会终止，通常由下列条件引起：

1) 正常退出（自愿的）。

2) 出错退出（自愿的）。

3) 严重错误（非自愿）。

4) 被其他进程杀死（非自愿）。

多数进程是由于完成了它们的工作而终止。当编译器完成了所给定程序的编译之后，编译器执行一个系统调用，通知操作系统它的工作已经完成。在UNIX中该调用是exit，而在Windows中，相关的调用是ExitProcess。面向屏幕的程序也支持自愿终止。字处理软件、Internet浏览器和类似的程序中总有一个供用户点击的图标或菜单项，用来通知进程删除它所打开的任何临时文件，然后终止。

进程终止的第二个原因是进程发现了严重错误。例如，如果用户键入命令

```
cc foo.c
```

要编译程序foo.c，但是该文件并不存在，于是编译器就会退出。在给出了错误参数时，面向屏幕的交互式进程通常并不退出。相反，这些程序会弹出一个对话框，并要求用户再试一次。

进程终止的第三个原因是由进程引起的错误，通常是由于程序中的错误所致。例如，执行了一条非法指令、引用不存在的内存，或除数是零等。有些系统中（如UNIX），进程可以通知操作系统，它希望自行处理某些类型的错误，在这类错误中，进程会收到信号（被中断），而不是在这类错误出现时终止。

第四种终止进程的原因是，某个进程执行一个系统调用通知操作系统杀死某个其他进程。在UNIX中，这个系统调用是kill。在Win32中对应的函数是TerminateProcess。在这两种情形中，“杀手”都必须获得确定的授权以便进行动作。在有些系统中，当一个进程终止时，不论是自愿的还是其他原因，由该进程所创建的所有进程也一律立即被杀死。不过，UNIX和Windows都不是这种工作方式。

2.1.4 进程的层次结构

某些系统中，当进程创建了另一个进程后，父进程和子进程就以某种形式继续保持关联。子进程自身可以创建更多的进程，组成一个进程的层次结构。请注意，这与植物和动物的有性繁殖不同，进程只有一个父进程（但是可以有零个、一个、两个或多个子进程）。

在UNIX中，进程和它的所有子女以及后裔共同组成一个进程组。当用户从键盘发出一个信号时，该信号被送给当前与键盘相关的进程组中的所有成员（它们通常是在当前窗口创建的所有活动进程）。每个进程可以分别捕获该信号、忽略该信号或采取默认的动作，即被该信号杀死。

这里有另一个例子，可以用来说明进程层次的作用，考虑UNIX在启动时如何初始化自己。一个称为init的特殊进程出现在启动映像中。当它开始运行时，读入一个说明终端数量的文件。接着，为每个终端创建一个新进程。这些进程等待用户登录。如果有一个用户登录成功，该登录进程就执行一个shell准备接收命令。所接收的这些命令会启动更多的进程，以此类推。这样，在整个系统中，所有的进程都属于以 init 为根的一棵树。

相反，Windows中没有进程层次的概念，所有的进程都是地位相同的。惟一类似于进程层次的暗示是在创建进程的时候，父进程得到一个特别的令牌（称为句柄），该句柄可以用来控制子进程。但是，它有权把这个令牌传送给某个其他进程，这样就不存在进程层次了。在UNIX中，进程就不能剥夺其子女的“继承权”。

2.1.5 进程的状态

尽管每个进程是一个独立的实体，有其自己的程序计数器和内部状态，但进程之间经常需要相互作用。一个进程的输出结果可能作为另一个进程的输入。在shell命令

```
cat chapter1 chapter2 chapter3 | grep tree
```

中，第一个进程运行cat，将三个文件连接并输出。第二个进程运行grep，它从输入中选择所有包含单词“tree”的那些行。根据这两个进程的相对速度（这取决于这两个程序的相对复杂度和各自所分配到的CPU时间），可能发生这种情况：grep准备就绪可以运行，但输入还没有完成。于是必须阻塞grep，直到输入到来。