9.1 PROCESS BASICS

A **process** is simply an instance of a running program. A process is said to be **born** when the program starts execution and remains alive as long as the program is active. After execution is complete, the process is said to **die**. A process also has a name, usually the name of the program being executed. For example, when you execute the **grep** command, a process named **grep** is created. However, a process can't be considered synonymous with a *program*. When two users run the same program, there's one program on disk but two processes in memory.

The kernel is responsible for the management of processes. It determines the time and priorities that are allocated to processes so that multiple processes are able to share CPU resources. It provides a mechanism by which a process is able to execute for a finite period of time and then relinquish control to another process. The kernel has to sometimes store *pages* (sections) of these processes in the swap area of the disk before calling them again for running. All this happens more than once a second, making the user oblivious to the switching process.

Files have attributes and so do processes. Some attributes of every process are maintained by the kernel in memory in a separate structure called the **process table**. You could say that the process table is the inode for processes. Two important attributes of a process are:

- *The Process-id (PID)* Each process is uniquely identified by a unique integer called the **Process-id (PID)** that is allotted by the kernel when the process is born. We need this PID to control a process, for instance, to kill it.
- *The Parent PID (PPID)* The PID of the parent is also available as a process attribute. When several processes have the same PPID, it often makes sense to kill the parent rather than all its children separately.

The other attributes are inherited by the child from its parent and are discussed in Section 9.4.

Things do go wrong at times. A process may go berserk and multiply rapidly, bringing the system to a complete standstill. A process may not complete in the expected time, and you may decide to suspend it, move it to the background, or even kill it. UNIX provides us with the tools to understand the process hierarchy and also control these processes.

9.6 PROCESS STATES AND ZOMBIES

At any instant of time, a process is in a particular **state**. A process after creation is in the *runnable* state before it actually runs (state *running*). While the process is running, it may be invoke a disk I/O operation when it has nothing to do except wait for the I/O to complete. The process then moves to the *sleeping* state to be woken up when the I/O operation is over. A process can also be *suspended* by pressing a key (usually, *[Ctrl-z]*). Processes whose parents don't wait for their death move to the *zombie* state.

When a process dies, it immediately moves to the **zombie** state. It remains in this state until the parent picks up the child's exit status (the reason for waiting) from the process table. When that is done, the kernel frees the process table entry. A zombie is a harmless dead child but you can't kill it.

It's also possible for the parent itself to die before the child dies. The child then becomes an **orphan** and the kernel makes **init** the parent of all orphans. When this adopted child dies, **init** waits for its death.

9.7 RUNNING JOBS IN BACKGROUND

A multitasking system lets a user do more than one job at a time. Since there can be only one job in the *foreground*, the rest of the jobs have to run in the *background*. There are two ways of doing this—with the shell's & operator and the **nohup** command. The latter permits you to log out while your jobs are running, but the former doesn't allow that (except in the C shell and Bash).

9.7.1 &: No Logging Out

The & is the shell's operator used to run a process in the background. The parent in this case doesn't wait for the child's death. Just terminate the command line with an &; the command wil run in the background:

```
$ sort -o emp.lst emp.lst &
550                                        The job's PID
```