4.1 COMPONENTS OF A PROCESS

A process consists of an address space and a set of data structures within the kernel. The address space is a set of memory pages1 that the kernel has marked for the process’s use. It contains the code and libraries that the process is executing, the process’s variables, its stacks, and various extra information needed by the kernel while the process is running. Because UNIX supports virtual memory, there is not necessarily a correlation between a page’s location within an address space and its location inside the machine’s physical memory or swap space.

The kernel’s internal data structures record various pieces of information about each process. Some of the more important of these are:

•   The process’s address space map

•   The current status of the process (sleeping, stopped, runnable, etc.)

•   The execution priority of the process

•   Information about the resources the process has used (for accounting)

•   The process’s signal mask (a record of which signals are blocked)

•   The owner of the process

IGNALS

Signals are process-level interrupt requests. About thirty different kinds are defined, and they’re used in a variety of ways:

• They can be sent among processes as a means of communication.

• They can be sent by the terminal driver to kill, interrupt, or suspend processes when special keys such as <Control-C> and <Control-Z> are typed.4

• They can be sent by the administrator (with kill) to achieve various results.

• They can be sent by the kernel when a process commits an infraction such as division by zero.

When a signal is received, one of two things can happen. If the receiving process has designated a handler routine for that particular signal, the handler is called with information about the context in which the signal was delivered. Otherwise, the kernel takes some default action on behalf of the process. The default action varies from signal to signal. Many signals terminate the process; some also generate a core dump.

A core dump is a memory image of a process that can be used for debugging.

Specifying a handler routine for a signal within a program is referred to as “catching” the signal. When thhandler completes, execution restarts from the point at which the signal was received.

To prevent signals from arriving, programs can request that they be either ignored or blocked. A signal that is ignored is simply discarded and has no effect on the process. A blocked signal is queued for delivery, but the kernel doesn’t require the process to act on it until the signal has been explicitly unblocked. The handler for a newly unblocked signal is called only once, even if the signal was received several times while reception was blocked.

Table 4.1 lists the signals that all administrators should know. The uppercase convention for signal names derives from C language tradition. You might also sometimes see signal names written with a SIG prefix (e.g., SIGHUP) for similar reasons.

Table 4.1 UNIX signals that every administrator should know

image