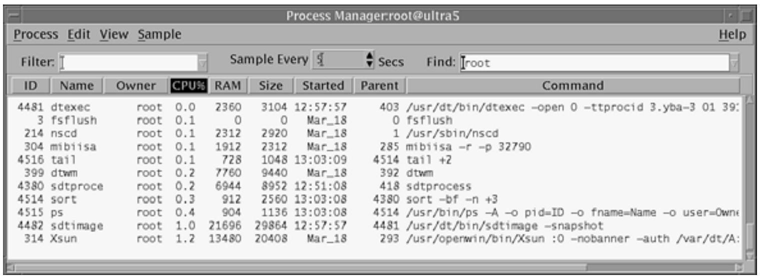
Each process attribute in the header of the Process Manager window provides detailed information about the process and is described in Table 5.10.

TABLE 5.10 Process Manager Window

Column Heading	Description
`ID`	Process ID
`Name`	Process name
`Owner`	Login ID name of the owner of the process
`CPU%`	Ratio of CPU time available in the same period, expressed as a percentage
`RAM`	Amount of RAM currently occupied by this process
`Swap`	Total swap size in virtual memory
`Started`	Actual start time (or date, if other than current)
`Parent`	Process ID of parent process, or PPID
`Command`	Actual Unix command (truncated) being executed

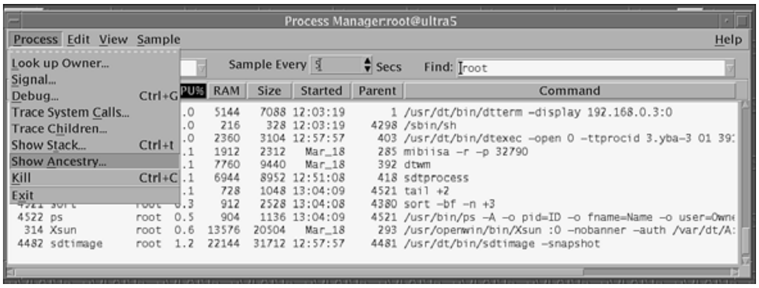
Click any of the column headings to sort the processes by that attribute. For example, click the `CPU` heading to sort all processes by their CPU usage. The list updates every 30 seconds, but you can enter a value in the `Sampling` field to update the list as frequently as you like. Finally, you can enter a text string that is common to the process entries of all the processes you want to display in the Find drop-down menu. In Figure 5.3, I entered "root" in the `Find` field to display all processes owned by root. I also changed the sampling rate to every 5 seconds and clicked the `CPU` heading to sort processes by their CPU usage.

Another nice feature of the Process Manager is the capability to display the ancestry of a process. When a Unix process initiates one or more processes, these are *child processes*, or children. Child and parent processes have the same user ID. To view a parent process and all the child processes that belong to it, highlight the process in the Process Manager window. Click Process from the toolbar at the top of the window and select Show Ancestry, as shown in Figure 5.4.



```
  ID  Name       Owner  CPU%   RAM   Size  Started  Parent Command
4481 dtexec       root  0.0   2360   3104 12:57:57     403 /usr/dt/bin/dtexec -open 0 -ttprocid 3.yba-3 01 39:
   3 fsflush      root  0.1      0      0   Mar_18       0 fsflush
 214 nscd         root  0.1   2312   2920   Mar_18       1 /usr/sbin/nscd
 304 mibiisa      root  0.1   1912   2312   Mar_18     285 mibiisa -r -p 32790
4516 tail         root  0.1    728   1048 13:03:09    4514 tail +2
 399 dtwm         root  0.2   7760   9440   Mar_18     392 dtwm
4380 sdtproce     root  0.2   6944   8952 12:51:08     418 sdtprocess
4514 sort         root  0.3    912   2560 13:03:08    4380 sort -bf -n +3
4515 ps           root  0.4    904   1136 13:03:08    4514 /usr/bin/ps -A -o pid=ID -o fname=Name -o user=Own
4482 sdtimage     root  1.0  21696  29864 12:57:57    4481 /usr/dt/bin/sdtimage -snapshot
 314 Xsun         root  1.2  13480  20408   Mar_18     293 /usr/openwin/bin/Xsun :0 -nobanner -auth /var/dt/A:
```

FIGURE 5.3 Sorted Process Manager window.



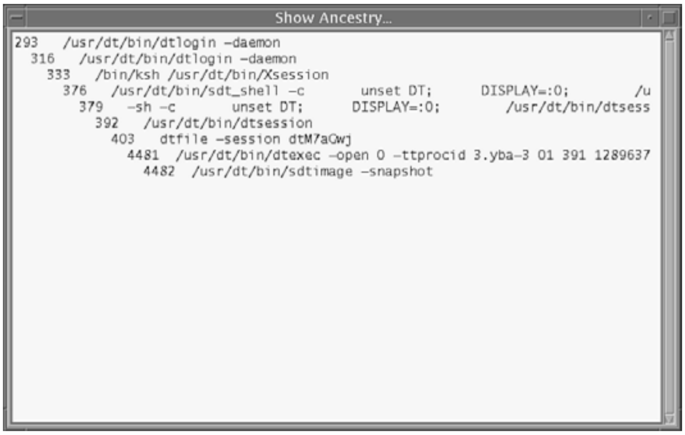
```
 PU%   RAM   Size  Started  Parent Command
 .0   5144   7088 12:03:19       1 /usr/dt/bin/dtterm -display 192.168.0.3:0
 .0    216    328 12:03:19    4298 /sbin/sh
 .0   2360   3104 12:57:57     403 /usr/dt/bin/dtexec -open 0 -ttprocid 3.yba-3 01 39:
 .1   1912   2312   Mar_18     285 mibiisa -r -p 32790
 .1   7760   9440   Mar_18     392 dtwm
 .1   6944   8952 12:51:08     418 sdtprocess
 .1    728   1048 13:04:09    4521 tail +2
 .3    912   2528 13:04:08    4380 sort -bf -n +3
4522 ps           root  0.5    904   1136 13:04:09    4521 /usr/bin/ps -A -o pid=ID -o fname=Name -o user=Own
 314 Xsun         root  0.6  13576  20504   Mar_18     293 /usr/openwin/bin/Xsun :0 -nobanner -auth /var/dt/A:
4482 sdtimage     root  1.2  22144  31712 12:57:57    4481 /usr/dt/bin/sdtimage -snapshot
```

FIGURE 5.4 Selecting Show Ancestry.

The window shown in Figure 5.5 displays showing all the processes belonging to the parent.

The command-line equivalent to the Ancestry selection in the Process Manager is the `ptree` command. Use this command when you don't have a graphical display terminal. The `ptree` command displays the process ancestry trees containing the specified PIDs or users. The child processes are displayed indented from their respective parent processes. For example, here is the process tree for the `-sh` process, which has a PID of `293`:

```
# ptree 293
```



```
293   /usr/dt/bin/dtlogin -daemon
  316   /usr/dt/bin/dtlogin -daemon
    333   /bin/ksh /usr/dt/bin/Xsession
      376   /usr/dt/bin/sdt_shell -c       unset DT;      DISPLAY=:0;        /u
        379   -sh -c       unset DT;      DISPLAY=:0;        /usr/dt/bin/dtsess
          392   /usr/dt/bin/dtsession
            403   dtfile -session dtM7aQwj
              4481  /usr/dt/bin/dtexec -open 0 -ttprocid 3.yba-3 01 391 1289637
                4482  /usr/dt/bin/sdtimage -snapshot
```

Figure 5.5 Show Ancestry window.

The system responds with this:

```
293  /usr/dt/bin/dtlogin -daemon
  316   /usr/dt/bin/dtlogin -daemon
    333   /bin/ksh /usr/dt/bin/Xsession
      376   /usr/dt/bin/sdt_shell -c  unset DT;DISPLAY=:0;/usr/dt/bin/dt
        379  -sh -c unset DT; DISPLAY=:0; usr/dt/bin/dtsession_res - \
              merge
          392   /usr/dt/bin/dtsession
            402   /usr/dt/bin/dtterm -session dthIaGth -C -ls
              418   -sh
```

SMC Process Tool

The Solaris Management Console (SMC) includes a GUI called the Process Tool, which is used for viewing and managing processes, similar to the Desktop Process Manager tool described in the previous section. You can use the job scheduler tool to

- Suspend a process
- Resume a suspended process
- Kill a process
- Display information about a process

To open the Process Tool, follow Step by Step 5.1.

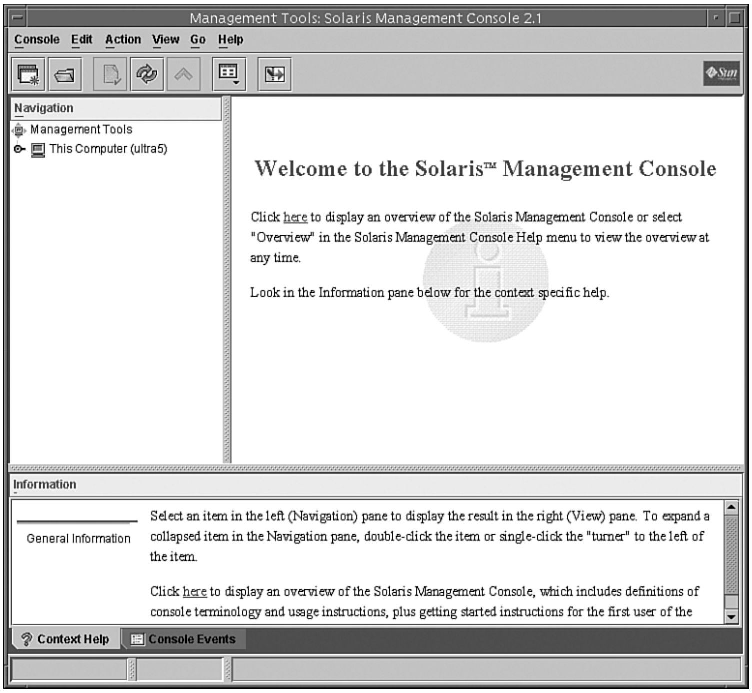
STEP BY STEP

5.1 Opening the Process Tool

1. Start up the Solaris Management Console in the background by typing

```
# smc &
```

2. The SMC Welcome window appears as shown in Figure 5.6.
3. In the SMC navigation pane, open the Process Tool by clicking on the This Computer icon, then click on the System Status icon, then click on the Processes icon as shown in Figure 5.7.
4. The Process Tool displays as shown in Figure 5.8



Welcome to the Solaris™ Management Console

Click here to display an overview of the Solaris Management Console or select "Overview" in the Solaris Management Console Help menu to view the overview at any time.

Look in the Information pane below for the context specific help.

Select an item in the left (Navigation) pane to display the result in the right (View) pane. To expand a collapsed item in the Navigation pane, double-click the item or single-click the "turner" to the left of the item.

Click here to display an overview of the Solaris Management Console, which includes definitions of console terminology and usage instructions, plus getting started instructions for the first user of the

FIGURE 5.6 SMC Welcome Window.

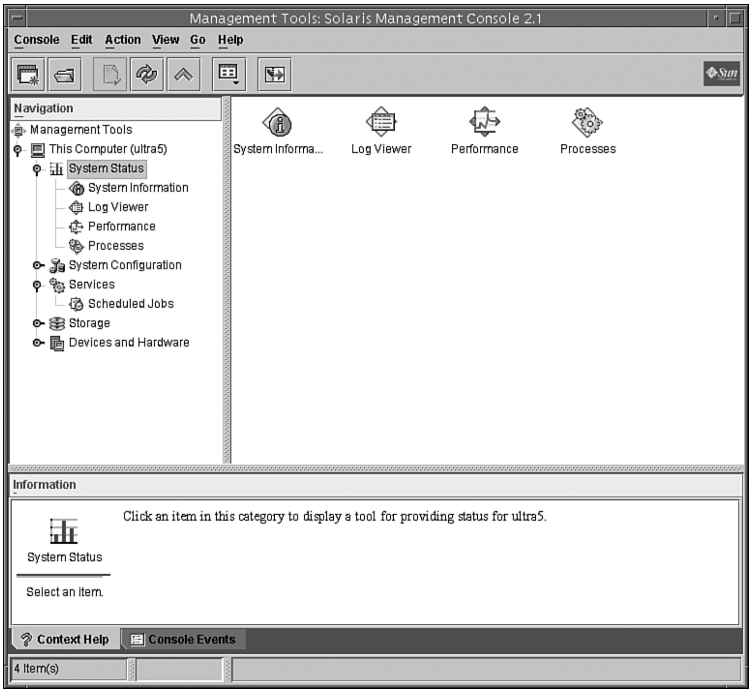
The Process Tool works much the same way as the Process Manager tool described earlier.

pargs

The `pargs` command is used from the command line to examine the arguments and environment variables of a process (or number of processes). `pargs` can also be used to examine core files.

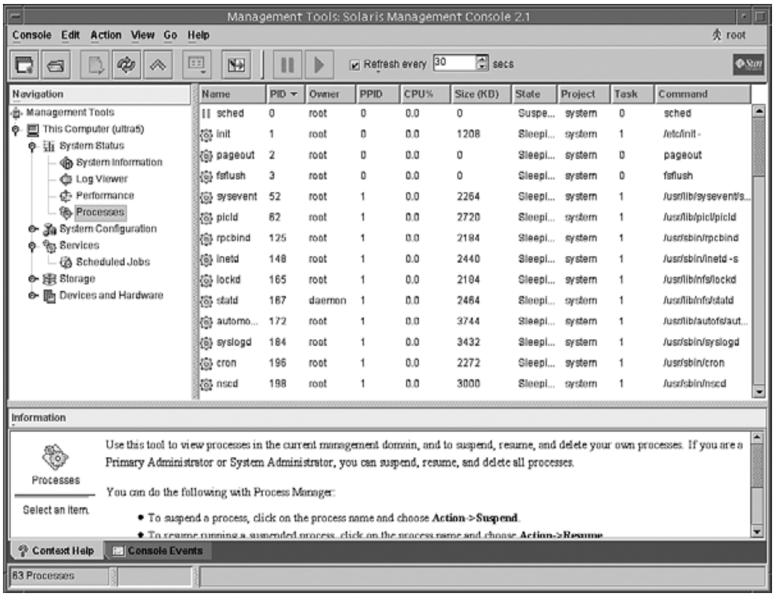
The syntax for the `pargs` command is as follows:

```
pargs [options] [pid ¦ core]
```



Click an item in this category to display a tool for providing status for ultra5.

FIGURE 5.7 Opening the Job Scheduler.



Name	PID	Owner	PPID	CPU%	Size (KB)	State	Project	Task	Command
sched	0	root	0	0.0	0	Suspe...	system	0	sched
init	1	root	0	0.0	1208	Sleepi...	system	1	/etc/init -
pageout	2	root	0	0.0	0	Sleepi...	system	0	pageout
fsflush	3	root	0	0.0	0	Sleepi...	system	0	fsflush
sysevent	52	root	1	0.0	2264	Sleepi...	system	1	/usr/lib/sysevent/s...
picld	62	root	1	0.0	2720	Sleepi...	system	1	/usr/lib/picl/picld
rpcbind	125	root	1	0.0	2184	Sleepi...	system	1	/usr/sbin/rpcbind
inetd	148	root	1	0.0	2440	Sleepi...	system	1	/usr/sbin/inetd -s
lockd	165	root	1	0.0	2184	Sleepi...	system	1	/usr/lib/nfs/lockd
statd	167	daemon	1	0.0	2464	Sleepi...	system	1	/usr/lib/nfs/statd
automo...	172	root	1	0.0	3744	Sleepi...	system	1	/usr/lib/autofs/aut...
syslogd	184	root	1	0.0	3432	Sleepi...	system	1	/usr/sbin/syslogd
cron	196	root	1	0.0	2272	Sleepi...	system	1	/usr/sbin/cron
nscd	198	root	1	0.0	3000	Sleepi...	system	1	/usr/sbin/nscd

Use this tool to view processes in the current management domain, and to suspend, resume, and delete your own processes. If you are a Primary Administrator or System Administrator, you can suspend, resume, and delete all processes.

You can do the following with Process Manager:

- To suspend a process, click on the process name and choose **Action->Suspend**.
- To resume running a suspended process, click on the process name and choose **Action->Resume**

FIGURE 5.8 Process Tool.

Table 5.11 describes the `pargs` command options and arguments.

TABLE 5.11 `pargs` Options and Arguments

Option/Arguments	Description
`-a`	Prints the process arguments.
`-c`	Treats strings in the target process as though they were encoded in 7-bit ASCII.
`-e`	Prints process environment variables and values.
`-F`	Force. Grabs the target process even if another process has control.
`-x`	Prints process auxiliary vector.
`<pid>`	Process ID list. The PID list can be a single process ID or multiple PIDs separated by a space.
`core`	Processes a core file.

For example, to use the `pargs` command to view all of the environment variables associated with the `telnetd` process, I first need to find the PID of the `telnetd` process using `pgrep` as follows:

```
# pgrep telnetd
16173
```

Next, I issue the `pargs` command using the PID for the `telnetd` process as an argument:

```
# pargs -e 16173
```

The system responds with

```
16173:  /usr/sbin/in.telnetd
envp[0]: SMF_RESTARTER=svc:/network/inetd:default
envp[1]: SMF_FMRI=svc:/network/telnet:default
envp[2]: SMF_METHOD=inetd_start
envp[3]: PATH=/usr/sbin:/usr/bin
envp[4]: TZ=US/Michigan
```

svcs

The Service Management Facility (SMF) was described in Chapter 3, "Perform System Boot and Shutdown Procedures for SPARC, x64, and x86-Based Systems," so I won't be redundant by describing it again here. However, this is just a reminder that you can use the `svcs` command with the `-p` option to list all processes associated with each service instance.

Process Types

When sitting at a terminal and typing in commands, the user is typically executing *foreground processes*. Commands such as `vi` are foreground processes—they read input from the keyboard and display output to the terminal. Foreground processes maintain control of the terminal, and the user cannot do anything else in that terminal window until the execution of that command is complete.

Some processes are not interactive and don't need to run in the foreground. These are referred to as *background processes* or *jobs*. A background process gets detached from the terminal, freeing up the terminal while it is running. When a user decides to run a process in the background, you must arrange for the process to get its input from another source. In addition, you need to arrange for the process to output to a device other than the terminal, such as a file.

To run a process in the background, enter an `&` (ampersand) after the command:

```
# find . -name core -print &
```

After typing in this command, you're returned to a command prompt. The `find` command executes in the background. One problem, however, is the standard output is still on your terminal. In other words, as the `find` command executes, the results still are displayed on your screen, which can become quite annoying. It's best to redirect the output to a file, as follows:

```
# find . -name core -print  > /tmp/results &
```

After you put the `find` command in the background, the system displays two numbers associated with that process—the job number and the process ID number (PID) as follows:

```
    [1]    14919
```

You use this job number to control background processes.

NOTE

No Job Control in the `sh` shell The Bourne shell does not provide job control. Job control enables you to check and manage your background jobs. Thus, with the Bourne shell, you can submit jobs to the background, but you cannot manage them. Use `jsh` (job shell), which provides all the functionality of `sh` and enables job control. The Korn shell (`ksh`) and the C shell (`csh`) both allow for job control.

The shell maintains a table containing information about processes that are currently in the background. This is referred to as the *jobs table*. The jobs table is unique to the user, and each user has his own jobs table. Furthermore, the jobs table contains only entries for jobs that are running in your current shell. If you start a new shell, the jobs table for the new shell is empty. Each job in the table is assigned a number that is unique to that user only. In other words, two users can each have a job numbered 1. Don't confuse this job number with a process ID number; remember, process IDs are unique, and no two share the same number. Any jobs that the user has placed in the background are displayed here by typing in the `jobs` command, as follows:

```
# jobs
```

The system responds with this:

```
[3] +  Running   find / -name bill -print > /tmp/results3 &
[2] -  Running   find / -name junk -print > /tmp/results2 &
[1]    Running   find / -name core -print > /tmp/results1 &
```

The jobs table contains the following information:

- A numeric value for each job
- A + (plus) symbol to designate the current job that user commands will operate on
- A - (minus) symbol to designate the next job that the user commands will operate on
- The status of the job
- The name of the job

Each job in the job table has one of the following states:

- `Running`—An active job
- `Stopped`—A job that has been suspended
- `Terminated`—A job that has been killed
- `Done`—A completed job

When the job finishes, the following is displayed on your terminal:

```
[1] +  Done      find / -name core -print > /tmp/results &
```

Note the job number of `1` and the status of `Done`.

If you want to terminate a job, use the `kill` command followed by a `%` (percent sign) and then the job number, as follows:

```
# kill %1
```

CAUTION

Pay special attention to the use of the `%` (percent) symbol—it's absolutely required. Without it, you could kill the wrong process and potentially crash the system. Get familiar with the `kill` command in the next section of this chapter before you use it.

If you do not enter a number following the `%` sign, the command acts upon the current job entry listed in the jobs table. For this example, you are going to kill job number 1, as follows:

```
# kill %1
```

The following message is displayed indicating successful termination:

```
[1] + Terminated   find / -name core -print > /tmp/results &
```

You can also bring a job back into the foreground with the `fg` command. Typing `fg` with no arguments brings the current job (the job with the + sign next to it in the jobs table) into the foreground. You can also specify the job by typing `fg %<job number>`, as follows:

```
# fg %2
```

This brings job 2 back into the foreground on your terminal.

In a windowing environment such as Java Desktop System, placing jobs in the background is not an issue. Typically, you start a job in one window and open another window to continue working. Therefore, placing jobs into the background has all but disappeared unless you are working on a character-based terminal.

Using Signals

Objective:

Clearing frozen processes.

Solaris supports the concept of sending software *signals* to a process. These signals are ways for other processes to interact with a running process outside the context of the hardware. The `kill` command is used to send a signal to a process. System administrators most often use the signals `SIGHUP`, `SIGKILL`, `SIGSTOP`, and `SIGTERM`. The `SIGHUP` signal is used by some utilities as a way to notify the process to do something, such as re-read its configuration file. The `SIGHUP` signal is also sent to a process if the remote connection is lost or hangs up. The `SIGKILL` signal is used to abort a process, and the `SIGSTOP` signal is used to pause a process. The SIGTERM signal is the default signal sent to processes by commands such as `kill` and `pkill` when no signal is specified. Table 5.12 describes the most common signals an administrator is likely to use.

EXAM ALERT

Don't worry about remembering all of the signals listed; just be familiar with the more common signals, such as `SIGHUP`, `SIGKILL`, `SIGSTOP`, and `SIGTERM`.

TABLE 5.12 Signals Available Under Solaris

Signal	Number	Description
`SIGHUP`	1	Hangup. Usually means that the controlling terminal has been disconnected.
`SIGINT`	2	Interrupt. The user can generate this signal by pressing Ctrl+C or Delete.

(continues)

TABLE 5.12 *Continued*

Signal	Number	Description
SIGQUIT	3	Quits the process and produces a core dump.
SIGILL	4	Illegal instruction.
SIGTRAP	5	Trace or breakpoint trap.
SIGABRT	6	Abort.
SIGEMT	7	Emulation trap.
SIGFPE	8	Arithmetic exception. Informs a process of a floating-point error.
SIGKILL	9	Killed. Forces the process to terminate. This is a sure kill.
SIGBUS	10	Bus error.
SIGSEGV	11	Segmentation fault.
SIGSYS	12	Bad system call.
SIGPIPE	13	Broken pipe.
SIGALRM	14	Alarm clock.
SIGTERM	15	Terminated. A gentle kill that gives processes a chance to clean up.
SIGUSR1	16	User signal 1.
SIGUSR2	17	User signal 2.
SIGCHLD	18	Child status changed.
SIGPWR	19	Power fail or restart.
SIGWINCH	20	Window size change.
SIGURG	21	Urgent socket condition.
SIGPOLL	22	Pollable event.
SIGSTOP	23	Stopped (signal). Pauses a process.
SIGTSTP	24	Stopped (user).
SIGCONT	25	Continued.
SIGTTIN	26	Stopped (tty input).
SIGTTOU	27	Stopped (tty output).
SIGVTALRM	28	Virtual timer expired.
SIGPROF	29	Profiling timer expired.
SIGXCPU	30	CPU time limit exceeded.
SIGXFSZ	31	File size limit exceeded.
SIGWAITING	32	Concurrency signal reserved by threads library.
SIGLWP	33	Inter-LWP signal reserved by threads library.
SIGFREEZE	34	Checkpoint freeze.
SIGTHAW	35	Checkpoint thaw.
SIGCANCEL	36	Cancellation signal reserved by the threads library.

NOTE

Obtain a list of the signals by typing: `man signal.h`

In addition, you can write a *signal handler*, or *trap*, in a program to respond to a signal being sent. For example, many system programs, such as the name server daemon, respond to the `SIGHUP` signal by re-reading their configuration files. This signal can then be used to update the process while running, without having to terminate and restart the process. Signal handlers cannot be installed for `SIGSTOP` (23) or `SIGKILL` (9). Because the process cannot install a signal handler for signal 9, an otherwise well-behaved process may leave temporary files around or not be able to finish out critical operations that it is in the middle of. Thus, `kill -9` invites corruption of application data files and should only be used as a last resort.

Here's an example of how to trap a signal in a script:

```
# trap '/bin/rm tmp$$;exit 1' 1 2 3 15
```

As the name suggests, `trap` traps system interrupt until some command can be executed. The previous example traps the signals `1`, `2`, `3`, and `15`, and executes the `/bin/rm tmp$$` command before exiting the program. The example deletes all `tmp` files even if the program terminates abnormally.

The `kill` command sends a terminate signal (signal 15) to the process, and the process is terminated. Signal 15, which is the default when no options are used with the `kill` command, is a gentle kill that allows a process to perform cleanup work before terminating. Signal 9, on the other hand, is called a sure, unconditional kill because it cannot be caught or ignored by a process. If the process is still around after a `kill -9`, either it is hung up in the Unix kernel, waiting for an event such as disk I/O to complete, or you are not the owner of the process.

The `kill` command is routinely used to send signals to a process. You can kill any process you own, and the superuser can kill all processes in the system except those that have process IDs `0`, `1`, `2`, `3`, and `4`. The `kill` command is poorly named because not every signal sent by it is used to kill a process. This command gets its name from its most common use—terminating a process with the `kill -15` signal.

NOTE

Forking Problem A common problem occurs when a process continually starts up new copies of itself—this is referred to as *forking* or *spawning*. Users have a limit on the number of new processes they can fork. This limit is set in the kernel with the `MAXUP` (maximum number of user processes) value. Sometimes, through user error, a process keeps forking new copies of itself until the user hits the `MAXUP` limit. As a user reaches this limit, the system appears to be waiting. If you kill some of the user's processes, the system resumes creating new processes on behalf of the user. It can be a no-win situation. The best way to handle these runaway processes is to send the `STOP` signal to all of the runaway processes to suspend the processes and then send a `KILL` signal to terminate the processes. Because the processes were first suspended, they can't create new ones as you kill them off.

You can send a signal to a process you own with the `kill` command. Many signals are available, as listed in Table 5.12. To send a signal to a process, first use the `ps` command to find the process ID (PID) number. For example, type `ps -ef` to list all processes and find the PID of the process you want to terminate:

```
# ps -ef

UID    PID  PPID  C    STIME  TTY   TIME  CMD
root     0     0  0   Nov 27  ?     0:01  sched
root     1     0  0   Nov 27  ?     0:01  /etc/init -
root     2     0  0   Nov 27  ?     0:00  pageout
root     3     0  0   Nov 27  ?    12:52  fsflush
root   101     1  0   Nov 27  ?     0:00  /usr/sbin/in.routed -q
root   298     1  0   Nov 27  ?     0:00  /usr/lib/saf/sac -t 300
root   111     1  0   Nov 27  ?     0:02  /usr/sbin/rpcbind
root   164     1  0   Nov 27  ?     0:01  /usr/sbin/syslogd -n -z 12
root   160     1  0   Nov 27  ?     0:01  /usr/lib/autofs/automountd
.
.
.
root  5497   433  1 09:58:02  pts/4  0:00  script psef
```

To kill the process with a PID number of `5497`, type this:

```
# kill 5497
```

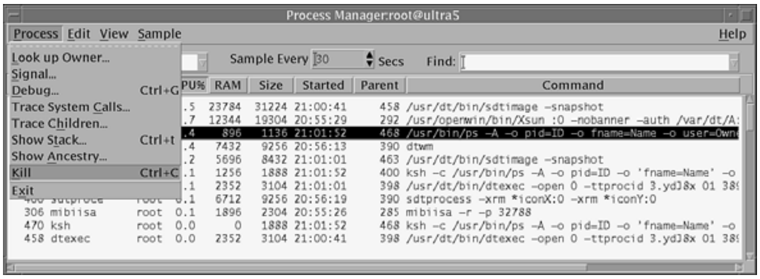
Another way to kill a process is to use the `pkill` command. `pkill` functions identically to `pgrep`, which was described earlier, except that instead of displaying information about each process, the process is terminated. A signal name or number may be specified as the first command-line option to `pkill`. The value for the signal can be any value described in Table 5.12. For example, to kill the process named `psef` with a `SIGKILL` signal, issue the following command:

```
# pkill -9 psef
```

NOTE

Killing a Process If no signal is specified, `SIGTERM (15)` is sent by default. This is the preferred signal to send when trying to kill a process. Only when a `SIGTERM` fails should you send a SIGKILL signal to a process. As stated earlier in this section, a process cannot install a signal handler for signal 9 and an otherwise well-behaved process might not shut down properly.

In addition, the Desktop Process Manager, which was described earlier, can be used to kill processes. In the Process Manager window, highlight the process that you want to terminate, click Process from the toolbar at the top of the window, and then select Kill from the pull-down menu, as shown in Figure 5.9.



```
PU%   RAM   Size  Started   Parent  Command
 .5  23784  31224 21:00:41    458 /usr/dt/bin/sdtimage -snapshot
 .7  12344  19304 20:55:29    292 /usr/openwin/bin/Xsun :0 -nobanner -auth /var/dt/A:
 .4    896   1136 21:01:52    468 /usr/bin/ps -A -o pid=ID -o fname=Name -o user=Own
 .4   7432   9256 20:56:13    390 dtwm
 .2   5696   8432 21:01:01    463 /usr/dt/bin/sdtimage -snapshot
 .1   1256   1888 21:01:52    400 ksh -c /usr/bin/ps -A -o pid=ID -o 'fname=Name' -o
 .1   2352   3104 21:01:01    398 /usr/dt/bin/dtexec -open 0 -ttprocid 3.ydJ8x 01 38
 .1   6712   9256 20:56:19    390 sdtprocess -xrm *iconX:0 -xrm *iconY:0
306 mibiisa    root 0.1   1896   2304 20:55:26    285 mibiisa -r -p 32788
470 ksh        root 0.0      0   1888 21:01:52    468 ksh -c /usr/bin/ps -A -o pid=ID -o 'fname=Name' -o
458 dtexec     root 0.0   2352   3104 21:00:41    398 /usr/dt/bin/dtexec -open 0 -ttprocid 3.ydJ8x 01 38
```

Figure 5.9 Killing processes.

The equivalent Unix command used by the Process Manager to terminate a process is shown here:

```
# kill -9 <PID>
```

`<PID>` is the process ID of the selected process.

The `preap` command forces the killing of a defunct process, known as a *zombie*. In previous Solaris releases, zombie processes that could not be killed off remained until the next system reboot. Defunct processes do not normally impact system operation; however, they do consume a small amount of system memory. See the `preap` manual page for further details of this command.

Scheduling Processes

Processes compete for execution time. *Scheduling*, one of the key elements in a time-sharing system, determines which of the processes executes next. Although hundreds of processes might be present on the system, only one actually uses a given CPU at any given time. Time sharing on a CPU involves suspending a process and then restarting it later. Because the suspension and resumption of active processes occurs many times each second, it appears to the user that the system is performing many tasks simultaneously.

Unix attempts to manage the priorities of processes by giving a higher priority to those that have used the least amount of CPU time. In addition, processes that are waiting on an event, such as a keyboard press, get higher priority than processes that are purely CPU-driven.

On any large system with a number of competing user groups, the task of managing resources falls to the system administrator. This task is both technical and political. As a system administrator, you must understand your company goals to manage this task successfully. When you understand the political implications of who should get priority, you are ready to manage the technical details. As root, you can change the priority of any process on the system by using the `nice` or `priocntl` commands. Before you do this, you must understand how priorities work.

Scheduling Priorities

All processes have assigned to them an execution priority—an integer value that is dynamically computed and updated on the basis of several different factors. Whenever the CPU is free, the scheduler selects the most favored process to resume executing. The process selected is the one with the lowest-priority number because lower numbers are defined as more favored than higher ones. Multiple processes at the same priority level are placed in the run queue for that priority level. Whenever the CPU is free, the scheduler starts the processes at the head of the lowest-numbered nonempty run queue. When the process at the top of a run queue stops executing, it goes to the end of the line and the next process moves up to the front. After a process begins to run, it continues to execute until it needs to wait for an I/O operation to complete, receives an interrupt signal, or exhausts the maximum execution time slice defined on that system. A typical time slice is 10 milliseconds.

A Unix process has two priority numbers associated with it. One of the priority numbers is its requested execution priority with respect to other processes. This value (its `nice` number) is set by the process's owner and by root; it appears in the `NI` column in a `ps -l` listing. The other priority assigned to a process is the execution priority. This priority is computed and updated dynamically by the operating system, taking into account such factors as the process's `nice` number, how much CPU time it has had recently, and other processes that are running and their priorities. The execution priority value appears in the `PRI` column on a `ps -l` listing.

Although the CPU is the most-watched resource on a system, it is not the only one. Memory use, disk use, I/O activity, and the number of processes all tie together in determining the computer's throughput. For example, suppose you have two groups, A and B. Both groups require large amounts of memory—more than is available when both are running simultaneously. Raising the priority of Group A over Group B might not help if Group B does not fully relinquish the memory it is using. Although the paging system does this over time, the process of swapping a process out to disk can be intensive and can greatly reduce performance. A better alternative might be to completely stop Group B with a signal and then continue it later, when Group A has finished.

Changing the Priority of a Time-Sharing Process with `nice`

The `nice` command is supported only for backward compatibility with previous Solaris releases. The `priocntl` command provides more flexibility in managing processes. The priority of a process is determined by the policies of its scheduling class and by its `nice` number. Each time-sharing process has a global priority that is calculated by adding the user-supplied priority, which can be influenced by the `nice` or `priocntl` commands, and the system-calculated priority.

The execution priority number of a process is assigned by the operating system and is determined by several factors, including its schedule class, how much CPU time it has used, and its `nice` number. Each time-sharing process starts with a default `nice` number, which it inherits from its parent process. The `nice` number is shown in the `NI` column of the `ps` report.

A user can lower the priority of a process by increasing its user-supplied priority number. Only the superuser can increase the priority of a process by lowering its `nice` value. This prevents users from increasing the priorities of their own processes, thereby monopolizing a greater share of the CPU.

Two versions of the `nice` command are available: the standard version, `/usr/bin/nice`, and a version that is integrated into the C shell as a C shell built-in. `/usr/bin/nice` numbers range from `0` to `+39` and the default value is 20, while the C-shell built-in version of `nice` has values that range from –20 to +20. The lower the number, the higher the priority and the faster the process runs.

Use the `/usr/bin/nice` command as described in Table 5.13 when submitting a program or command.

TABLE 5.13 Setting Priorities with `nice`

Command	Description
Lowering the Priority of a Process Using `/usr/bin/nice`	
`nice <process_name>`	Increases the `nice` number by 4 units (the default)
`nice -4 <process_name>`	Increases the `nice` number by 4 units
`nice -10 <process_name>`	Increases the `nice` number by 10 units
Increasing the Priority of a Process	
`nice -n -10 <process_name>`	Raises the priority of the command by lowering the `nice` number

NOTE

Root may run commands with a priority higher than normal by using a negative increment, such as -10. A negative increment assigned by an unprivileged user is ignored.

As a system administrator, you can use the `renice` command to change the priority of a process after it has been submitted. The `renice` command has the following form:

```
renice priority -n <value> -p <pid>
```

Use the `ps -elf` command to find the PID of the process for which you want to change the priority. The process that you want to change in the following example is named `largejob`:

```
# ps –elf¦grep largejob
9 S   0   8200  4100  0  84  20  f0274e38  193        Jun 04 ?    0:00 largejob
```

Issue the following command to increase the priority of PID `8200`:

```
renice -n -4 -p 8200
```

Issuing the `ps -elf` command again shows the process with a higher priority:

```
# ps –elf¦grep largejob
9 S   0  8200  4100  0  60  16   f0274e38  193      Jun 04  ?    0:00  largejob
```

Changing the Scheduling Priority of Processes with `priocntl`

The standard priority scheme has been improved since earlier versions of Solaris as part of its support for real-time processes. Real-time processes are designed to work in application areas in which a nearly immediate response to events is required. These processes are given nearly complete access to all system resources when they are running. Solaris uses time-sharing priority numbers ranging from `-20` to `20`. Solaris uses the `priocntl` command, intended as an improvement over the `nice` command, to modify process priorities. To use `priocntl` to change a priority on a process, type this:

```
priocntl -s -p <new-priority>  -i pid <process-id>
```

`new-priority` is the new priority for the process, and *`process-id`* is the PID of the process you want to change.

The following example sets the priority level for process `8200` to `-5`:

```
priocntl -s -p -5 -i pid 8200
```

The following example is used to set the priority (`nice` value) for every process created by a given parent process:

```
priocntl -s -p -5 -I ppid 8200
```

As a result of this command, all processes forked from process `8200` have a priority of `-5`.

The priority value assigned to a process can be displayed using the `ps` command, which was described earlier in this chapter.

The functionality of the `priocntl` command goes much further than what is described in this section. Consult the online manual pages for more information about the `priocntl` command.

Fair Share Scheduler (FSS) and the Fixed Scheduler (FX)

The *Fair Share Scheduler (FSS)* in Solaris 10 can be used to control the allocation of resources. The *Fixed Scheduler (FX)* is a fixed priority scheduler that provides an ensured priority for processes. Neither of these are objectives on the CX-310-200 exam and they are not covered in this chapter.

Using the Solaris Batch-Processing Facility

A way to divide processes on a busy system is to schedule jobs so that they run at different times. A large job, for example, could be scheduled to run at 2:00 a.m., when the system would normally be idle. Solaris supports two methods of batch processing: the `crontab` and `at` commands. The `crontab` command schedules multiple system events at regular intervals, and the `at` command schedules a single system event.

Configuring `crontab`

Objective:

Explain how to schedule the automatic recurring execution of a command.

`cron` is a Unix utility named after Chronos ("time"), the ancient Greek god of time. It enables you to execute commands automatically according to a schedule you define. The `cron` daemon schedules system events according to commands found in each `crontab` file. A `crontab` file consists of commands, one per line, that will be executed at regular intervals. The beginning of each line contains five date and time fields that tell the `cron` daemon when to execute the command. The sixth field is the full pathname of the program you want to run. These fields, described in Table 5.14, are separated by spaces.

TABLE 5.14 The `crontab` File

Field	Description	Values
1	Minute	`0` to `59`. A `*` in this field means every minute.
2	Hour	`0` to `23`. A `*` in this field means every hour.
3	Day of month	`1` to `31`. A `*` in this field means every day of the month.
4	Month	`1` to `12`. A `*` in this field means every month.
5	Day of week	`0` to `6` (`0` = `Sunday`). A `*` in this field means every day of the week.
6	Command	Enter the command to be run.

Follow these guidelines when making entries in the `crontab` file:

- Use a space to separate fields.
- Use a comma to separate multiple values in any of the date or time fields.
- Use a hyphen to designate a range of values in any of the date or time fields.
- Use an asterisk as a wildcard to include all possible values in any of the date or time fields. For example, an asterisk (*) can be used in the first five fields (time fields) to mean all legal values.
- Use a comment mark (#) at the beginning of a line to indicate a comment or a blank line.
- Each command within a `crontab` file must consist of one line, even if it is very long, because `crontab` does not recognize extra carriage returns.
- There can be no blank lines in the `crontab` file. Although this is not documented well, and some crontab files I've seen contain blank lines, the system will generate an email to root with a message that "there is an error in your crontab file."

The following sample `crontab` command entry displays a reminder in the user's console window at 5:00 p.m. on the 1st and 15th of every month:

```
0 17 1,15 * * echo Hand in Timesheet > /dev/console
```

`crontab` files are found in the `/var/spool/cron/crontabs` directory. Several `crontab` files besides root are provided during the SunOS software installation process; they are also located in this directory. Other `crontab` files are named after the user accounts for which they are created, such as bill, glenda, miguel, or nicole. They also are located in the /var/spool/cron/ crontabs directory. For example, a `crontab` file named root is supplied during software installation. Its contents include these command lines:

```
10 3 * * * /usr/sbin/logadm
15 3 * * 0 /usr/lib/fs/nfs/nfsfind
30 3 * * * [ -x /usr/lib/gss/gsscred_clean ] && /usr/lib/gss/gsscred_clean
#10 3 * * * /usr/lib/krb5/kprop_script ___slave_kdcs___
```

The first command line instructs the system to run `/usr/sbin/logadmin` at 3:10 a.m. every day of the week. The second command line orders the system to execute `nfsfind` on Sunday at 3:15 a.m. The third command line runs each night at 3:30 a.m. and executes the `gsscred` command. The fourth command is commented out. The cron daemon never exits and is started via the `svc:/system/cron:default` service. The `/etc/cron.d/FIFO` file is used as a lock file to prevent running more than one instance of cron.

Creating and Editing a `crontab` File

Creating an entry in the `crontab` file is as easy as editing a text file using your favorite editor. Use the steps described next to edit this file; otherwise, your changes are not recognized until the next time the `cron` daemon starts up. `cron` examines `crontab` configuration files only during its own process-initialization phase or when the `crontab` command is run. This reduces the overhead of checking for new or changed files at regularly scheduled intervals.

Step by Step 5.2 tells you how to create or edit a `crontab` file.

STEP BY STEP

5.2 Creating or Editing a `crontab` File

1. (Optional) To create or edit a `crontab` file belonging to root or another user, become superuser.
2. Create a new `crontab` file or edit an existing one by typing the following:

   ```
   # crontab -e
   ```

NOTE

`crontab` Default Editor The `crontab` command chooses the system default editor, which is `ed`, unless you've set the `VISUAL` or `EDITOR` variable to `vi` (or another editor), as follows:

```
# EDITOR=vi;export EDITOR
```

3. Add command lines to the file, following the syntax described in Table 5.14. Because `cron` jobs do not inherit the users environment, such as `PATH`, you should specify the full pathname for commands.
4. Save the changes and exit the file. The `crontab` file is placed in `/var/spool/cron/crontabs`.
5. Verify the `crontab` file by typing the following:

   ```
   # crontab -l
   ```

   The system responds by listing the contents of the `crontab` file.

Controlling Access to `crontab`

TIP

Many of the questions that you encounter regarding `cron` are related to controlling access to `cron` using the files described in this section. Make sure that you have a clear understanding of how you can control a user's access to `cron` by adding and removing entries in these files.

You can control access to `crontab` by modifying two files in the `/etc/cron.d` directory: `cron.deny` and `cron.allow`. These files permit only specified users to perform `crontab` tasks such as creating, editing, displaying, and removing their own `crontab` files. The `cron.deny` and `cron.allow` files consist of a list of usernames, one per line. These access control files work together in the following manner:

- If `cron.allow` exists, only the users listed in this file can create, edit, display, and remove `crontab` files.

- If `cron.allow` doesn't exist, all users may submit `crontab` files, except for users listed in `cron.deny`.
- If neither `cron.allow` nor `cron.deny` exists, superuser privileges are required to run `crontab`.

Superuser privileges are required to edit or create `cron.deny` and `cron.allow`.

During the Solaris software installation process, a default `/etc/cron.d/cron.deny` file is provided. It contains the following entries:

- `daemon`
- `bin`
- `nuucp`
- `listen`
- `nobody`
- `noaccess`

None of the users listed in the `cron.deny` file can access `crontab` commands. The system administrator can edit this file to add other users who are denied access to the `crontab` command. No default `cron.allow` file is supplied. This means that, after the Solaris software installation, all users (except the ones listed in the default `cron.deny` file) can access `crontab`. If you create a `cron.allow` file, only those users can access `crontab` commands.

Scheduling a Single System Event (`at`)

Objective:

Explain how to schedule an automatic one-time execution of a command using the command line.

The `at` command is used to schedule jobs for execution at a later time. Unlike `crontab`, which schedules a job to happen at regular intervals, a job submitted with `at` executes once, at the designated time.

To submit an `at` job, type `at` followed by the time that you would like the program to execute. You'll see the `at>` prompt displayed and it's here that you enter the `at` commands. When you are finished entering the `at` command, press `control-d` to exit the `at` prompt and submit the job as shown in the following example:

```
# at 07:45am today
at> who > /tmp/log
at> <Press Control-d>
job 912687240.a at Thu Jun 6 07:14:00
```

When you submit an `at` job, it is assigned a job identification number, which becomes its filename along with the .a extension. The file is stored in the `/var/spool/cron/atjobs` directory. In much the same way as it schedules `crontab` jobs, the `cron` daemon controls the scheduling of `at` files.

The command syntax for `at` is shown here:

```
# at [-m -l -r] <time> <date>
```

The `at` command is described in Table 5.15.

TABLE 5.15 `at` Command Syntax

Option	Description
`-m`	Sends you mail after the job is completed.
`-l`	Reports all jobs for the user.
`-r`	Removes a specified job.
`<time>`	The hour when you want to schedule the job. Add am or pm if you do not specify the hours according to a 24-hour clock. `midnight`, `noon`, and now are acceptable keywords. Minutes are optional.
`<date>`	The first three or more letters of a month, a day of the week, or the keywords `today` or `tomorrow`.

You can set up a file to control access to the `at` command, permitting only specified users to create, remove, or display queue information about their `at` jobs. The file that controls access to `at` is `/etc/cron.d/at.deny`. It consists of a list of usernames, one per line. The users listed in this file cannot access `at` commands. The default `at.deny` file, created during the SunOS software installation, contains the following usernames:

- `daemon`
- `bin`
- `smtp`
- `nuucp`
- `listen`
- `nobody`
- `noaccess`

With superuser privileges, you can edit this file to add other usernames whose `at` access you want to restrict.

Checking Jobs in Queue (`atq` and `at -l`)

To check your jobs that are waiting in the `at` queue, use the `atq` command. This command displays status information about the `at` jobs you created. Use the `atq` command to verify that you have created an `at` job. The `atq` command confirms that `at` jobs have been submitted to the queue, as shown in the following example:

```
# atq
```

The system responds with this:

```
Rank  Execution Date Owner  Job           Queue  Job Name
1st   Jun  6, 08:00  root   912690000.a     a    stdin
2nd   Jun  6, 08:05  root   912690300.a     a    stdin
```

Another way to check an `at` job is to issue the `at -l` command. This command shows the status information on all jobs submitted by a user, as shown in this example:

```
# at -l
```

The system responds with this:

```
user = root    912690000.a    Thu Jun  6 08:00:00
user = root    912690300.a    Thu Jun  6 08:05:00
```

Removing and Verifying Removal of `at` Jobs

To remove the `at` job from the queue before it is executed, type this:

```
# at -r [job-id]
```

`job-id` is the identification number of the job you want to remove.

Verify that the `at` job has been removed by using the `at -l` (or `atq`) command to display the jobs remaining in the `at` queue. The job whose identification number you specified should not appear. In the following example, you'll remove an `at` job that was scheduled to execute at 8:00 a.m. on June 6. First, check the `at` queue to locate the job identification number:

```
# at -l
```

The system responds with this:

```
user = root    912690000.a    Thu Jun  6 08:00:00
user = root    912690300.a    Thu Jun  6 08:05:00
```

Next, remove the job from the `at` queue:

```
# at -r 912690000.a
```

Finally, verify that this job has been removed from the queue:

```
# at -l
```

The system responds with this:

```
# user = root    912690300.a    Thu Jun  6 08:05:00
```

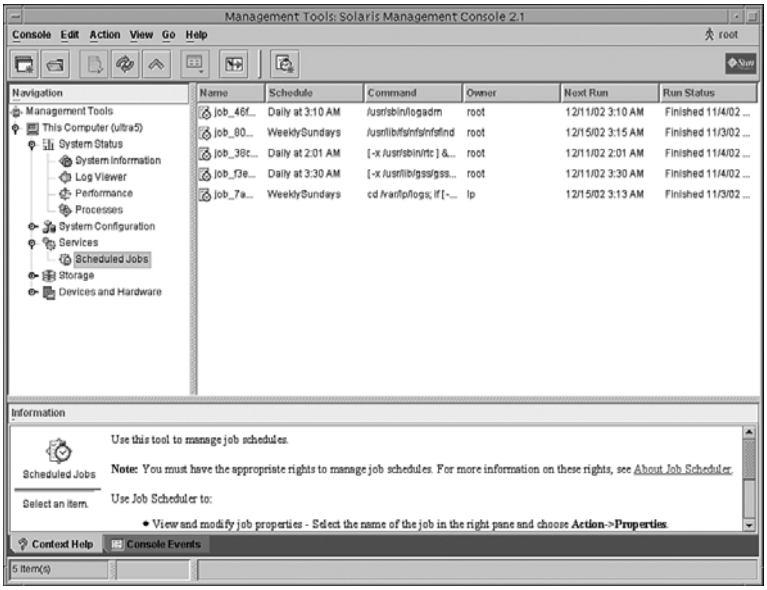
Job Scheduler

The Solaris Management Console (SMC) includes a graphical tool to create and schedule jobs on your system. You can use the Job Scheduler Tool to

- View and modify job properties
- Delete a job
- Add a scheduled job
- Enable or disable job logging

To open the Job Scheduler, follow the steps described in the "SMC Process Tool" section to start up the SMC using the `smc` command.

1. In the Navigation pane of the SMC Welcome window, open the Job Scheduler by clicking on the This Computer icon, then click on the Services icon, and then click on the Scheduled Jobs icon as shown in Figure 5.10.



Name	Schedule	Command	Owner	Next Run	Run Status
job_46f...	Daily at 3:10 AM	/usr/sbin/logadm	root	12/11/02 3:10 AM	Finished 11/4/02 ...
job_80...	WeeklySundays	/usr/lib/fs/nfs/nfsfind	root	12/15/02 3:15 AM	Finished 11/3/02 ...
job_38c...	Daily at 2:01 AM	[-x /usr/sbin/rtc] &...	root	12/11/02 2:01 AM	Finished 11/4/02 ...
job_f3e...	Daily at 3:30 AM	[-x /usr/lib/gss/gss...	root	12/11/02 3:30 AM	Finished 11/4/02 ...
job_7a...	WeeklySundays	cd /var/lp/logs; if [-...	lp	12/15/02 3:13 AM	Finished 11/3/02 ...

Use this tool to manage job schedules.

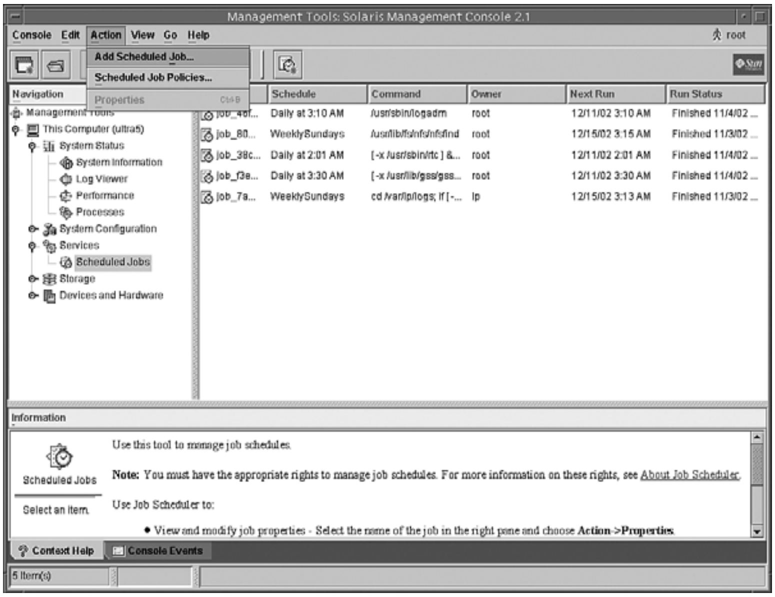
Note: You must have the appropriate rights to manage job schedules. For more information on these rights, see About Job Scheduler.

Use Job Scheduler to:

- View and modify job properties - Select the name of the job in the right pane and choose **Action->Properties**.

FIGURE 5.10 Opening the Job Scheduler.

2. You can add jobs to the `crontab` by selecting Action from the top toolbar as shown in Figure 5.11.



	Schedule	Command	Owner	Next Run	Run Status
job_46f...	Daily at 3:10 AM	/usr/sbin/logadm	root	12/11/02 3:10 AM	Finished 11/4/02 ...
job_80...	WeeklySundays	/usr/lib/fs/nfs/nfsfind	root	12/15/02 3:15 AM	Finished 11/3/02 ...
job_38c...	Daily at 2:01 AM	[-x /usr/sbin/rtc] &...	root	12/11/02 2:01 AM	Finished 11/4/02 ...
job_f3e...	Daily at 3:30 AM	[-x /usr/lib/gss/gss...	root	12/11/02 3:30 AM	Finished 11/4/02 ...
job_7a...	WeeklySundays	cd /var/lp/logs; if [-...	lp	12/15/02 3:13 AM	Finished 11/3/02 ...

Information

Scheduled Jobs

Select an item.

Use this tool to manage job schedules.

Note: You must have the appropriate rights to manage job schedules. For more information on these rights, see About Job Scheduler.

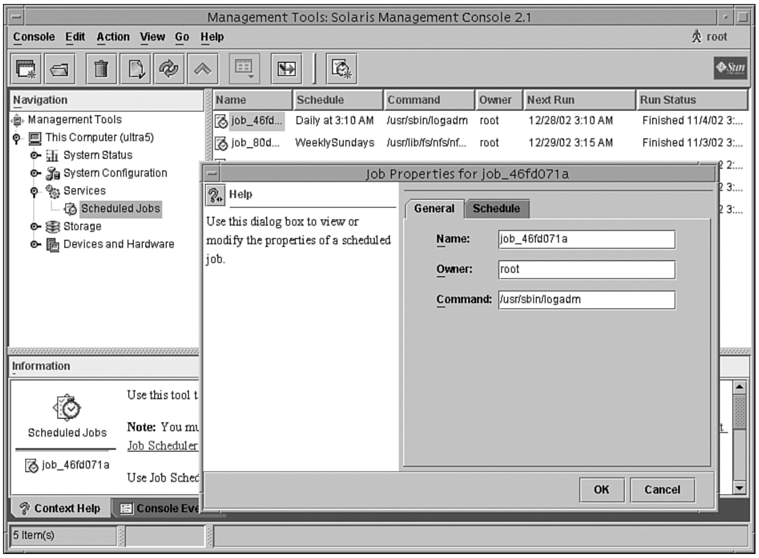
Use Job Scheduler to:

- View and modify job properties - Select the name of the job in the right pane and choose **Action->Properties**.

5 Item(s)

FIGURE 5.11 Adding a `cron` job.

3. Modify a `cron` job by double clicking on the job in the main window pane as shown in Figure 5.12.



Name	Schedule	Command	Owner	Next Run	Run Status
job_46fd...	Daily at 3:10 AM	/usr/sbin/logadm	root	12/28/02 3:10 AM	Finished 11/4/02 3:...
job_80d...	WeeklySundays	/usr/lib/fs/nfs/nf...	root	12/29/02 3:15 AM	Finished 11/3/02 3:...

Job Properties for job_46fd071a

Use this dialog box to view or modify the properties of a scheduled job.

Name: job_46fd071a

Owner: root

Command: /usr/sbin/logadm

Information

Scheduled Jobs

job_46fd071a

Use this tool t

Note: You mu
Job Scheduler

Use Job Sched

5 Item(s)

FIGURE 5.12 Modifiying a `cron` job.

Summary

This chapter described Solaris processes and the various Solaris utilities available to monitor them. Using commands such as `ps`, `prstat`, `pargs`, `sdtprocess`, and the SMC Process Tool, you can view all the attributes associated with a process. In addition, we described foreground and background jobs.

The concept of sending signals to a process was described. A signal is a message sent to a process to interrupt it and cause a response or action. You also learned how to send signals to processes to cause a response such as terminating a process.

Setting process priorities was described. We also described the concept of projects and tasks along with administrative commands used to administer them. The various commands, such as `nice` and `priocntl`, that are used to set and change process priorities were described. In addition, you learned how to use the `crontab` and `at` facilities. You can use these facilities to submit batch jobs and schedule processes to run when the system is less busy, to reduce the demand on resources such as the CPU and disks.

The system administrator needs to be aware of the processes that belong to each application. As users report problems, the system administrator can quickly locate the processes being used and look for irregularities. By keeping a close watch on system messages and processes, you'll become familiar with what is normal and what is abnormal. Don't wait for problems to happen—watch system messages and processes daily. Create shell scripts to watch processes for you and to look for irregularities in the system log files. By taking a proactive approach to system administration, you'll find problems before they affect the users.

In Chapter 6, "Managing the LP Print Service," we'll explore another topic that you'll need to become acquainted with—the LP Print Service, the facility responsible for printing within the Solaris environment.

Key Terms

- `at` command
- Process Manager GUI
- Child process
- `cron`
- `crontab`
- `crontab` file
- `mpstat`
- `ptree`
- `nice` command
- Parent process
- `pgrep` command
- `preap` command
- `priocntl` command
- Process
- Project (as it relates to process management)
- `prstat` command

- `ps` command
- Signals
- SMC Job Scheduler
- SMC Process Tool
- `time`
- Zombie process

Exercises

5.1 Displaying Process Information

In this exercise, you'll use the various utilities described in this chapter to display information about active processes.

Estimated time: 10 minutes

1. Log in as root into the Java Desktop Environment or CDE.
2. Open a new window and display the active processes using the `ps` command:

   ```
   # ps -ef
   ```

3. Open another new window and display the active processes using the `prstat` command:

   ```
   # prstat
   ```

   Notice how the `ps` command took a snapshot of the active processes, but the `prstat` command continues to update its display.
4. Type q to exit `prstat`.
5. Display the `dtlogin` process and all of its child processes. First obtain the PID of the `dtlogin` process with the `pgrep` command:

   ```
   # pgrep dtlogin
   ```

   Now use the `ptree` command with the PID of the `dtlogin` process to display the ancestry tree:

   ```
   # ptree <PID from dtlogin>
   ```

6. Now start the Process Manager.

   ```
   # sdtprocess &
   ```

   Notice how the window updates periodically.
7. In the sample field at the top of the window, change the sample period from 30 to 5 seconds.
8. Sort the processes by ID by clicking on the ID button in the header.

5.2 Using the Batch Process

In this exercise, you'll use `crontab` to configure a process to execute everyday at a specified time.

Estimated time: 10 minutes

1. Log in as root into a Java Desktop or CDE session.
2. Make sure your default shell editor is set to `vi` (EDITOR=vi;export EDITOR) before beginning this exercise.
3. Open a new window and edit the `crontab` entry.

   ```
   # crontab -e
   ```

4. Enter the following after the last line at the end of the file:

   ```
   0 11 * * * echo Hand in Timesheet > /dev/console
   ```

5. Save and close the file.

   Open a console window and at 11:00 a.m., you'll see the message `Hand in Timesheet` displayed.

Exam Questions

1. Which of the following commands finds all processes that have `dt` in the process argument string? Choose all that apply.

   ❍ **A.** `pgrep -l -f dt`

   ❍ **B.** `ps -ef dt`

   ❍ **C.** `ps -el dt`

   ❍ **D.** `ps -ef¦grep dt`

2. Which one of the following commands kills a process named `test`?

   ❍ **A.** `pkill test`

   ❍ **B.** `kill test`

   ❍ **C.** `ps -ef¦¦grep kill¦ kill -9`

   ❍ **D.** `kill -test`

3. Which commands display active system processes and update at a specified interval? Choose all that apply.

❍ **A.** `ps`

❍ **B.** `prstat`

❍ **C.** `sdtprocess`

❍ **D.** `ptree`

4. In output from the `ps` command, what does an `R` stand for in the `S` field?

❍ **A.** The process is on the run queue.

❍ **B.** The process is receiving input.

❍ **C.** It is a regular process.

❍ **D.** The process is sleeping, so it must be restarted.

5. In output from the `ps` command, which of the following does the `UID` field display?

❍ **A.** The parent process

❍ **B.** The process id

❍ **C.** The process owner

❍ **D.** The priority of the process

6. Which one of the following options to the `ps` command lists only processes for a particular user?

❍ **A.** `-P`

❍ **B.** `-f`

❍ **C.** `-l`

❍ **D.** `-u`

7. Which one of the following commands lists all processes running on the local system?

❍ **A.** `ps -e`

❍ **B.** `ps -a`

❍ **C.** `ps -f`

❍ **D.** `ps -t`

8. Which one of the following sends a terminate signal (signal 15) to a process with a PID of 2930?

❍ **A.** `kill 2930`

❍ **B.** `stop 2930`

❍ **C.** Ctrl+C

❍ **D.** `cancel 2930`

9. Which one of the following signals kills a process unconditionally?

❍ **A.** 9

❍ **B.** 0

❍ **C.** 15

❍ **D.** 1

10. Which of the following commands is used to change the priority on a process? Choose all that apply.

❍ **A.** `renice`

❍ **B.** `priocntl`

❍ **C.** `ps`

❍ **D.** `hup`

11. Which one of the following commands is issued to increase the priority of PID 8200?

❍ **A.** `renice -n -4 -p 8200`

❍ **B.** `nice -n -4 -p 8200`

❍ **C.** `nice -i 8200`

❍ **D.** `renice -I -p 8200`

12. Which utilities can be used to show the process ancestry tree? Choose all that apply.

❍ **A.** `ps`

❍ **B.** `ptree`

❍ **C.** `sdtprocess`

❍ **D.** `prstat`

13. Which of the following commands schedules a command to run once at a given time?

❍ **A.** `crontab`

❍ **B.** `priocntl`

❍ **C.** `at`

❍ **D.** `cron`

14. Which of the following commands show(s) the jobs queued up by the `at` command? Choose all that apply.

❍ **A.** `atq`

❍ **B.** `at -l`

❍ **C.** `ps`

❍ **D.** `crontab`

15. Which one of the following `crontab` entries instructs the system to run `logchecker` at 3:10 on Sunday and Thursday nights?

❍ **A.** `0 4 * * 10,3 /etc/cron.d/logchecker`

❍ **B.** `10 3 * * 0,4 /etc/cron.d/logchecker`

❍ **C.** `* 10 3 0,4 /etc/cron.d/logchecker`

❍ **D.** `10 3 * * 0-4 /etc/cron.d/logchecker`

16. Which one of the following logs keeps a record of all `cron` activity?

❍ **A.** `/var/cron/log`

❍ **B.** `/var/spool/cron/log`

❍ **C.** `/var/adm/cron`

❍ **D.** `/var/adm/messages`

17. A user wants to execute a command later today, after leaving work. Which one of the following commands will allow him to do this?

❍ **A.** `runat`

❍ **B.** `at`

❍ **C.** `submit`

❍ **D.** None of the above

18. You've added the user name `bcalkins` to the `/etc/cron.d/cron.allow` file. You've removed the name `bcalkins` from the `/etc/cron.d/cron.deny` file. Which statement is true regarding `crontab`?

   ❍ **A.** `bcalkins` cannot create `crontab` entries.

   ❍ **B.** `bcalkins` can create `crontab` entries.

   ❍ **C.** Only root can create `crontab` entries.

   ❍ **D.** No one can create `crontab` entries.

Answers to Exam Questions

1. **A, D.** Use the `pgrep` and `ps` commands to view processes running on your system. The commands `pgrep -l -f  dt` and `ps -ef¦grep dt` find all the processes that have `dt` in the process argument string and display them.
2. **A.** The command `pkill test` kills a process named `test`.
3. **B, C.** The `prstat` and `sdtprocess` commands display active system processes and can be configured to update at a specified interval.
4. **A.** In output from the `ps` command, the `R` in the `S` field means that the process is on the run queue.
5. **C.** In output from the `ps` command, the `UID` field displays the process owner.
6. **D.** The `-u` option to the `ps` command lists only processes for a particular user.
7. **A.** The `-e` option to the `ps` command lists all processes currently running on the system. The other options only list processes for the local user.
8. **A.** The command `kill 2930` sends a terminate signal (signal 15) to a process with a PID of 2930.
9. **A.** Signal 9 stops a process unconditionally.
10. **A, B.** The commands `renice` and `priocntl` are used to change the priority on a process.
11. **A.** The `renice -n -4 -p 8200` command is issued to increase the priority of a process with a PID of 8200.
12. **B, C.** The utilities `ptree` and `sdtprocess` are used to show the process ancestry tree.
13. **C.** The `at` command schedules a command to run once at a given time.
14. **A, B.** The `atq` and `at -l` commands show the jobs queued up by the `at` command.
15. **B.** The `crontab` entry `10 3 * * 0,4 /etc/cron.d/logchecker` instructs the system to run `logchecker` at 3:10 on Sunday and Thursday nights.

16. **A.** The log file named `/var/cron/log` keeps a record of all `cron` activity.
17. **B.** Use the `at` command to execute a command or script at a later time.
18. **B.** Users can manage jobs if their name appears in the `/etc/cron.d/cron.allow` file and does not appear in the `/etc/cron.d/cron.deny` file.

Suggested Reading and Resources

1. Calkins, Bill. *Inside Solaris 9*. New Riders Publishing. November 2002.

Index