【OS】Day9

【Ch2】Process API(3)

5.5 Process Control and Users

Beyond `fork(),` `exec()`, and `wait()`, there are a lot of other interfaces for interacting with processes in UNIX systems.

For example, the `kill()` system call is used to send signals to a process, including directives to pause, die, and other useful imperatives.

For convenience, in most UNIX shells, certain keystroke combinations are configures to deliver a specific signal to the currently running process; for example, control-c sends a `SIGINT`(interrupt) to the process(normally terminating it) and control-z sends a `SIGTSTOP`(stop) signal thus pausing the process in mid-execution.

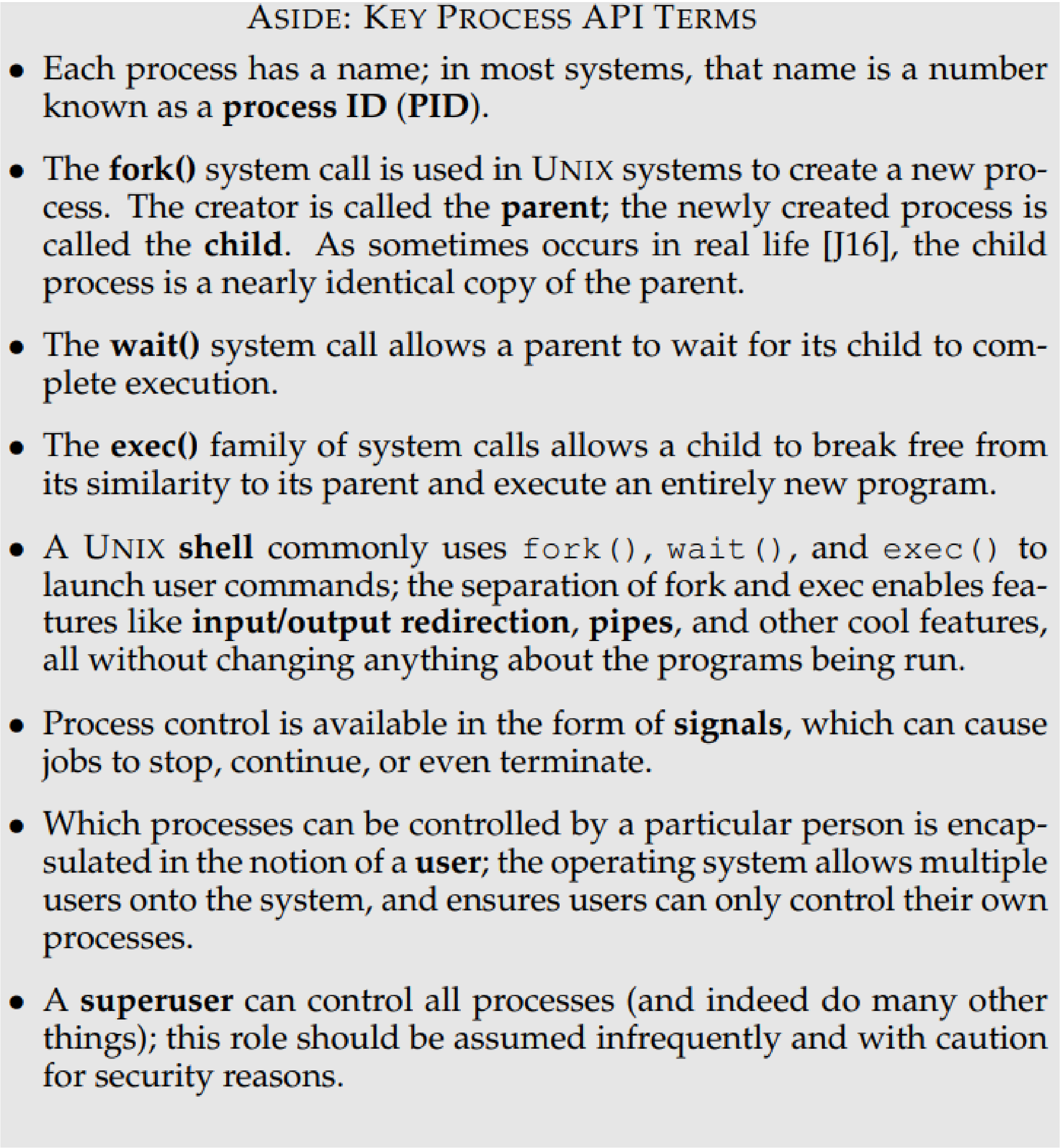
5.6 Useful Tools

There are many command-line tools that are useful as well. For example, using the `ps` command allows you to see which processes are running.

The tool `top` is also quite helpful, as it displays the processes of the system and how much CPU and other resources they are eating up.

The command `kill` can be used to send arbitrary signals to processes, as can the slightly more user friendly killall.

5.7 Summary



ASIDE: KEY PROCESS API TERMS

- Each process has a name; in most systems, that name is a number known as a **process ID (PID)**.

- The **fork()** system call is used in UNIX systems to create a new process. The creator is called the **parent**; the newly created process is called the **child**. As sometimes occurs in real life [J16], the child process is a nearly identical copy of the parent.

- The **wait()** system call allows a parent to wait for its child to complete execution.

- The **exec()** family of system calls allows a child to break free from its similarity to its parent and execute an entirely new program.

- A UNIX **shell** commonly uses `fork()`, `wait()`, and `exec()` to launch user commands; the separation of fork and exec enables features like **input/output redirection**, **pipes**, and other cool features, all without changing anything about the programs being run.

- Process control is available in the form of **signals**, which can cause jobs to stop, continue, or even terminate.

- Which processes can be controlled by a particular person is encapsulated in the notion of a **user**; the operating system allows multiple users onto the system, and ensures users can only control their own processes.

- A **superuser** can control all processes (and indeed do many other things); this role should be assumed infrequently and with caution for security reasons.