操作系统实验 3：文件系统的用户界面

王靖康
515030910059
网络空间安全学院
wangjksjtu_01@sjtu.edu.cn

1 实验题目

1. 编写一个文件复制的 C 语言程序：分别使用文件的系统调用 read(fd, buf, nbytes), write(fd, buf, nbytes) 和文件的库函数 fread(buf, size, nitems, fp), fwrite(buf, size, nitems, fp)，编写一个文件的复制程序。当上述函数中 nbytes, size 和 nitems 都取值为 1 时（即一次读写一个字节），比较这两种程序的执行效率。当 nbytes 取 1024 字节，size 取 1024 字节，且 nitems 取 1 时（即一次读写 1024 字节），再次比较这两种程序的执行效率。

2. 分别使用 fscanf 和 fprintf，fgetc 和 fputc，fgets 和 fputs（仅限于行结构的文本文件），实现上述的文件复制程序

3. 编写一个父子进程之间用无名管道进行数据传送的 C 程序。父进程逐一读出一个文件的内容，并通过管道发送给子进程。子进程从管道中读出信息，再将其写入一个新的文件。程序结束后，对原文件和新文件的内容进行比较。

4. 在两个用户的独立程序之间，使用有名管道，重新编写一个 C 程序，实现题目 3 的功能。

2 实验目的

- 理解有名管道和无名管道的原理和区别;
- 提高用 C 语言编制程序的能力，熟悉标准库函数 API 接口;
- 进一步理解、使用和掌握文件的系统调用、文件的标准子例程，能利用和选择这些基本的文件操作完成复杂的文件处理工作。

3 实验 1,2: 文件复制

本实验共使用了五种方式使用 C 语言实现文件的复制程序。下面将分别介绍 C 语言编程库函数、五个复制文件子函数的具体实现以及功能测试和性能对比。

3.1 编程接口介绍

打开文件 open(): int open(const char * pathname,int flags, mode_t mode); 其中，参数 pathname 指向欲打开的文件路径字符串，flag 为打开文件模式，具体包括：O_RDONLY 以只读方式打开文件, O_WRONLY 以只写方式打开文件, O_RDWR 以可读写方式打开文

件。上述三种模式互斥，不能同时使用。但上述模式可与以下一些标志使用或运算一起使用。O_CREAT 若打开的文件不存在则自动建立该文件, O_APPEND 表示所写入的数据会以附加的方式加入到文件后面等。

关闭文件 close(): int close(int fd); 其中，fd 表示需要关闭文件的文件描述符，调用成功返回 0 错误的返回-1。

读取文件 read(): ssize_t read(int fd,void * buf ,size_t count); 该函数会把参数 fd 所指的文件传送 count 个字节到 buf 指针所指的内存中。若参数 count 为 0，则 read() 不会有作用并返回 0。返回值为实际读取到的字节数，如果返回 0，表示已到达文件尾或是无可读取的数据，文件读写位置会随读取到的字节移动。

读取文件 write(): ssize_t write (int fd,const void * buf,size_t count); 会把参数 buf 所指的内存写入 count 个字节到参数 fd 所指的文件内。文件读写位置也会随之移动。如果顺利 write() 会返回实际写入的字节数。当有错误发生时则返回-1，错误代码存入 errno 中。

打开文件流 fopen(): FILE * fopen(const char * path, const char * mode); 其中，path 字符串包含欲打开的文件路径及文件名，参数 mode 字符串则代表着流形态。mode 有下列几种形态字符串: "r" 或"rb" 表示以只读方式打开文件，该文件必须存在; "w" 或"wb" 表示以写方式打开文件，并把文件长度截短为零。"a" 或"ab" 表示以写方式打开文件，新内容追加在文件尾。文件顺利打开后，指向该流的文件指针就会被返回。如果文件打开失败则返回 NULL，并把错误代码存在 errno 中。

关闭文件流 fclose(): int fclose(FILE * stream); 其中，stream 为文件流指针。若关文件动作成功则返回 0，有错误发生时则返回 EOF，并把错误代码存到 errno。

读取文件流 fread(): size_t fread (void *buffer, size_t size, size_t count, FILE *stream); 其中，buffer 表示用于接收数据的内存地址; size 表示要读写的字节数，单位是字节; count 表示进行读写多少个 size 字节的数据项, 每个元素是 size 字节; stream 表示输入流。返回实际读取的元素个数. 如果返回值与 count 不相同, 则可能文件结尾或发生错误，从 ferror 和 feof 获取错误信息或检测是否到达文件结尾。

读取文件流 fwrite(): size_t fwrite(const void* buffer, size_t size, size_t count, FILE* stream); 参数意义与 fread() 函数相同，返回实际写入的数据块数目。

格式化文件读取 fscanf(): int fscanf(FILE *stream, const char *format, ...); 其中，stream 为指向 FILE 对象的指针，format 为格式化说明符，类似 scanf() 函数。

格式化文件输出 fscanf(): int fprintf(FILE *stream, const char *format, ...); 参数意义与 fscanf() 相同。

字符读入 fgetc(): int fgetc(FILE *stream); 从文件流中读入一个字符, stream 为指向 FILE 对象的指针。

行读入 fgets(): char* fgets(char *s, int size, FILE *stream); 读取少于 size 长度的字符到字符数组 s，直到新的一行开始或是文件结束。

字符输出 fputc(): int fputc(const char *s, FILE *stream) 像文件流写入一个字符，stream 为指向 FILE 对象的指针。成功返回一个非负整数，出错返回 EOF。

字符读入 fputs(): int fputs(const char *s, FILE *stream); 从文件流中读入一个字符, strem 为指向 FILE 对象的指针。成功返回一个非负整数，出错返回 EOF。

3.2 模块设计实现

read/write(文件系统调用)

```
void copy1(char *infile, char *outfile) {
    int in, out, length;
    char buffer[BUFFER_SIZE];

    in = open(infile, O_RDONLY,S_IRUSR);
    out = open(outfile, O_WRONLY|O_CREAT);

    if (in == -1 || out == -1) {
        if (in == -1) {
            printf("[Error]: Can not open file %s\n", infile);
        }
        else {
            printf("[Error]: Can not create file %s\n", outfile);
        }
        exit(1);
    }

    while ((length = read(in, buffer, 1024)) > 0) {
        write(out, buffer, length);
    }

    close(in);
    close(out);
}
```

fread/fwrite(文件流读写)

```
void copy2(char *infile, char *outfile) {
    char buffer[BUFFER_SIZE];

    FILE *in = fopen(infile, "r");
    FILE *out = fopen(outfile, "w");

    if (in == NULL || out == NULL) {
        if (in == NULL) {
            printf("[Error]: Can not open file %s\n", infile);
        }
        else {
            printf("[Error]: Can not create file %s\n", outfile);
        }
        exit(1);
    }
    int length = 0;
    while((length = fread(buffer, sizeof(char), BUFFER_SIZE, in)) > 0) {
        fwrite(buffer, sizeof(char), length, out);
```

```
    }

    fclose(in);
    fclose(out);
}
```

fscanf/fprintf(文件流格式化读写)

```
void copy3(char *infile, char *outfile) {
    FILE *in = fopen(infile, "r");
    FILE *out = fopen(outfile, "w");

    char val;
    while (fscanf(in, "%c", &val) != EOF) {
        fprintf(out, "%c", val);
    }

    fclose(in);
    fclose(out);
}
```

fgetc/fputc(文件流字符读写)

```
void copy4(char *infile, char *outfile) {
    FILE *in = fopen(infile, "r");
    FILE *out = fopen(outfile, "w");

    int c;
    while ((c = fgetc(in)) != EOF) {
        fputc(c, out);
    }

    fclose(in);
    fclose(out);
}
```

fgets/fputs(文件流行读写)

```
void copy5(char *infile, char *outfile) {
    FILE *in = fopen(infile, "r");
    FILE *out = fopen(outfile, "w");

    char buffer[BUFFER_SIZE];
    while (fgets(buffer, BUFFER_SIZE, in)!=NULL) {
        fputs(buffer, out);
    }

    fclose(in); fclose(out);
}
```

3.3 功能测试

分别为以上编写的五个函数编译生成二进制文件，进行功能测试和性能测试。

如图 1 所示（以 copy1 函数为例），测试程序功能。如图所示，copy1 文件成功将大小为 500M 的 test.in 文件复制产生了 test.out 文件。且两个文件大小相同，内容完全相同(diff)。另一方面，使用 time 命令对 copy1 到 copy4 四个程序进行性能测试（由于使用 dd

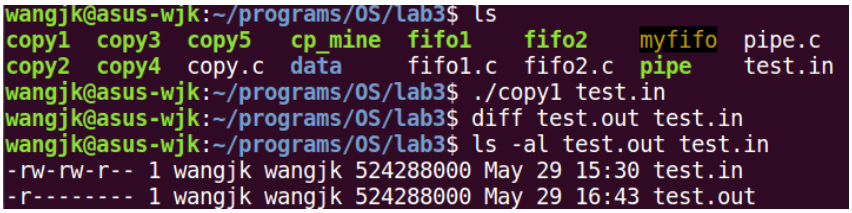

Figure 1: 使用编译好的程序复制大文件

if=/dev/zero of=test.in bs=1024k count=500 生成的文件为二进制文件，并非文本文件，故无法比较单行读入程序），比较几种方法的大文件（500M）读写速度。

如表 1 所示，四种读写方式的性能排序为：文件流读写 > 系统文件读写 > 文件流子符读写 > 文件流格式化读写。原因为 fread 和 fwrite 相比较系统调用 read 和 write 会自动分配缓存，速度比较快。但是，read 和 write 比较节约内存空间。在文件比较大的时候两者的差异比较明显。而由于 fgetc/fputc 是字符级别的读写，故速度明显变慢，fscanf/fprintf 是格式化读写，有复杂的判断逻辑，速度最慢。

Table 1: 复制程序性能对比

Time	read/write	**fread/fwrite**	fscanf/fprintf	fgetc/fputc
Real	1.119s	**0.518s**	31.233s	9.722s
User	0.136s	**0.115s**	29.257s	8.956s
Sys	0.982s	**0.401s**	0.928s	0.713s

4 实验 3: 无名管道

管道是重要的进程间数据交换的方式，具有以下特点：

- 管道是单向的、先进先出的，它把一个进程的输出和另一个进程的输入连接在一起。
- 一个进程（写进程）在管道的尾部写入数据，另一个进程（读进程）从管道的头部读出数据。
- 数据被一个进程读出后，将被从管道中删除，其它读进程将不能再读到这些数据。
- 管道提供了简单的流控制机制，进程试图读空管道时，进程将阻塞。同样，管道已经满时，进程再试图向管道写入数据，进程将阻塞。
- 管道包括无名管道和有名管道两种，前者用于父进程和子进程间的通信，后者可用于运行于同一系统中的任意两个进程间的通信。

4.1 编程接口介绍

无名管道是半双工的，就是对于一个管道来讲，只能读，或者写。另外，无名管道只能在相关的，有共同祖先的进程间使用（即一般用户父子进程）。通过 fork 或者 execve 调用创建的子进程继承了父进程的文件描述符。

打开/关闭管道: int pipe(int filedes[2]); 如果成功建立了管道，则会打开两个文件描述符，并把他们的值保存在一个整数数组中。第一个文件描述符用于读取数据，第二个文件描述符用于写入数据。管道的两个文件描述符相当于管道的两端，一端只负责读数据，一端只负责写数据如果出错返回-1，同时设置 errno。

读写管道: 读写管道与读写普通文件方式一样，调用 write 与 read 函数即可。注意无名管道是半双工的，不能对一个管道的某一端同时进行读写操作。

4.2 模块设计实现

```
1     if (pipe(pipe_fd)<0)     /*创建管道,成功返回0，否则返回-1*/
2         return -1;
3
4     if ((pid=fork())==0) {
5         close(pipe_fd[1]);     /*关闭子进程写描述符*/
6         while((length = read(pipe_fd[0], buffer, BUFFER_SIZE)) > 0) { /*
              子进程读取管道内容*/
7             fwrite(buffer, sizeof(char), length, out);
8         }
9         close(pipe_fd[0]); /*关闭子进程读描述符*/
10        exit(0);
11    }
12    else if (pid>0) {
13        close(pipe_fd[0]); /*关闭父进程读描述符,并分多次向管道中写入文件
              读出的数据*/
14        while((length = fread(buffer, sizeof(char), BUFFER_SIZE, in)) >
              0) {
15            write(pipe_fd[1], buffer, length);
16        }
17        close(pipe_fd[1]); /*关闭父进程写描述符*/
18        exit(0);
19    }
```

4.3 功能测试

代码 4.2 实现了通过 fork 创建子进程，父进程从文件 test.in 中读取数据，分多次写入管道。同时，子进程从管道中多次读取数据，并写入 test.out 文件中。通过进程通信实现了文件复制的功能。

```
wangjk@asus-wjk:~/programs/OS/lab3$ ls
copy1  copy3  copy5   cp_mine  fifo1    fifo2    myfifo  pipe.c
copy2  copy4  copy.c  data     fifo1.c  fifo2.c  pipe    test.in
wangjk@asus-wjk:~/programs/OS/lab3$ ./pipe
wangjk@asus-wjk:~/programs/OS/lab3$ ls -al test.in test.out
-rw-rw-r-- 1 wangjk wangjk 11473 May 27 08:38 test.in
-rw-rw-r-- 1 wangjk wangjk 11473 May 29 17:13 test.out
wangjk@asus-wjk:~/programs/OS/lab3$ diff test.in test.out
wangjk@asus-wjk:~/programs/OS/lab3$ head -n 3 test.in
G9MluBHZuM
4y3uucQnOI
DIFlOzxwIE
```

Figure 2: 无名管道父子进程通信

5 实验 4: 有名管道

不同于无名管道，有名管道创建设备文件，以 FIFO 的形式存在于文件系统中。这样，即使与 FIFO 的创建进程不存在亲缘关系的进程，只要可以访问该路径，就能够彼此通过 FIFO 相互通信。因此，通过 FIFO 不相关的进程也能交换数据。值得注意的是，FIFO 严格遵循先进先出（first in first out），对管道及 FIFO 的读总是从开始处返回数据，对它们的写则把数据添加到末尾。

5.1 编程接口介绍

创建有名管道: int mkfifo(const char * pathname, mode_t mode); 其中，pathname 表示管道文件名，mode 表示管道的读写模式。可以使用 fopen, fclose, fwrite, fread 操作一般文件的操作读写管道文件。值得注意的是，有名管道也是半双工通信，但通过两个有名管道可以实现全双工通信。

5.2 模块设计实现

以只读阻塞方式打开管道

```
1   /* 以只读阻塞方式打开有名管道 */
2   fd = open(FIFO, O_RDONLY);
3   if (fd == -1) {
4   printf("[Error]:␣Cannot␣open␣fifo␣file\n");
5   exit(1);
6   }
7
8   while (1) {
9       bzero(buffer, sizeof(buffer));
10      while ((nread = read(fd, buffer, BUFFER_SIZE)) > 0) {
11          printf("[Info]:␣Reading␣from␣FIFO␣%d\n", nread);
12          fwrite(buffer, sizeof(char), nread, out);
13      }
14      printf("[Info]:␣Writing␣to␣file␣'%s'\n", outfile);
15      break;
16  }
17
18  close(fd);
```

以只写阻塞方式打开管道

```
    /* 以只写阻塞方式打开FIFO管道 */
    fd = open(FIFO, O_WRONLY);
    if (fd == -1) {
        printf("[Info]: Open fifo file error\n");
        exit(1);
    }

    /*向管道中写入字符串*/
    while(1) {
        while ((length = fread(buffer, sizeof(char), BUFFER_SIZE, in)) >
            0) {
            printf("[Info]: Reading from file '%s'\n", infile);
            write(fd, buffer, length);
            printf("[Info]: Writing to FIFO %d\n", length);
        }
        break;
    }
    close(fd);
```

5.3 功能测试

代码 5.2 实现了两个非亲缘进程的有名管道数据同步读写。一个进程从文件 test.in 中分块读入数据（BUFFER_SIZE=1024），并将数据写入管道。另一个进程从 FIFO 管道中分块读入数据，并将数据写入文件 test.out 文件中。从而通过管道实现了文件复制功能（如图 3 所示，先运行 fifo1, 后运行 fifo2）。

```
wangjk@asus-wjk:~/programs/OS/lab3$ ls
copy1  copy3  copy5   cp_mine  fifo1    fifo2    pipe    test.in
copy2  copy4  copy.c  data     fifo1.c  fifo2.c  pipe.c
wangjk@asus-wjk:~/programs/OS/lab3$ sudo ./fifo1
[sudo] password for wangjk:
[Info]: Reading from FIFO 1024
[Info]: Reading from FIFO 1024
[Info]: Reading from FIFO 1024
[Info]: Reading from FIFO 1024
[Info]: Reading from FIFO 1024
[Info]: Reading from FIFO 1024
[Info]: Reading from FIFO 1024
[Info]: Reading from FIFO 1024
[Info]: Reading from FIFO 1024
[Info]: Reading from FIFO 1024
[Info]: Reading from FIFO 1024
[Info]: Reading from FIFO 209
[Info]: Writing to file 'test.out'
wangjk@asus-wjk:~/programs/OS/lab3$ ls test.in test.out -al
-rw-rw-r-- 1 wangjk wangjk 11473 May 27 08:38 test.in
-rw-r--r-- 1 root   root   11473 May 29 17:32 test.out
wangjk@asus-wjk:~/programs/OS/lab3$ diff test.in test.out
```

Figure 3: 有名管道非亲缘进程通信

从图 3 可以得到，文件分多次通过管道读写，实现了进程间的数据交换。在使用过程中，在本地创建临时 myfifo 文件，通过 diff 程序比对赋值后的 test.out 与初始 test.in 文件，二者内容完全相同，说明程序编写正确。注意由于在运行程序是使用了管理员权限，赋值后的文件所有者也为 root。

6 总结与思考

6.1 错误分析

程序调试: 与实验二和四类似，管道编程的过程中也经常会遇到一些难以定位的错误，这主要是由于管道通信中错误的逻辑，容易导致双方均发生阻塞，同时无法通过输出定位错误。因此需要在编程中使用 fflush(stdout) 手动刷新缓冲区，利用使用输出进行程序调试。

fputs/fgets 性能测试: 在本次赋值文件编程性能测试的环节中，由 fputs 和 fgets 编写的 copy5 程序无法对使用 dd 命令生成的二进制文件进行操作，这是由于 fputs/fgets 是对每行进行读写，因此仅适用于文本型数据。对于二进制数据，会导致超出缓冲区从而导致程序崩溃。

6.2 个人总结

本次实验总体上说难度不是很大。但与实验二相同，在实际代码编写过程中，由于存在堵塞情况的发生，代码的调试也花费了较多的时间。经过该实验的学习，使我门对于系统文件 IO 操作有了较为全面的了解（掌握了 read/write/fread/fwrite/fgetc/fputc/fgets/fputs/fscanf/fprintf 等函数的使用），通过性能对比，对多种读写机制的内在机理有了更深刻的理解。同时，通过有名管道和无名管道的通信，使得我对这两种数据交换机制有了更为深刻的理解，提高了代码编写与调试水平。最后，十分感谢刘老师上课的指导和帮助，使得我对 fork 函数和管道机制有了较为深刻的理解。

7 附录

7.1 程序源代码

实验 1,2–文件复制

```c
// copy.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

#define BUFFER_SIZE 1024


void copy1(char *infile, char *outfile) {
    int in, out, length;
    char buffer[BUFFER_SIZE];
```

```
15
16      in = open(infile, O_RDONLY,S_IRUSR);
17      out = open(outfile, O_WRONLY|O_CREAT);
18
19      if (in == -1 || out == -1) {
20          if (in == -1) {
21              printf("[Error]:␣Can␣not␣open␣file␣%s\n", infile);
22          }
23          else {
24              printf("[Error]:␣Can␣not␣create␣file␣%s\n", outfile);
25          }
26          exit(1);
27      }
28
29      while ((length = read(in, buffer, 1024)) > 0) {
30          write(out, buffer, length);
31      }
32
33      close(in);
34      close(out);
35  }
36
37
38  void copy2(char *infile, char *outfile) {
39      char buffer[BUFFER_SIZE];
40
41      FILE *in = fopen(infile, "r");
42      FILE *out = fopen(outfile, "w");
43
44      if (in == NULL || out == NULL) {
45          if (in == NULL) {
46              printf("[Error]:␣Can␣not␣open␣file␣%s\n", infile);
47          }
48          else {
49              printf("[Error]:␣Can␣not␣create␣file␣%s\n", outfile);
50          }
51          exit(1);
52      }
53      // printf("%s", buffer);
54      int length = 0;
55      while((length = fread(buffer, sizeof(char), BUFFER_SIZE, in)) > 0) {
56          fwrite(buffer, sizeof(char), length, out);
57      }
58
59      fclose(in);
60      fclose(out);
61  }
62
```

```
63
64  void copy3(char *infile, char *outfile) {
65      FILE *in = fopen(infile, "r");
66      FILE *out = fopen(outfile, "w");
67
68      char val;
69      while (fscanf(in, "%c", &val) != EOF) {
70          // printf("%c", val);
71          fprintf(out, "%c", val);
72      }
73
74      fclose(in);
75      fclose(out);
76  }
77
78
79  void copy4(char *infile, char *outfile) {
80      FILE *in = fopen(infile, "r");
81      FILE *out = fopen(outfile, "w");
82
83      int c;
84      while ((c = fgetc(in)) != EOF) {
85          // printf("%c", val);
86          fputc(c, out);
87      }
88
89      fclose(in);
90      fclose(out);
91  }
92
93
94  void copy5(char *infile, char *outfile) {
95      FILE *in = fopen(infile, "r");
96      FILE *out = fopen(outfile, "w");
97
98      char buffer[BUFFER_SIZE];
99
100     while (fgets(buffer, BUFFER_SIZE, in)!=NULL) {
101         // printf("%c", val);
102         fputs(buffer, out);
103     }
104
105     fclose(in);
106     fclose(out);
107 }
108
109
110 int main() {
```

```
111     copy5("test.in", "test.out");
112     return 0;
113 }
```

实验 3–无名管道

```
1  // pipe.c
2  #include <unistd.h>
3  #include <sys/types.h>
4  #include <errno.h>
5  #include <stdio.h>
6  #include <stdlib.h>
7  #include <string.h>
8
9  #define BUFFER_SIZE 1024
10
11 int main() {
12     int pipe_fd[2]; /*用于保存两个文件描述符*/
13     pid_t pid;
14     char buffer[BUFFER_SIZE]; /*用于读数据的缓存*/
15     int r_num; /*用于保存读入数据大数量*/
16     char infile[] = "test.in";
17     char outfile[] = "test.out";
18     FILE *in = fopen(infile, "r");
19     FILE *out = fopen(outfile, "w");
20     int length;
21
22     if (pipe(pipe_fd)<0)    /*创建管道,成功返回0, 否则返回-1*/
23         return -1;
24
25     if ((pid=fork())==0) {
26         close(pipe_fd[1]);    /*关闭子进程写描述符*/
27         while((length = read(pipe_fd[0], buffer, BUFFER_SIZE)) > 0) { /*
               子进程读取管道内容*/
28             fwrite(buffer, sizeof(char), length, out);
29         }
30         close(pipe_fd[0]); /*关闭子进程读描述符*/
31         exit(0);
32     }
33     else if (pid>0) {
34         close(pipe_fd[0]);/*/关闭父进程读描述符,并分多次向管道中写入test.
               in文件内容*/
35         while((length = fread(buffer, sizeof(char), BUFFER_SIZE, in)) >
               0) {
36             write(pipe_fd[1], buffer, length);
37         }
38         close(pipe_fd[1]); /*关闭父进程写描述符*/
39         exit(0);
40     }
```

```
41      return 0;
42  }
```

实验 4–有名管道

```
1   // fifo1.c
2   #include <sys/types.h>
3   #include <sys/stat.h>
4   #include <errno.h>
5   #include <fcntl.h>
6   #include <stdio.h>
7   #include <stdlib.h>
8   #include <string.h>
9   #include <unistd.h>
10
11  #define FIFO   "myfifo"    /* 有名管道文件名*/
12  #define BUFFER_SIZE   1024
13
14  int main()
15  {
16      char buffer[BUFFER_SIZE];
17      int  fd;
18      int  nread;
19      char *outfile = "test.out";
20
21      FILE *out = fopen(outfile, "w");
22
23      /* 判断有名管道是否已存在, 若尚未创建, 则以相应的权限创建*/
24      if (access(FIFO, F_OK) == -1) {
25          if ((mkfifo(FIFO, 0666) < 0) && (errno != EEXIST)) {
26              printf("[Error]:␣Cannot␣create␣fifo␣file\n");
27              exit(1);
28          }
29      }
30
31      /* 以只读阻塞方式打开有名管道 */
32      fd = open(FIFO, O_RDONLY);
33      if (fd == -1) {
34      printf("[Error]:␣Cannot␣open␣fifo␣file\n");
35      exit(1);
36      }
37
38      while (1) {
39          bzero(buffer, sizeof(buffer));
40          while ((nread = read(fd, buffer, BUFFER_SIZE)) > 0) {
41              // printf("[Info]: Read '%s' from FIFO\n", buffer);
42              printf("[Info]:␣Reading␣from␣FIFO␣%d\n", nread);
43              fwrite(buffer, sizeof(char), nread, out);
44          }
```

```
45          printf("[Info]: Writing to file '%s'\n", outfile);
46          break;
47      }
48
49      close(fd);
50      exit(0);
51  }
```

```
1   // fifo2.c
2   #include <sys/types.h>
3   #include <sys/stat.h>
4   #include <errno.h>
5   #include <fcntl.h>
6   #include <stdio.h>
7   #include <stdlib.h>
8   #include <unistd.h>
9
10  #define FIFO   "myfifo"    /* 有名管道文件名*/
11  #define BUFFER_SIZE 1024
12  /*常量PIPE_BUF 定义在于limits.h中*/
13
14  int main(int argc, char * argv[]) /*参数为即将写入的字符串*/
15  {
16      int fd;
17      char buffer[BUFFER_SIZE];
18      int length;
19      char infile [] = "test.in";
20      FILE *in = fopen(infile, "r");
21
22      /* 以只写阻塞方式打开FIFO管道 */
23      fd = open(FIFO, O_WRONLY);
24      if (fd == -1) {
25          printf("[Info]: Open fifo file error\n");
26          exit(1);
27      }
28
29      /*向管道中写入字符串*/
30
31      while(1) {
32          while ((length = fread(buffer, sizeof(char), BUFFER_SIZE, in)) >
                0) {
33              printf("[Info]: Reading from file '%s'\n", infile);
34              write(fd, buffer, length);
35              printf("[Info]: Writing to FIFO %d\n", length);
36              // printf("Write '%s' to FIFO\n", buffer);
37          }
38          break;
39      }
```

```
40      close(fd);
41      exit(0);
42
43  }
```