行文件同时把需要复制的文件信息传递给它。

cp的主程序（还有很多其他的程序）包含一个函数声明：

```
main(argc, argv, envp)
```

在这里，参数argc表示命令行中包括程序名的项的数目。在上面所举的例子中，argc的值为3。

第二个参数argv是一个指向数组的指针。数组的第i项是一个指向命令行中第i个字符串的指针。在此例中，argv[0]指向字符串“cp”。以此类推，argv[1]指向五字节长度的字符串“file1”，argv[2]指向五字节长度的字符串“file2”。

main的第三个参数envp是一个指向环境的指针，这里的环境，是指一个包含若干个形如name = value赋值语句的字符串数组，这个数组将传递终端类型、主目录名等信息给程序。在图10-7中，没有要传给子进程的环境列表，所以在这里，execve函数的第三个参数是0。

```
while (TRUE) {                                  /*永远重复*/
    type_prompt( );                             /*在屏幕上显示提示符*/
    read_command(command, params);              /*从键盘读取输入行*/
                                                /*创建子进程*/
    pid = fork( );
    if (pid < 0) {
        printf("Unable to fork 0");             /*错误状态*/
        continue;                               /*重复循环*/
    }

    if (pid != 0) {
        waitpid (-1, &status, 0);               /*父进程等待子进程*/
    } else {
        execve(command, params, 0);             /*子进程执行操作*/
    }
}
```

图10-7 一个高度简化的shell

如果exec函数看起来太复杂了，不要泄气，这已经是最复杂的系统调用了，剩下的要简单很多。作为一个简单的例子，我们来考虑exit函数，当进程结束运行时会调用这个函数。它有一个参数，即退出状态（从0到255），这个参数的值最后会传递给父进程调用waitpid函数的第二个参数——状态参数。状态参数的低字节部分包含着结束状态，0意味着正常结束，其他的值代表各种不同的错误。状态参数的高字节部分包含着子进程的退出状态（从0到255），其值由子进程调用的exit系统调用指定。例如，如果父进程执行如下语句：

```
n = waitpid(-1, &status, 0);
```

它将一直处于挂起状态，直到有子进程结束运行。如果子进程退出时以4作为exit函数的参数，父进程将会被唤醒，同时将变量*n*设置为子进程的PID，变量*status*设置为0x0400（在C语言中，以0x作为前缀表示十六进制）。变量*status*的低字节与信号有关，高字节是子进程返回时调用exit函数的参数值。

如果一个进程退出但是它的父进程并没有在等待它，这个进程进入*僵死状态*（zombie state）。最后当父进程等待它时，这个进程才会结束。

一些与信号相关的系统调用以各种各样的方式被运用。比方说，如果一个用户偶然间命令文字编辑器显示一篇超长文档的全部内容，然后意识到这是一个误操作，这就需要采用某些方法来打断文字编辑器的工作。对于用户来说，最常用的选择是敲击某些特定的键（如DEL或者CTRL-C等），从而给文字编辑器发送一个信号。文字编辑器捕捉到这个信号，然后停止显示。

为了表明所关心的信号有哪些，进程可以调用系统调用sigaction。这个函数的第一个参数是希望捕捉的信号（如图10-5所示）。第二个参数是一个指向结构的指针，在这个结构中包括一个指向信号处理函数的指针以及一些其他的位和标志。第三个参数也是一个指向结构的指针，这个结构接收系统返回的当前正在进行的信号处理的相关信息，有可能以后这些信息需要恢复。

信号处理函数可以运行任意长的时间。尽管如此，在实践当中，通常情况下信号处理函数都非常短小精悍。当信号处理完毕之后，控制返回到断点处继续执行。