```
argv[0] = "copyfile"
argv[1] = "abc"
argv[2] = "xyz"
```

正是通过这个数组，程序访问其参数。

声明了五个变量。前面两个（in_fd和out_fd）用来保存文件描述符，即打开一个文件时返回一个小整数。后面两个（rd_count和wt_count）分别是由read和write系统调用所返回的字节计数。最后一个（buffer）是用于保存所读出的数据以及提供写入数据的缓冲区。

第一行实际语句检查argc，看它是否是3。如果不是，它以状态码1退出。任何非0的状态码均表示出错。在本程序中，状态码是惟一的出错报告处理。一个程序的产品版通常会打印出错信息。

```
/* 复制文件程序，有基本的错误检查和错误报告 */

#include <sys/types.h>                  /* 包括必要的头文件 */
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[]);       /* ANSI原型 */

#define BUF_SIZE 4096                   /* 使用一个4096字节大小的缓冲区 */
#define OUTPUT_MODE 0700                /* 输出文件的保护位 */

int main(int argc, char *argv[])
{
    int in_fd, out_fd, rd_count, wt_count;
    char buffer[BUF_SIZE];

    if (argc != 3) exit(1);             /* 如果argc不等于3，语法错 */

    /* 打开输入文件并创建输出文件 */
    in_fd = open(argv[1], O_RDONLY);    /* 打开源文件 */
    if (in_fd < 0) exit(2);             /* 如果该文件不能打开，退出 */
    out_fd = creat(argv[2], OUTPUT_MODE);  /* 创建目标文件 */
    if (out_fd < 0) exit(3);            /* 如果该文件不能被创建，退出 */

    /* 循环复制 */
    while (TRUE) {
        rd_count = read(in_fd, buffer, BUF_SIZE); /* 读一块数据 */
        if (rd_count <= 0) break;       /* 如果文件结束或读时出错，退出循环 */
        wt_count = write(out_fd, buffer, rd_count); /* 写数据 */
        if (wt_count <= 0) exit(4);     /* wt_count <=0是一个错误 */
    }

    /* 关闭文件 */
    close(in_fd);
    close(out_fd);
    if (rd_count == 0)                  /* 没有读取错误 */
        exit(0);
    else
        exit(5);                        /* 有读取错发生 */
}
```

图4-5 复制文件的一个简单程序

接着我们试图打开源文件并创建目标文件。如果源文件成功打开，系统会给in_fd赋予一个小的整数，用以标识源文件。后续的调用必须引用这个整数，使系统知道需要的是哪一个文件。类似地，如果目标文件也成功地创建了，out_fd会被赋予一个标识用的值。create的第二个变量是设置保护模式。如果打开或创建文件失败，对应的文件描述符被设为-1，程序带着出错码退出。

接下来是用来复制文件的循环。一开始试图读出4KB数据到buffer中。它通过调用库过程read来完成这项工作，该过程实际激活了read系统调用。第一个参数标识文件，第二个参数指定缓冲区，第三个参数指定读出多少字节。赋予rd_count的字节数是实际所读出的字节数。通常这个数是4096，除非文