# 7

# *Multitasking*

*In this chapter:*
- *Running a Command in the Background*
- *Checking on a Process*
- *Cancelling a Process*

Unix can do many jobs at once, dividing the processor's time between the tasks so quickly that it looks as if everything is running at the same time. This is called *multitasking*.

With a window system, you can have many applications running at the same time, with many windows open. But most Unix systems also let you run more than one program inside the *same terminal*. This is called *job control*. It gives some of the benefits of window systems to users who don't have windows. But, even if you're using a window system, you may want to use job control to do several things inside the same terminal window. For instance, you may prefer to do most of your work from one terminal window, instead of covering your desktop with multiple windows.

Why else would you want job control? Suppose you're running a program that will take a long time to process. On a single-task operating system such as MS-DOS, you would enter the command and wait for the system prompt to return, telling you that you could enter a new command. In Unix, however, you can enter new commands in the "foreground" while one or more programs are still running in the "background."

When you enter a command as a background process, the shell prompt reappears immediately so that you can enter a new command. The original program will still run in the background, but you can use the system to do other things during that time. Depending on your system and your shell, you may even be able to log off and let the background process run to completion.

# *Running a Command in the Background*

Running a program as a background process is most often done to free a terminal when you know the program will take a long time to run. It's used whenever you want to launch a new window program from an existing terminal window—so that you can keep working in the existing terminal, as well as in the new window.

To run a program in the background, add the "&" character at the end of the command line before you press the RETURN key. The shell then assigns and displays a process ID number for the program:

```
$ sort bigfile > bigfile.sort &
[1] 29890
$
```

(Sorting is a good example because it can take a while to sort huge files, so users often do it in the background.)

The process ID (PID) for this program is 29890. The PID is useful when you want to check the status of a background process, or if you need to, cancel it. You don't need to remember the PID, because there are Unix commands (explained in later sections of this chapter) to check on the processes you have running. Some shells write a status line to your screen when the background process finishes.

Here's another example. If you're using a terminal window, and you'd like to open another terminal window, you can probably click a button or choose a menu item to do that. But, if you occasionally want to specify command-line options for that new window, it's much easier to type the options on a command line in an existing window. (Most menus and buttons don't give you the flexibility to choose options each time you open a new window.) For instance, by default, an **xterm** window saves 64 lines of your previous work in its "scrollback buffer." If you'll be doing a lot of work that you'll want to review with the scrollbar, you might want to open a new window with a 2000-line scrollback buffer. You could enter the following command in an existing **xterm** window:

```
$ xterm -sl 2000 &
[1] 19283
```

A new **xterm** window should pop open—where you'll be able to scroll almost forever.

In the C shell, you can put an entire sequence of commands separated by semicolons (;) into the background by putting an ampersand at the end of the entire command line. In other shells, enclose the command sequence in parentheses before adding the ampersand. For instance, you might want to sort a file, then print it after **sort** finishes. The syntax that works on all shells is:

(*command1; command2*) &

The examples above work on all shells. On many systems, the shells have the feature we mentioned earlier called *job control*. You can use the *suspend character* (usually CTRL-Z) to suspend a program running in the foreground. The program pauses and you get a new shell prompt. You can then do anything else you like, including putting the suspended program into the background using the **bg** command. The **fg** command brings a suspended or background process to the foreground.

For example, you might start **sort** running on a big file, and, after a minute, want to send email. Stop **sort**, then put it in the background. The shell prints a message, then another shell prompt. Send mail while **sort** runs.

```
$ sort hugefile1 hugefile2 > sorted
        ...time goes by...
CTRL-Z  Stopped
$ bg
[1]    sort hugefile1 hugefile2 > sorted &
$ mail eduardo@nacional.cl
        ...
```

# Checking on a Process

If a background process takes too long, or you change your mind and want to stop a process, you can check the status of the process and even cancel it.

## ps

When you enter the command **ps**, you can see how long a process has been running, the process ID of the background process and the terminal from which it was run. The **tty** program shows the name of the terminal where it's running; this is especially helpful when you're using a window system or you're logged into multiple terminals. Example 7-1 shows this in more detail.

*Example 7-1. Output of ps and tty programs*

```
$ ps
   PID TTY      TIME CMD
 27285 pts/3    0:01 csh
 27285 pts/3    0:01 ps
 29771 pts/2    0:00 csh
 29792 pts/2    0:54 sort
$ tty
/dev/pts/3
```

In its basic form, **ps** lists the following:

*Process ID (PID)*
: A unique number assigned by Unix to the process.

*Terminal name (TTY)*
: The Unix name for the terminal from which the process was started.

*Run time (TIME)*
: The amount of computer time (in minutes and seconds) that the process has used.

*Command (CMD)*
: The name of the process.

In a window system, each terminal window has its own terminal name. Example 7-1 shows processes running on two terminals: *pts/3* and *pts/2*. Some versions of **ps** list only the processes on the same terminal where you run **ps**; other versions list processes on all terminals where you're logged in. If you have more than one terminal window open, but all the entries in the TTY column show the same terminal name, try typing either "**ps x**" or "**ps -u** *username*", where *username* is your username. If you need to find out the name of a particular terminal, run the **tty** program from a shell prompt in that window, as shown in Example 7-1.

While using a window system, you may see quite a few processes you don't recognize; they're probably helping the window manager do its job. You may also see the names of any other programs running in the background and the name of your shell's process (**sh**, **csh**, and so on)—although different versions of **ps** may show fewer processes by default. **ps** may or may not list its own process.

You should be aware that there are two types of programs on Unix systems: directly executable programs and interpreted programs. Directly executable programs are written in a programming language such as C or Pascal and stored in a file that the system can read directly. Interpreted programs, such as shell scripts and Perl scripts, are sequences of

commands that are read by an interpreter program. If you execute an interpreted program, you will see an additional command (such as **perl**, **sh**, or **csh**) in the **ps** listing, as well as any Unix commands that the interpreter is executing now.

Shells with job control have a command called **jobs** which lists background processes started from that shell. As mentioned earlier, there are commands to change the foreground/background status of jobs. There are other job control commands as well. See the references in the section "Documentation" in Chapter 8.

# Cancelling a Process

You may decide that you shouldn't have put a process in the background. Or you decide that the process is taking too long to execute. You can cancel a background process if you know its process ID.

## kill

The **kill** command aborts a process. The command's format is:

**kill** *PID(s)*

**kill** terminates the designated process IDs (shown under the PID heading in the **ps** listing). If you do not know the process ID, do a **ps** first to display the status of your processes.

In the following example, the "**sleep** *n*" command simply causes a process to "go to sleep" for *n* number of seconds. We enter two commands, **sleep** and **who**, on the same line, as a background process.

```
$ (sleep 60; who)&
[1] 21087
$ ps
 PID    TTY   TIME  COMMAND
20055    4    0:10  sh
21087    4    0:01  sh
21088    4    0:00  sleep
21089    4    0:02  ps
$ kill 21088
[1]+  Terminated              sleep 60
$ tom     tty2   Aug 30  11:27
grace   tty4   Aug 30  12:24
tim     tty5   Aug 30  07:52
dale    tty7   Aug 30  14:34
```

We decided that 60 seconds was too long to wait for the output of **who**. The **ps** listing showed that **sleep** had the process ID number 21088, so we

used this PID to kill the **sleep** process. You should see a message like "terminated" or "killed"; if you don't, use another **ps** command to be sure the process has been killed.

The **who** program is executed immediately, since it is no longer waiting on **sleep**; it lists the users logged into the system.

### *Problem checklist*

*The process didn't die when I told it to.*

Some processes can be hard to kill. If a normal kill of these processes is not working, enter "**kill -9** *PID*". This is a sure kill and can destroy almost anything, including the shell that is interpreting it.

In addition, if you've run an interpreted program (such as a shell script), you may not be able to kill all dependent processes by killing the interpreter process that got it all started; you may need to kill them individually. However, killing a process that is feeding data into a pipe generally kills any processes receiving that data.