第十一章 进程

Modern operating systems are usually multitasking, meaning that they create the illusion of doing more than one thing at once by rapidly switching from one executing program to another. The Linux kernel manages this through the use of processes. Processes are how Linux organizes the different programs waiting for their turn at the CPU.

通常，现在的操作系统都支持多任务，意味着操作系统（给用户）造成了一种假象，(让用户觉得) 它同时能够做多件事情，事实上，它是快速地轮换执行这些任务的。Linux 内核通过使用进程，来 管理多任务。通过进程，Linux 安排不同的程序等待使用 CPU。

Sometimes a computer will become sluggish or an application will stop responding. In this chapter, we will look at some of the tools available at the command line that let us examine what programs are doing, and how to terminate processes that are misbehaving.

有时候，计算机变得呆滞，运行缓慢，或者一个应用程序停止响应。在这一章中，我们将看一些 可用的命令行工具，这些工具帮助我们查看程序的执行状态，以及怎样终止行为不当的进程。

This chapter will introduce the following commands:

这一章将介绍以下命令：

* ps – Report a snapshot of current processes
* top – Display tasks
* jobs – List active jobs
* bg – Place a job in the background
* fg – Place a job in the foreground
* kill – Send a signal to a process
* killall – Kill processes by name
* shutdown – Shutdown or reboot the system
* ps – 报告当前进程快照
* top – 显示任务
* jobs – 列出活跃的任务
* bg – 把一个任务放到后台执行
* fg – 把一个任务放到前台执行
* kill – 给一个进程发送信号
* killall – 杀死指定名字的进程
* shutdown – 关机或重启系统

**进程是怎样工作的**

When a system starts up, the kernel initiates a few of its own activities as processes and launches a program called init. init, in turn, runs a series of shell scripts (located in /etc) called init scripts, which start all the system services. Many of these services are implemented as daemon programs, programs that just sit in the background and do their thing without having any user interface. So even if we are not logged in, the system is at least a little busy performing routine stuff.

当系统启动的时候，内核先把一些它自己的程序初始化为进程，然后运行一个叫做 init 的程序。init， 依次地，再运行一系列的称为 init 脚本的 shell 脚本（位于/etc），它们可以启动所有的系统服务。 其中许多系统服务以守护（daemon）程序的形式实现，守护程序仅在后台运行，没有任何用户接口。 这样，即使我们没有登录系统，至少系统也在忙于执行一些例行事务。

The fact that a program can launch other programs is expressed in the process scheme as a parent process producing a child process.

一个程序可以发动另一个程序，这个事实在进程方案中，表述为一个父进程创建了一个子进程。

The kernel maintains information about each process to help keep things organized. For example, each process is assigned a number called a process ID or PID. PIDs are assigned in ascending order, with init always getting PID 1. The kernel also keeps track of the memory assigned to each process, as well as the processes’ readiness to resume execution. Like files, processes also have owners and user IDs, effective user IDs, etc.

内核维护每个进程的信息，以此来保持事情有序。例如，系统分配给每个进程一个数字，这个数字叫做 进程 ID 或 PID。PID 号按升序分配，init 进程的 PID 总是1。内核也对分配给每个进程的内存进行跟踪。 像文件一样，进程也有所有者和用户 ID，有效用户 ID，等等。

**查看进程**

The most commonly used command to view processes (there are several) is ps. The ps program has a lot of options, but in it simplest form it is used like this:

查看进程，最常使用地命令（有几个命令）是 ps。ps 程序有许多选项，它最简单地使用形式是这样的：

[me@linuxbox ~]$ ps

PID TTY TIME CMD

5198 pts/1 00:00:00 bash

10129 pts/1 00:00:00 ps

The result in this example lists two processes, process 5198 and process 10129, which are bash and ps respectively. As we can see, by default, ps doesn’t show us very much, just the processes associated with the current terminal session. To see more, we need to add some options, but before we do that, let’s look at the other fields produced by ps. TTY is short for “Teletype,” and refers to the controlling terminal for the process. Unix is showing its age here. The TIME field is the amount of CPU time consumed by the process. As we can see, neither process makes the computer work very hard.

上例中，列出了两个进程，进程 5198 和进程 10129，各自代表命令 bash 和 ps。正如我们所看到的， 默认情况下，ps 不会显示很多进程信息，只是列出与当前终端会话相关的进程。为了得到更多信息， 我们需要加上一些选项，但是在这样做之前，我们先看一下 ps 命令运行结果的其它字段。 TTY 是 “Teletype” 的简写，是指进程的控制终端。这里，Unix 展示它的年龄。TIME 字段表示 进程所消耗的 CPU 时间数量。正如我们所看到的，这两个进程使计算机工作起来很轻松。

If we add an option, we can get a bigger picture of what the system is doing:

如果给 ps 命令加上选项，我们可以得到更多关于系统运行状态的信息：

[me@linuxbox ~]$ ps x

PID TTY STAT TIME COMMAND

2799 ? Ssl 0:00 /usr/libexec/bonobo-activation-server –ac

2820 ? Sl 0:01 /usr/libexec/evolution-data-server-1.10 --

and many more...

Adding the “x” option (note that there is no leading dash) tells ps to show all of our processes regardless of what terminal (if any) they are controlled by. The presence of a “?” in the TTY column indicates no controlling terminal. Using this option, we see a list of every process that we own.

加上 “x” 选项（注意没有开头的 “-“ 字符），告诉 ps 命令，展示所有进程，不管它们由什么 终端（如果有的话）控制。在 TTY 一栏中出现的 “?” ，表示没有控制终端。使用这个 “x” 选项，可以 看到我们所拥有的每个进程的信息。

Since the system is running a lot of processes, ps produces a long list. It is often helpful to pipe the output from ps into less for easier viewing. Some option combinations also produce long lines of output, so maximizing the terminal emulator window may be a good idea, too.

因为系统中正运行着许多进程，所以 ps 命令的输出结果很长。这经常很有帮助，要是把 ps 的输出结果 管道到 less 命令，借助 less 工具，更容易浏览。一些选项组合也会产生很长的输出结果，所以最大化 终端仿真器窗口，也是一个好主意。

A new column titled STAT has been added to the output. STAT is short for “state” and reveals the current status of the process:

输出结果中，新添加了一栏，标题为 STAT 。STAT 是 “state” 的简写，它揭示了进程当前状态：

| *Table 11-1: Process States* | |
| --- | --- |
| State | Meaning |
| R | Running. This means that the process is running or ready to run. |
| S | Sleeping. A process is not running; rather, it is waiting for an event, such as a keystroke or network packet. |
| D | Uninterruptible Sleep. Process is waiting for I/O such as a disk drive. |
| T | Stopped. Process has been instructed to stop. More on this later. |
| Z | A defunct or “zombie” process. This is a child process that has terminated, but has not been cleaned up by its parent. |
| < | A high priority process. It's possible to grant more importance to a process, giving it more time on the CPU. This property of a process is called niceness. A process with high priority is said to be less nice because it's taking more of the CPU's time, which leaves less for everybody else. |
| N | A low priority process. A process with low priority (a “nice” process) will only get processor time after other processes with higher priority have been serviced. |

| *表11-1: 进程状态* | |
| --- | --- |
| 状态 | 意义 |
| R | 运行。这意味着，进程正在运行或准备运行。 |
| S | 正在睡眠。 进程没有运行，而是，正在等待一个事件， 比如说，一个按键或者网络数据包。 |
| D | 不可中断睡眠。进程正在等待 I/O，比方说，一个磁盘驱动器的 I/O。 |
| T | 已停止. 已经指示进程停止运行。稍后介绍更多。 |
| Z | 一个死进程或“僵尸”进程。这是一个已经终止的子进程，但是它的父进程还没有清空它。 （父进程没有把子进程从进程表中删除） |
| < | 一个高优先级进程。这可能会授予一个进程更多重要的资源，给它更多的 CPU 时间。 进程的这种属性叫做 niceness。具有高优先级的进程据说是不好的（less nice）， 因为它占用了比较多的 CPU 时间，这样就给其它进程留下很少时间。 |
| N | 低优先级进程。 一个低优先级进程（一个“好”进程）只有当其它高优先级进程执行之后，才会得到处理器时间。 |

The process state may be followed by other characters. These indicate various exotic process characteristics. See the ps man page for more detail.

进程状态信息之后，可能还跟随其他的字符。这表示各种外来进程的特性。详细信息请看 ps 手册页。

Another popular set of options is “aux” (without a leading dash). This gives us even more information:

另一个流行的选项组合是 “aux”（不带开头的”-“字符）。这会给我们更多信息：

[me@linuxbox ~]$ ps aux

USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND

root 1 0.0 0.0 2136 644 ? Ss Mar05 0:31 init

root 2 0.0 0.0 0 0 ? S&lt; Mar05 0:00 [kt]

and many more...

This set of options displays the processes belonging to every user. Using the options without the leading dash invokes the command with “BSD style” behavior. The Linux version of ps can emulate the behavior of the ps program found in several different Unix implementations. With these options, we get these additional columns:

这个选项组合，能够显示属于每个用户的进程信息。使用这个选项，可以唤醒 “BSD 风格” 的输出结果。 Linux 版本的 ps 命令，可以模拟几个不同 Unix 版本中的 ps 程序的行为。通过这些选项，我们得到 这些额外的列。

| *Table 11-2: BSD Style ps Column Headers* | |
| --- | --- |
| Header | Meaning |
| USER | User ID. This is the owner of the process. |
| %CPU | CPU usage in percent |
| %MEM | Memory usage in percent |
| VSZ | Virtual memory size |
| RSS | Resident Set Size. The amount of physical memory (RAM) the process is using in kilobytes. |
| START | Time when the process started. For values over twenty four hours, a date is used. |

| *表11-2: BSD 风格的 ps 命令列标题* | |
| --- | --- |
| 标题 | 意思 |
| USER | 用户 ID. 进程的所有者。 |
| %CPU | 以百分比表示的 CPU 使用率 |
| %MEM | 以百分比表示的内存使用率 |
| VSZ | 虚拟内存大小 |
| RSS | 进程占用的物理内存的大小，以千字节为单位。 |
| START | 进程运行的起始时间。若超过24小时，则用天表示。 |

**用 top 命令动态查看进程**

While the ps command can reveal a lot about what the machine is doing, it provides only a snapshot of the machine’s state at the moment the ps command is executed. To see a more dynamic view of the machine’s activity, we use the top command:

虽然 ps 命令能够展示许多计算机运行状态的信息，但是它只是提供，ps 命令执行时刻的机器状态快照。 为了看到更多动态的信息，我们使用 top 命令：

[me@linuxbox ~]$ top

The top program displays a continuously updating (by default, every 3 seconds) display of the system processes listed in order of process activity. The name “top” comes from the fact that the top program is used to see the “top” processes on the system. The top display consists of two parts: a system summary at the top of the display, followed by a table of processes sorted by CPU activity:

top 程序连续显示系统进程更新的信息（默认情况下，每三秒钟更新一次），”top”这个名字 来源于这个事实，top 程序是用来查看系统中“顶端”进程的。top 显示结果由两部分组成： 最上面是系统概要，下面是进程列表，以 CPU 的使用率排序。

top - 14:59:20 up 6:30, 2 users, load average: 0.07, 0.02, 0.00

Tasks: 109 total, 1 running, 106 sleeping, 0 stopped, 2 zombie

Cpu(s): 0.7%us, 1.0%sy, 0.0%ni, 98.3%id, 0.0%wa, 0.0%hi, 0.0%si

Mem: 319496k total, 314860k used, 4636k free, 19392k buff

Swap: 875500k total, 149128k used, 726372k free, 114676k cach

PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND

6244 me 39 19 31752 3124 2188 S 6.3 1.0 16:24.42 trackerd

....

The system summary contains a lot of good stuff. Here’s a rundown:

其中系统概要包含许多有用信息。下表是对系统概要的说明：

| *Table 11-3: top Information Fields* | | |
| --- | --- | --- |
| Row | Field | Meaning |
| 1 | top | Name of the program |
|  | 14:59:20 | Current time of day. |
|  | up 6:30 | This is called uptime. It is the amount of time since the machine was last booted. In this example, the system has been up for six and a half hours. |
|  | 2 users | There are two users logged in. |
|  | load average: | Load average refers to the number of processes that are waiting to run, that is, the number of processes that are in a runnable state and are sharing the CPU. Three values are shown, each for a different period of time. The first is the average for the last 60 seconds, the next the previous 5 minutes, and finally the previous 15 minutes. Values under 1.0 indicate that the machine is not busy. |
| 2 | Tasks: | This summarizes the number of processes and their various process states. |
| 3 | Cpu(s): | This row describes the character of the activities that the CPU is performing. |
|  | 0.7%us | 0.7% of the CPU is being used for user processes. This means processes outside of the kernel itself. |
|  | 1.0%sy | 1.0% of the CPU is being used for system (kernel) processes. |
|  | 0.0%ni | 0.0% of the CPU is being used by “nice” (low priority) processes. |
|  | 98.3%id | 98.3% of the CPU is idle. |
|  | 0.0%wa | 0.0% of the CPU is waiting for I/O. |
| 4 | Mem: | Shows how physical RAM is being used. |
| 5 | Swap: | Shows how swap space (virtual memory) is being used. |

| *表11-3: top 命令信息字段* | | |
| --- | --- | --- |
| 行号 | 字段 | 意义 |
| 1 | top | 程序名。 |
|  | 14:59:20 | 当前时间。 |
|  | up 6:30 | 这是正常运行时间。它是计算机从上次启动到现在所运行的时间。 在这个例子里，系统已经运行了六个半小时。 |
|  | 2 users | 有两个用户登录系统。 |
|  | load average: | 加载平均值是指，等待运行的进程数目，也就是说，处于运行状态的进程个数， 这些进程共享 CPU。展示了三个数值，每个数值对应不同的时间周期。第一个是最后60秒的平均值， 下一个是前5分钟的平均值，最后一个是前15分钟的平均值。若平均值低于1.0，则指示计算机 工作不忙碌。 |
| 2 | Tasks: | 总结了进程数目和各种进程状态。 |
| 3 | Cpu(s): | 这一行描述了 CPU 正在执行的进程的特性。 |
|  | 0.7%us | 0.7% of the CPU is being used for user processes. 这意味着进程在内核之外。 |
|  | 1.0%sy | 1.0%的 CPU 时间被用于系统（内核）进程。 |
|  | 0.0%ni | 0.0%的 CPU 时间被用于"nice"（低优先级）进程。 |
|  | 98.3%id | 98.3%的 CPU 时间是空闲的。 |
|  | 0.0%wa | 0.0%的 CPU 时间来等待 I/O。 |
| 4 | Mem: | 展示物理内存的使用情况。 |
| 5 | Swap: | 展示交换分区（虚拟内存）的使用情况。 |

The top program accepts a number of keyboard commands. The two most interesting are h, which displays the program’s help screen, and q, which quits top.

top 程序接受一系列从键盘输入的命令。两个最有趣的命令是 h 和 q。h，显示程序的帮助屏幕，q， 退出 top 程序。

Both major desktop environments provide graphical applications that display information similar to top (in much the same way that Task Manager in Windows works), but I find that top is better than the graphical versions because it is faster and it consumes far fewer system resources. After all, our system monitor program shouldn’t be the source of the system slowdown that we are trying to track.

两个主要的桌面环境都提供了图形化应用程序，来显示与 top 程序相似的信息 （和 Windows 中的任务管理器差别不多），但是我觉得 top 程序要好于图形化的版本， 因为它运行速度快，并且消费很少的系统资源。毕竟，我们的系统监测程序不能成为 系统怠工的源泉，而这是我们试图追踪的信息。

**控制进程**

Now that we can see and monitor processes, let’s gain some control over them. For our experiments, we’re going to use a little program called xlogo as our guinea pig. The xlogo program is a sample program supplied with the X Window System (the underlying engine that makes the graphics on our display go) which simply displays a re- sizable window containing the X logo. First, we’ll get to know our test subject:

现在我们可以看到和监测进程，然后得到一些对它们的控制权。为了我们的实验，我们将使用 一个叫做 xlogo 的小程序，作为我们的实验品。这个 xlogo 程序是 X 窗口系统 （底层引擎使图形界面显示在屏幕上）提供的实例程序，这个实例简单地显示一个大小可调的 包含 X 标志的窗口。首先，我们需要知道测试的主题：

[me@linuxbox ~]$ xlogo

After entering the command, a small window containing the logo should appear somewhere on the screen. On some systems, xlogo may print a warning message, but it may be safely ignored.

命令执行之后，一个包含 X 标志的小窗口应该出现在屏幕的某个位置上。在一些系统中，xlogo 命令 会打印一条警告信息，但是不用理会它。

Tip: If your system does not include the xlogo program, try using gedit or kwrite instead.

小贴士：如果你的系统不包含 xlogo 程序，试着用 gedit 或者 kwrite 来代替。

We can verify that xlogo is running by resizing its window. If the logo is redrawn in the new size, the program is running.

通过调整它的窗口大小，我们能够证明 xlogo 程序正在运行。如果这个标志以新的尺寸被重画， 则这个程序正在运行。

Notice how our shell prompt has not returned? This is because the shell is waiting for the program to finish, just like all the other programs we have used so far. If we close the xlogo window, the prompt returns.

注意，为什么我们的 shell 提示符还没有返回？这是因为 shell 正在等待这个程序结束， 就像到目前为止我们用过的其它所有程序一样。如果我们关闭 xlogo 窗口，shell 提示符就返回了。

**中断一个进程**

Let’s observe what happens when we run xlogo again. First, enter the xlogo command and verify that the program is running. Next, return to the terminal window and type Ctrl-c.

我们再运行 xlogo 程序一次，观察一下发生了什么事。首先，执行 xlogo 命令，并且 证实这个程序正在运行。下一步，回到终端窗口，按下 Ctrl-c。

[me@linuxbox ~]$ xlogo

[me@linuxbox ~]$

In a terminal, typing Ctrl-c, interrupts a program. This means that we politely asked the program to terminate. After typing Ctrl-c, the xlogo window closed and the shell prompt returned.

在一个终端中，输入 Ctrl-c，中断一个程序。这意味着，我们礼貌地要求终止这个程序。 输入 Ctrl-c 之后，xlogo 窗口关闭，shell 提示符返回。

Many (but not all) command line programs can be interrupted by using this technique.

通过这个技巧，许多（但不是全部）命令行程序可以被中断。

**把一个进程放置到后台(执行)**

Let’s say we wanted to get the shell prompt back without terminating the xlogo program. We’ll do this by placing the program in the background. Think of the terminal as having a foreground (with stuff visible on the surface like the shell prompt) and a background (with hidden stuff behind the surface.) To launch a program so that it is immediately placed in the background, we follow the command with an- “&” character:

比方说，我们想让 shell 提示符返回，却没有终止 xlogo 程序。为达到这个目的，我们把 这个程序放到后台执行。把终端看作是一个有前台（表层放置可见的事物，像 shell 提示符） 和后台（表层之下放置隐藏的事物）（的设备）。启动一个程序，让它立即在后台 运行，我们在程序命令之后，加上”&”字符：

[me@linuxbox ~]$ xlogo &

[1] 28236

[me@linuxbox ~]$

After entering the command, the xlogo window appeared and the shell prompt returned, but some funny numbers were printed too. This message is part of a shell feature called job control. With this message, the shell is telling us that we have started job number 1 (“[1]”) and that it has PID 28236. If we run ps, we can see our process:

执行命令之后，这个 xlogo 窗口出现，并且 shell 提示符返回，同时打印一些有趣的数字。 这条信息是 shell 特性的一部分，叫做工作控制。通过这条信息，shell 告诉我们，已经启动了 工作号为1（“［1］”），PID 为28236的程序。如果我们运行 ps 命令，可以看到我们的进程：

[me@linuxbox ~]$ ps

PID TTY TIME CMD

10603 pts/1 00:00:00 bash

28236 pts/1 00:00:00 xlogo

28239 pts/1 00:00:00 ps

The shell’s job control facility also gives us a way to list the jobs that are have been launched from our terminal. Using the jobs command, we can see this list:

工作控制，这个 shell 功能可以列出从终端中启动的任务。执行 jobs 命令，我们可以看到这个输出列表：

[me@linuxbox ~]$ jobs

[1]+ Running xlogo &

The results show that we have one job, numbered “1”, that it is running, and that the command was xlogo &.

结果显示我们有一个任务，编号为“1”，它正在运行，并且这个任务的命令是 xlogo ＆。

**进程返回到前台**

A process in the background is immune from keyboard input, including any attempt interrupt it with a Ctrl-c. To return a process to the foreground, use the fg command, this way:

一个在后台运行的进程对一切来自键盘的输入都免疫，也不能用 Ctrl-c 来中断它。使用 fg 命令，让一个进程返回前台执行：

[me@linuxbox ~]$ jobs

[1]+ Running xlogo &

[me@linuxbox ~]$ fg %1

xlogo

The command fg followed by a percent sign and the job number (called a jobspec) does the trick. If we only have one background job, the jobspec is optional. To terminate xlogo, type Ctrl-c.

fg 命令之后，跟随着一个百分号和工作序号（叫做 jobspec）。如果我们只有一个后台任务，那么 jobspec 是可有可无的。输入 Ctrl-c 来终止 xlogo 程序。

**停止一个进程**

Sometimes we’ll want to stop a process without terminating it. This is often done to allow a foreground process to be moved to the background. To stop a foreground process, type Ctrl-z. Let’s try it. At the command prompt, type xlogo, the Enter key, then Ctrl-z:

有时候，我们想要停止一个进程，而没有终止它。这样会把一个前台进程移到后台等待。 输入 Ctrl-z，可以停止一个前台进程。让我们试一下。在命令提示符下，执行 xlogo 命令， 然后输入 Ctrl-z:

[me@linuxbox ~]$ xlogo

[1]+ Stopped xlogo

[me@linuxbox ~]$

After stopping xlogo, we can verify that the program has stopped by attempting to resize the xlogo window. We will see that it appears quite dead. We can either restore the program to the foreground, using the fg command, or move the program to the background with the bg command:

停止 xlogo 程序之后，通过调整 xlogo 的窗口大小，我们可以证实这个程序已经停止了。 它看起来像死掉了一样。使用 fg 命令，可以恢复程序到前台运行，或者用 bg 命令把程序移到后台。

[me@linuxbox ~]$ bg %1

[1]+ xlogo &

[me@linuxbox ~]$

As with the fg command, the jobspec is optional if there is only one job.

和 fg 命令一样，如果只有一个任务的话，jobspec 参数是可选的。

Moving a process from the foreground to the background is handy if we launch a graphical program from the command, but forget to place it in the background by appending the trailing “&”.

因为把一个进程从前台移到后台很方便，如果我们从命令行启动一个图形界面的程序，但是 忘记把它放到后台执行，即没有在命令后加上字符”＆”，（也不用担心）。

Why would you want to launch a graphical program from the command line? There are two reasons. First, the program you wish to run might not be listed on the window manager’s menus (such as xlogo). Secondly, by launching a program from the command line, you might be able to see error messages that would otherwise be invisible if the program were launched graphically. Sometimes, a program will fail to start up when launched from the graphical menu. By launching it from the command line instead, we may see an error message that will reveal the problem. Also, some graphical programs have many interesting and useful command line options.

为什么要从命令行启动一个图形界面程序呢？有两个原因。第一个，你想要启动的程序，可能 没有在窗口管理器的菜单中列出来（比方说 xlogo）。第二个，从命令行启动一个程序， 你能够看到一些错误信息，如果从窗口系统中运行程序的话，这些信息是不可见的。有时候， 一个程序不能从图形界面菜单中启动。这时候，应该从命令行中启动它。我们可能会看到 错误信息，这些信息揭示了问题所在。一些图形界面程序还有许多有意思并且有用的命令行选项。

**Signals**

The kill command is used to “kill” programs. This allows us to terminate programs that need killing. Here’s an example:

kill 命令被用来“杀死”程序。这样我们就可以终止需要杀死的程序。这里有一个实例：

[me@linuxbox ~]$ xlogo &

[1] 28401

[me@linuxbox ~]$ kill 28401

[1]+ Terminated xlogo

We first launch xlogo in the background. The shell prints the jobspec and the PID of the background process. Next, we use the kill command and specify the PID of the process we want to terminate. We could have also specified the process using a jobspec (for example, “%1”) instead of a PID.

首先，我们在后台启动 xlogo 程序。shell 打印出 jobspec 和这个后台进程的 PID。下一步，我们使用 kill 命令，并且指定我们想要终止的进程 PID。也可以用 jobspec（例如，“％1”）来代替 PID。

While this is all very straightforward, there is more to it than that. The kill command doesn’t exactly “kill” programs, rather it sends them signals. Signals are one of several ways that the operating system communicates with programs. We have already seen signals in action with the use of Ctrl-c and Ctrl-z. When the terminal receives one of these keystrokes, it sends a signal to the program in the foreground. In the case of Ctrl-c, a signal called INT (Interrupt) is sent; with Ctrl-z, a signal called TSTP (Terminal Stop.) Programs, in turn, “listen” for signals and may act upon them as they are received. The fact that a program can listen and act upon signals allows a program to do things like save work in progress when it is sent a termination signal.

虽然这个命令很直接了当，但不仅仅这些。这个 kill 命令不是确切地“杀死”程序，而是给程序 发送信号。信号是操作系统与程序之间进行通信，所采用的几种方式中的一种。我们已经看到 信号，在使用 Ctrl-c 和 Ctrl-z 的过程中。当终端接受了其中一个按键组合后，它会给在前端运行 的程序发送一个信号。在使用 Ctrl-c 的情况下，会发送一个叫做 INT（中断）的信号；当使用 Ctrl-z 时，则发送一个叫做 TSTP（终端停止）的信号。程序，反过来，倾听信号的到来，当程序 接到信号之后，则做出响应。一个程序能够倾听和响应信号，这个事实允许一个程序做些事情， 比如，当程序接到一个终止信号时，它可以保存所做的工作。

**通过 kill 命令给进程发送信号**

The kill command is used to send signals to programs. Its most common syntax looks like this:

kill 命令被用来给程序发送信号。它最常见的语法形式看起来像这样：

kill [-signal] PID...

If no signal is specified on the command line, then the TERM (Terminate) signal is sent by default. The kill command is most often used to send the following signals:

如果在命令行中没有指定信号，那么默认情况下，发送 TERM（终止）信号。kill 命令被经常 用来发送以下命令：

|  |  |  |
| --- | --- | --- |
| *Table 11-4: Common Signals* | | |
| Number | Name | Meaning |
| 1 | HUP | Hangup. This is a vestige of the good old days when terminals were attached to remote computers with phone lines and modems. The signal is used to indicate to programs that the controlling terminal has “hung up.” The effect of this signal can be demonstrated by closing a terminal session. The foreground program running on the terminal will be sent the signal and will terminate.  This signal is also used by many daemon programs to cause a reinitialization. This means that when a daemon is sent this signal, it will restart and re-read its configuration file. The Apache web server is an example of a daemon that uses the HUP signal in this way. |
| 2 | INT | Interrupt. Performs the same function as the Ctrl-c key sent from the terminal. It will usually terminate a program. |
| 9 | KILL | Kill. This signal is special. Whereas programs may choose to handle signals sent to them in different ways, including ignoring them all together, the KILL signal is never actually sent to the target program. Rather, the kernel immediately terminates the process. When a process is terminated in this manner, it is given no opportunity to “clean up” after itself or save its work. For this reason, the KILL signal should only be used as a last resort when other termination signals fail. |
| 15 | TERM | Terminate. This is the default signal sent by the kill command. If a program is still “alive” enough to receive signals, it will terminate. |
| 18 | CONT | Continue. This will restore a process after a STOP signal. |
| 19 | STOP | Stop. This signal causes a process to pause without terminating. Like the KILL signal, it is not sent to the target process, and thus it cannot be ignored. |

|  |  |  |
| --- | --- | --- |
| *表 11-4: 常用信号* | | |
| 编号 | 名字 | 含义 |
| 1 | HUP | 挂起。这是美好往昔的痕迹，那时候终端机通过电话线和调制解调器连接到 远端的计算机。这个信号被用来告诉程序，控制的终端机已经“挂起”。 通过关闭一个终端会话，可以说明这个信号的作用。发送这个信号到终端机上的前台程序，程序会终止。  许多守护进程也使用这个信号，来重新初始化。这意味着，当发送这个信号到一个守护进程后， 这个进程会重新启动，并且重新读取它的配置文件。Apache 网络服务器守护进程就是一个例子。 |
| 2 | INT | 中断。实现和 Ctrl-c 一样的功能，由终端发送。通常，它会终止一个程序。 |
| 9 | KILL | 杀死。这个信号很特别。鉴于进程可能会选择不同的方式，来处理发送给它的 信号，其中也包含忽略信号，这样呢，从不发送 Kill 信号到目标进程。而是内核立即终止 这个进程。当一个进程以这种方式终止的时候，它没有机会去做些“清理”工作，或者是保存劳动成果。 因为这个原因，把 KILL 信号看作杀手锏，当其它终止信号失败后，再使用它。 |
| 15 | TERM | 终止。这是 kill 命令发送的默认信号。如果程序仍然“活着”，可以接受信号，那么 这个信号终止。 |
| 18 | CONT | 继续。在停止一段时间后，进程恢复运行。 |
| 19 | STOP | 停止。这个信号导致进程停止运行，而没有终止。像 KILL 信号，它不被 发送到目标进程，因此它不能被忽略。 |

Let’s try out the kill command:

让我们实验一下 kill 命令：

[me@linuxbox ~]$ xlogo &

[1] 13546

[me@linuxbox ~]$ kill -1 13546

[1]+ Hangup xlogo

In this example, we start the xlogo program in the background and then send it a HUP signal with kill. The xlogo program terminates and the shell indicates that the background process has received a hangup signal. You may need to press the enter key a couple of times before you see the message. Note that signals may be specified either by number or by name, including the name prefixed with the letters “SIG”:

在这个例子里，我们在后台启动 xlogo 程序，然后通过 kill 命令，发送给它一个 HUP 信号。 这个 xlogo 程序终止运行，并且 shell 指示这个后台进程已经接受了一个挂起信号。在看到这条 信息之前，你可能需要多按几次 enter 键。注意，既可以用号码，也可以用名字，不过要在名字前面 加上字母“SIG”，来指定所要发送的信号。

[me@linuxbox ~]$ xlogo &

[1] 13546

[me@linuxbox ~]$ kill -1 13546

[1]+ Hangup xlogo

Repeat the example above and try out the other signals. Remember, you can also use jobspecs in place of PIDs.

重复上面的例子，试着使用其它的信号。记住，你也可以用 jobspecs 来代替 PID。

Processes, like files, have owners, and you must be the owner of a process (or the superuser) in order to send it signals with kill.

进程，和文件一样，拥有所有者，所以为了能够通过 kill 命令来给进程发送信号， 你必须是进程的所有者（或者是超级用户）。

In addition to the list of signals above, which are most often used with kill, there are other signals frequently used by the system. Here is a list of other common signals:

除了上表列出的 kill 命令最常使用的信号之外，还有一些系统频繁使用的信号。以下是其它一些常用 信号列表：

|  |  |  |
| --- | --- | --- |
| *Table 11-5: Other Common Signals* | | |
| Number | Name | Meaning |
| 3 | QUIT | Quit |
| 11 | SEGV | Segmentation Violation. This signal is sent if a program makes illegal use of memory, that is, it tried to write somewhere it was not allowed to. |
| 20 | TSTP | Terminal Stop. This is the signal sent by the terminal when the Ctrl-z key is pressed. Unlike the STOP signal, the TSTP signal is received by the process and may be ignored. |
| 28 | WINCH | Window Change. This is a signal sent by the system when a window changes size. Some programs , like top and less will respond to this signal by redrawing themselves to fit the new window dimensions. |

|  |  |  |
| --- | --- | --- |
| *表 11-5: 其它常用信号* | | |
| 编号 | 名字 | 含义 |
| 3 | QUIT | 退出 |
| 11 | SEGV | 段错误。如果一个程序非法使用内存，就会发送这个信号。也就是说， 程序试图写入内存，而这个内存空间是不允许此程序写入的。 |
| 20 | TSTP | 终端停止。当按下 Ctrl-z 组合键后，终端发送这个信号。不像 STOP 信号， TSTP 信号由目标进程接收，且可能被忽略。 |
| 28 | WINCH | 改变窗口大小。当改变窗口大小时，系统会发送这个信号。 一些程序，像 top 和 less 程序会响应这个信号，按照新窗口的尺寸，刷新显示的内容。 |

For the curious, a complete list of signals can be seen with the following command:

为了满足读者的好奇心，通过下面的命令可以得到一个完整的信号列表：

[me@linuxbox ~]$ kill -l

**通过 killall 命令给多个进程发送信号**

It’s also possible to send signals to multiple processes matching a specified program or user name by using the killall command. Here is the syntax:

也有可能通过 killall 命令，给匹配特定程序或用户名的多个进程发送信号。下面是 killall 命令的语法形式：

killall [-u user] [-signal] name...

To demonstrate, we will start a couple of instances of the xlogo program and then terminate them:

为了说明情况，我们将启动一对 xlogo 程序的实例，然后再终止它们：

[me@linuxbox ~]$ xlogo &

[1] 18801

[me@linuxbox ~]$ xlogo &

[2] 18802

[me@linuxbox ~]$ killall xlogo

[1]- Terminated xlogo

[2]+ Terminated xlogo

Remember, as with kill, you must have superuser privileges to send signals to processes that do not belong to you.

记住，和 kill 命令一样，你必须拥有超级用户权限才能给不属于你的进程发送信号。

**更多和进程相关的命令**

Since monitoring processes is an important system administration task, there are a lot of commands for it. Here are some to play with:

因为监测进程是一个很重要的系统管理任务，所以有许多命令与它相关。玩玩下面几个命令：

|  |  |
| --- | --- |
| *Table 11-6: Other Process Related Commands* | |
| Command | Description |
| pstree | Outputs a process list arranged in a tree-like pattern showing the parent/child relationships between processes. |
| vmstat | Outputs a snapshot of system resource usage including, memory, swap and disk I/O. To see a continuous display, follow the command with a time delay (in seconds) for updates. For example: vmstat 5. Terminate the output with Ctrl-c. |
| xload | A graphical program that draws a graph showing system load over time |
| tload | Similar to the xload program, but draws the graph in the terminal. Terminate the output with Ctrl-c. |

|  |  |
| --- | --- |
| *表11-6: 其它与进程相关的命令* | |
| 命令名 | 命令描述 |
| pstree | 输出一个树型结构的进程列表，这个列表展示了进程间父/子关系。 |
| vmstat | 输出一个系统资源使用快照，包括内存，交换分区和磁盘 I/O。 为了看到连续的显示结果，则在命令名后加上延时的时间（以秒为单位）。例如，“vmstat 5”。 终止输出，按下 Ctrl-c 组合键。 |
| xload | 一个图形界面程序，可以画出系统负载的图形。 |
| tload | 与 xload 程序相似，但是在终端中画出图形。使用 Ctrl-c，来终止输出。 |