系统会有自己添加的额外信号，但是使用了这些信号的程序一般情况下将没有办法移植到Linux的其他版本或者UNIX系统上。

10.3.2 Linux中进程管理相关的系统调用

现在来关注一下Linux系统中与进程管理相关的系统调用。主要的系统调用如图10-6所示。为了开始我们的讨论，fork函数是一个很好的切入点。fork系统调用是Linux系统中创建一个新进程的主要方式，同时也被其他传统的UNIX系统所支持（在下一部分将讨论另一种创建进程的方法）。fork函数创建一个与原始进程完全相同的进程副本，包括相同的文件描述符、相同的寄存器内容和其他的所有东西。fork函数调用之后，原始进程和它的副本（即父进程和子进程）各循其路。虽然在fork函数刚刚结束调用的时候，父、子进程所拥有的全部变量都具有相同的变量值，但是由于父进程的全部地址空间已经被子进程完全复制，父、子进程中的任何一个对内存的后续操作所引起的变化将不会影响另外一个进程。fork函数的返回值，对于子进程来说，恒为0；对于父进程来说，是它所生成的子进程的PID。使用返回的PID，可以区分哪一个进程是父进程，哪一个进程是子进程。

在大多数情况下，调用fork函数之后，子进程需要执行不同于父进程的代码。以shell为例。它从终端读取一行命令，调用fork函数生成一个子进程，然后等待子进程来执行这个命令，子进程结束之后继续读取下一条命令。在等待子进程结束的过程中，父进程调用系统调用waitpid，一直等待直到子进程结束运行（如果该父进程不止拥有一个子进程，那么要一直等待直到所有的子进程全部结束运行）。waitpid系统调用有三个参数。设置第一个参数可以使调用者等待某一个特定的子进程。如果第一个参数为-1，任何一个子进程结束系统调用waitpid即可返回（比如说，第一个子进程）。第二个参数是一个用来存储子进程退出状态（正常退出、异常退出和退出值）的变量地址。第三个参数决定了如果没有子进程结束运行的话，调用者是阻塞还是返回。

仍然以shell为例，子进程必须执行用户键入的命令。子进程通过调用系统调用exec来执行用户命令，以exec函数的第一个参数命名的文件将会替换掉子进程原来的全部核心映像。图10-7展示了一个高度简化的shell（有助于理解系统调用fork，waitpid和exec的用法）。

系统调用	描述
pid=fork ()	创建一个与父进程一样的子进程
pid=waitpid (pid,&statloc,opts)	等待子进程终止
s=execve (name,argv,envp)	替换进程的核心映像
exit (status)	终止进程运行并返回状态值
s=sigaction (sig,&act,&oldact)	定义信号处理的动作
s=sigreturn (&context)	从信号返回
s=sigprocmask (how,&set,&old)	检查或更换信号掩码
s=sigpending (set)	获得阻塞信号集合
s=sigsuspend (sigmask)	替换信号掩码或挂起进程
s=kill (pid,sig)	发送信号到进程
residual=alarm (seconds)	设置报警时钟
s=pause ()	挂起调用程序直到下一个信号出现

图10-6 一些与进程相关的系统调用。如果发生错误，则返回值s是-1，pid指进程ID，residual指前一个警报的剩余时间。参数的含义由其名字指出

在大多数情况下，exec函数有三个参数：待执行文件的文件名，指向参数数组的指针和指向环境数组的指针。简单介绍一下其他的类似函数。很多库函数，如execl、execv、execle和execve，允许省略参数或者用不同的方式来指定参数。上述的所有库函数都会调用相同的底层系统调用。尽管系统调用是exec函数，但是函数库中却没有同名的库函数，所以只能使用上面提到的其他函数。

考虑在shell中输入如下命令：

```
cp file1 file2
```

用来建立一个名为file2的file1的副本。在shell调用fork函数之后，子进程定位并执行文件名为cp的可执