9.8 nice: JOB EXECUTION WITH LOW PRIORITY

Processes in the UNIX system are usually executed with equal priority. This is not always desirable since high-priority jobs must be completed at the earliest. UNIX offers the **nice** command, which is used with the & operator to *reduce* the priority of jobs. More important jobs can then have greater access to the system resources (being "nice" to your neighbors).

To run a job with a low priority, the command name should be prefixed with **nice**:

```
nice wc -l uxmanual
```

or better still with

```
nice wc -l uxmanual &
```

nice is a built-in command in the C shell. nice values are system-dependent and typically range from 1 to 19. Commands execute with a nice value that is generally in the middle of the range—usually 10. A higher nice value implies a lower priority. **nice** reduces the priority of any process,

thereby raising its nice value. You can also specify the nice value explicitly with the -n option:

```
nice -n 5 wc -l uxmanual &                    Nice value increased by 5 units
```

A nonprivileged user can't increase the priority of a process; that power is reserved for the superuser. The nice value is displayed with the **ps -o nice** command.

9.9 KILLING PROCESSES WITH SIGNALS

The UNIX system often requires to communicate the occurrence of an event to a process. This is done by sending a **signal** to the process. Each signal is identified by a number and is designed to perform a specific function. Because the same signal number may represent two different signals on two different machines, signals are better represented by their symbolic names having the SIG prefix. They can be generated from the keyboard or by the **kill** command, which we'll be discussing soon.

If a program is running longer than you anticipated and you want to terminate it, you normally press the interrupt key. This sends the process the SIGINT signal (numbered 2). The default action of this signal is to kill the process. A process may also ignore a signal or execute some user-defined code written to handle that signal. Chapter 24 discusses all of these options in detail; in this chapter we are concerned with a signal's default action only.

Irrespective of what you do, there are two signals that a process can't ignore or run user-defined code to handle it. They are the SIGKILL and SIGSTOP signals which always perform the default action associated with them. We'll now learn to use the **kill** command to send specific signals to processes.

9.9.1 kill: Premature Termination of a Process

The **kill** command sends a signal, *usually* with the intention of killing one or more processes. **kill** is an internal command in most shells; the external **/bin/kill** is executed only when the shell lacks the kill capability. The command uses one or more PIDs as its arguments, and by default uses the SIGTERM (15) signal. Thus,

```
kill 105                                   It's like using kill -s TERM 105
```

terminates the job having PID 105. To facilitate premature termination, the & operator displays the PID of the process that's run in the background. If you don't remember the PID, use **ps** to know that and then use **kill**.

If you run more than one job—either in the background or in different windows in the X Window system, you can kill them all with a single **kill** statement. Just specify all their PIDs with **kill**:

```
kill 121 122 125 132 138 144
```

If all these processes have the same parent, you may simply kill the parent in order to kill all its children. However, when you use **nohup** with a set of commands and log out, you can't kill the parent as **init** acquires their parentage. You then have to kill the processes individually because you can't kill **init**.

12.2 pr: PAGINATING FILES

The **pr** command prepares a file for printing by adding suitable headers, footers and formatted text. A simple invocation of the command is to use it with a filename as argument:

```
$ pr dept.lst

May 06 10:38 1997  dept.lst Page 1

01:accounts:6213
02:admin:5423
03:marketing:6521
04:personnel:2365
05:production:9876
06:sales:1006

...blank lines...
```

These six lines are the original contents of `dept.lst` *shown in Section 5.1*

pr adds five lines of margin at the top and five at the bottom. The lower portion of the page has not been shown in the examples for reasons of economy. The header shows the date and time of last modification of the file along with the filename and page number.