```
1        pid = Fork();
```

我们将在本书剩余的部分中都使用错误处理包装函数。它们能够保持代码示例简洁，而又不会给你错误的假象，认为允许忽略错误检查。注意，当在本书中谈到系统级函数时，我们总是用它们的小写字母的基本名字来引用它们，而不是用它们大写的包装函数名来引用。

关于 Unix 错误处理以及本书中使用的错误处理包装函数的讨论，请参见附录 A。包装函数定义在一个叫做 csapp.c 的文件中，它们的原型定义在一个叫做 csapp.h 的头文件中；可以从 CS:APP 网站上在线地得到这些代码。

8.4　进程控制

Unix 提供了大量从 C 程序中操作进程的系统调用。这一节将描述这些重要的函数，并举例说明如何使用它们。

8.4.1　获取进程 ID

每个进程都有一个唯一的正数(非零)进程 ID(PID)。getpid 函数返回调用进程的 PID。getppid 函数返回它的父进程的 PID(创建调用进程的进程)。

```
#include <sys/types.h>
#include <unistd.h>

pid_t getpid(void);
pid_t getppid(void);
```

返回：调用者或其父进程的 PID。

getpid 和 getppid 函数返回一个类型为 pid_t 的整数值，在 Linux 系统上它在 types.h 中被定义为 int。

8.4.2　创建和终止进程

从程序员的角度，我们可以认为进程总是处于下面三种状态之一：

- 运行。进程要么在 CPU 上执行，要么在等待被执行且最终会被内核调度。
- 停止。进程的执行被挂起(suspended)，且不会被调度。当收到 SIGSTOP、SIGTSTP、SIGTTIN 或者 SIGTTOU 信号时，进程就停止，并且保持停止直到它收到一个 SIGCONT 信号，在这个时刻，进程再次开始运行。(信号是一种软件中断的形式，将在 8.5 节中详细描述。)
- 终止。进程永远地停止了。进程会因为三种原因终止：1)收到一个信号，该信号的默认行为是终止进程，2)从主程序返回，3)调用 exit 函数。

```
#include <stdlib.h>

void exit(int status);
```

该函数不返回。

exit 函数以 status 退出状态来终止进程(另一种设置退出状态的方法是从主程序中返回一个整数值)。

父进程通过调用 fork 函数创建一个新的运行的子进程。