实验三 文件系统的用户界面

学号：515030910067　姓名：　杨超琪　　日期：　2018.06.02

一、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。你还可用其他的方法实现文件的复制功能吗？

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 3 的功能。

二、算法设计思路

1. 调用 read & write、fread & fwrite 函数复制文件

使用 Unix 文件系统的系统调用函数 open 函数将文件转化成标准读入数据流，再使用系统调用 read 将文件数据流读入缓冲 buf，然后使用系统调用 write 将缓冲 buf 读入另一个文件输出流，进过多次的读出与写入，即可将一个文件复制到另一个文件中。

使用 C 语言文件的库函数 fopen 函数将文件转化成标准读入数据流，再使用库函数 fread 将文件数据流读入缓冲 buf，然后使用库函数 fwrite 将缓冲 buf 读入另一个文件输出流，进过多次的读出与写入，即可将一个文件复制到另一个文件中。

在两种方式比较过程中，分别指定调用参数 nbytes, size 和 nitems 分别取（1，1，1）与（1024，1024，1）代表分别是进行单个字符的读出写入与一行字符的读出写入，比较两种方式所用的时间。

其中，使用 time.h 库文件中的 clock()函数查看文件复制开始点与结束点的时间，统计比较两种方法文件复制的效率。

注意，在这里使用 read & write、fread & fwrite 都是直接对二进制文件进行操作，因此可以对任意文件进行。

2. 调用 fscanf & fprintf、fgetc & fputc、fgets & fputs 函数复制行文件

六个函数 fscanf & fprintf、fgetc & fputc、fgets & fputs 分别均是 C 语言文件库中的函数，分别用于格式化文件的读入输出，单个字符进行读入输出，字符串进行读入输出。我们的思路与上一题一样，先打开一个文件的数据流，然后分别使用 fscanf、fgets、fgetc 获得标准输入流，字符串输入流，单个字符输入流，利用 buf 进行存储，利用 fprintf、fputs、fputc 将流文件输出到另一个文件，完成复制。

其中，使用 time.h 库文件中的 clock()函数查看文件复制开始点与结束点的时间，统计比较三种方法文件复制的效率。

注意，在这里使用 fscanf & fprintf、fgetc & fputc、fgets & fputs 都是只能对行结构文件进行操作，若否，则可能输出乱码。

3. 无名管道数据传输

仅有的消息通信和共享内存不能在进程之间传递大量的数据与消息，因此产生了一种些数据只能顺序卸载尾部，度数据只允许从头部顺序地读出的机构：管道。

无名管道 pipe 只能在与创建 pipe 进程的同一进程族内传递数据的打

开文件，其他进程感觉不到该管道的存在。此时操作系统会为父子进程分别分配一个 file 结构，父子进程只需要响应地打开和关闭读写端口就可以进程数据的传递。

在这里，我们使用 fgets & fputs 两个函数，首先是父进程使用 fgets 函数从 input.txt 文件中读入数据流，利用 write()函数传递到 pipe 的写入端口；子进程使用 read()函数从 pipe 的读端口获得数据流，利用 fputs 函数将其写入到 output.txt 文件中，从而实现了文件的复制。

4. 有名管道数据传输

有名管道则不同，其像普通文件一样有目录项，在文件系统中能长久地存在，任何有访问权限的用户都可以通过路径名来打开它，进而进行数据的存取，因此无关的进程就可以通过有名管道进行通信。

首先我们使用 mknod()函数创建有名管道。第一个进程进程使用只写方式打开有名管道，使用 fgets 函数从 input.txt 中读入数据，并用 write()写到管道中；另一个进程使用只读模式打开管道，利用 read()函数从管道中读入数据，使用 fputs 函数将数据存入文件，实现文件的复制。

三、模块设计、功能和接口说明

1. 调用 read & write、fread & fwrite 函数复制文件

1) read(int fd, void *buf, size_t count);

返回值：成功返回读取的字节数，出错返回-1 并设置 errno，如果在调 read 之前已到达文件末尾，则这次 read 返回 0。

参数说明：fd 表示的是读文件的描述字；buf 表示的是读操作的数据的保存首地址指针；count 表示的是想要读的字节的数目。

2) write(int fd, const void *buf, size_t count);

返回值：成功返回写入的字节数，出错返回-1 并设置 errno 写常规文件时，write 的返回值通常等于请求写的字节数 count，而向终端设备或网络写则不一定。

参数说明：fd 表示的是写文件的描述字；buf 表示的是数据首地

址的指针，将会把 buf 保存的地址存入文件中；count 表示的是请求写的数据的字节个数。

3) fread(void * ptr, size_t size, size_t count, FILE * stream);

返回值：函数返回读取数据的个数。

参数说明：ptr：指向保存结果的指针；size：每个数据类型的大小；count：数据的个数；stream：文件指针函数返回读取数据的个数。

4) fwrite(const void * ptr, size_t size, size_t count, FILE * stream);

返回值：函数返回写入数据的个数。

参数说明：ptr：指向保存数据的指针；size：每个数据类型的大小；count：数据的个数；stream：文件指针函数返回写入数据的个数。

2. 调用 fscanf & fprintf、fgetc & fputc、fgets & fputs 函数复制行文件

1) fscanf(FILE*fp, const char*format,[argument])

参数说明：fp 表示文件的描述字；char*format 表示格式化的字符串，可以使用正则表达式，在试验中我们使用了"%[^\n]"，表示匹配直到遇到换行符。

2) fprintf (FILE* stream, const char*format, [argument])

参数说明：stream 表示流文件描述字；char*format 表示格式化的字符串，可用正则表达式。

3) fgetc(FILE *fp)

参数说明：fp 表示流文件的描述字，表示从 fp 文件中读入一个字符。

4) fputc (char c, FILE *fp)

参数说明：fp 表示流文件的描述字，表示将字符 c 写入 fp 文件。

5) fgets(char *buf, int bufsize, FILE *fp)

参数说明：buf 表示将得到的数据存入 buf 为首指针的地址中；bufsize 表示读入的字节个数；fp 表示流文件的描述字。

6) fputs(char *buf，FILE *fp)

参数说明：buf 表示将 buf 为首指针的地址中的数据存入 fp。

3. **无名管道数据传输**

1) pipe(int fd[2]);

声明无名管道，在创建子进程之前调用，在后续操作中，父子进程通过这个无名管道，控制读 fd[0]，写 fd[1]口即可进行通信。

4. **有名管道数据传输**

1) mknod (char *pathname, int mode, int device);

声明有名管道，只需要其中一个进程创建即可，pathname 中是名字，mode 中可以设置管道文件的属性与权限，device 是设备号。

四、重要数据结构和变量说明

1. begin_time & end_time;

begin_time 和 end_time 都是 clock_t 的实例，用于保存当前位置的时间，这两个变量用于记录不同读写复制方式所使用的事时间。

2. buf[BUFE_SIZE]

buf 在每个程序中都使用到了，用于存储中间数据流。

3. fd[2];

fd 表示的是管道的名称，fd[0]为读端，fd[1]为写端。

五、测试方法与分析

1. **调用 read & write、fread & fwrite 函数复制文件**

1) **使用 read & write 并且是一次一个字节模式**

```
linuxvirtualpc1@ubuntu:~/lab3$ sudo gcc -o 1 1.cpp
linuxvirtualpc1@ubuntu:~/lab3$ sudo ./1 test.mp4 test2.mp4
copy test.mp4 success
timespan: 2.180734 seconds.
```

2) 使用 read & write 并且是一次 1024 个字节的模式

```
linuxvirtualpc1@ubuntu:~/lab3$ sudo gcc -o 2 2.cpp
linuxvirtualpc1@ubuntu:~/lab3$ sudo ./2 test.mp4 test2.mp4
copy test.mp4 success
timespan: 0.004511 seconds.
```

3) 使用 fread & fwrite 并且是一次一个字节模式

```
linuxvirtualpc1@ubuntu:~/lab3$ sudo gcc -o 3 3.c
linuxvirtualpc1@ubuntu:~/lab3$ sudo ./3 test.mp4 test2.mp4
copy test.mp4 success
timespan: 0.114848 seconds.
```

4) 使用 fread & fwrite 并且是一次 1024 个字节的模式

```
linuxvirtualpc1@ubuntu:~/lab3$ sudo gcc -o 4 4.c
linuxvirtualpc1@ubuntu:~/lab3$ sudo ./4 test.mp4 test2.mp4
copy test.mp4 success
timespan: 0.003191 seconds.
```

对比分析表：

Time(s)	(nbytes,size,nitems)=(1,1,1)	(nbytes,size,nitems)=(1024,1024,1)
read & write	2.180734	0.004511
fread&fwrite	0.114848	0.003191

从对比分析表中可以看出：（1）使用单个字节读写的效率远远低于使用多个（1024）个字节的读写效率；（2）系统调用 read & write 的效率低于使用库函数 fread & fwrite 的效率。使用系统调用会影响系统的性能。与函数调用相比，系统调用的开销要大些，因为在执行系统调用时，Linux 必须从运行用户代码切换到执行内核代码，然后再返回用户代码。

循环执行系统调用 write()时，每次向文件输出 1 字节，但由于块设备的读写是通过系统缓冲区进行的，故 1024 次写才需要实际使用一次 I/O 操作，但需要 1024 次从用户态转换到核心态和从核心态转换回用户态的开销。

循环执行 fwrite()时，每次希艾娜个用户态空间的流文件缓冲区写入 1 字节，1024 次写操作填满该缓冲区后，才发起一次 write 的系统调用，转换到核心态，并执行一次实际的 I/O 操作。两种方式花费 I/O 时间相同，但后者

仅需要 1 次用户态核心态切换的系统开销。故当不是以整块的方法输入输出数据时，使用流文件操作比使用系统调用效率高很多。

2. 调用 fscanf & fprintf、fgetc & fputc、fgets & fputs 函数复制行文件

在使用 fscanf & fprintf、fgetc & fputc、fgets & fputs 函数复制文件时，输入文件为 input.txt，输出文件分别为 output1.txt，output2.txt，output3.txt。

```
linuxvirtualpc1@ubuntu:~/lab3$ sudo gcc -o 5 5.c
linuxvirtualpc1@ubuntu:~/lab3$ sudo ./5
using fgets & fputs:
##############################
I will run, I will climb, I will soar
I'm undefeated
Jumping out of my skin, pull the chord
Yeah I believe it
The past is everything we were don't make us who we are
so I'll dream until I make it real and all I see is stars
It's not until you fall that you fly.
##############################
timespan: 0.000030 seconds.

using fscanf & fprintf:
##############################
I will run, I will climb, I will soar
I'm undefeated
Jumping out of my skin, pull the chord
Yeah I believe it
The past is everything we were don't make us who we are
so I'll dream until I make it real and all I see is stars
It's not until you fall that you fly.
##############################
timespan: 0.000078 seconds.

using fgetc & fputc:
##############################
I will run, I will climb, I will soar
I'm undefeated
Jumping out of my skin, pull the chord
Yeah I believe it
The past is everything we were don't make us who we are
so I'll dream until I make it real and all I see is stars
It's not until you fall that you fly.
##############################
timespan: 0.000063 seconds.
```

对比分析表:

	fgets & fputs	**fscanf & fprintf**	**fgetc & fputc**
Time(s)	0.000030	0.000078	0.000063

在对比分析表中，我们也对比了使用 fscanf & fprintf、fgetc & fputc、fgets & fputs 函数进行复制时的效率。发现 fgets & fputs 效果比较好。

同时，我们也使用了上一次实验使用的消息队列的方式来进行文件的复制。

```
linuxvirtualpc1@ubuntu:~/lab3$ sudo ./client
[sudo] password for linuxvirtualpc1:
Client has sent the following Message to the Server.
############################
I will run, I will climb, I will soar
I'm undefeated
Jumping out of my skin, pull the chord
Yeah I believe it
The past is everything we were don't make us who we are
so I'll dream until I make it real and all I see is stars
It's not until you fall that you fly.
############################
timespan: 0.000102 seconds.
```

最后，我们发现，fscanf & fprintf、fgetc & fputc、fgets & fputs 以及消息队列的方式，文件复制的过程都是正确的。

```
linuxvirtualpc1@ubuntu:~/lab3$ cat input.txt
I will run, I will climb, I will soar
I'm undefeated
Jumping out of my skin, pull the chord
Yeah I believe it
The past is everything we were don't make us who we are
so I'll dream until I make it real and all I see is stars
It's not until you fall that you fly.
linuxvirtualpc1@ubuntu:~/lab3$ diff input.txt output1.txt
linuxvirtualpc1@ubuntu:~/lab3$ diff input.txt output2.txt
linuxvirtualpc1@ubuntu:~/lab3$ diff input.txt output3.txt
linuxvirtualpc1@ubuntu:~/lab3$ diff input.txt output4.txt
linuxvirtualpc1@ubuntu:~/lab3$
```

3. 无名管道数据传输

父进程使用 fgets 函数从 input.txt 文件中读入数据流，利用 write()函数传递到 pipe 的写入端口；子进程使用 read()函数从 pipe 的读端口获得数据流，利用 fputs 函数将其写入到 output5.txt 文件中。

同时我们也记录了使用无名管道的时间。

```
linuxvirtualpc1@ubuntu:~/lab3$ sudo gcc -o 6 6.c
linuxvirtualpc1@ubuntu:~/lab3$ sudo ./6
##### unnamed pipe #######
##############################
I will run, I will climb, I will soar
I'm undefeated
Jumping out of my skin, pull the chord
Yeah I believe it
The past is everything we were don't make us who we are
so I'll dream until I make it real and all I see is stars
It's not until you fall that you fly.
##############################
timespan: 0.000039 seconds.
```

4. 有名管道数据传输

使用 mknod()函数创建有名管道。第一个进程 process1 使用只写方式打开有名管道，使用 fgets 函数从 input.txt 中读入数据，并用 write()写到管道中；另一个进程 process2 使用只读模式打开管道，利用 read()函数从管道中读入数据，使用 fputs 函数将数据存入文件 output6.txt。

同时我们也记录了使用无名管道的时间。

```
linuxvirtualpc1@ubuntu:~/lab3$ sudo gcc -o process2 process2.c
linuxvirtualpc1@ubuntu:~/lab3$ sudo ./process2
####### named pipe #############
##############################
I will run, I will climb, I will soar
I'm undefeated
Jumping out of my skin, pull the chord
Yeah I believe it
The past is everything we were don't make us who we are
so I'll dream until I make it real and all I see is stars
It's not until you fall that you fly.
##############################
timespan: 0.000033 seconds.
```

使用有名管道与无名管道的数据正确性：

```
linuxvirtualpc1@ubuntu:~/lab3$ diff input.txt output5.txt
linuxvirtualpc1@ubuntu:~/lab3$
linuxvirtualpc1@ubuntu:~/lab3$ diff input.txt output6.txt
linuxvirtualpc1@ubuntu:~/lab3$
```

六、程序及测试的改进与体会

1、存在不足及展望：

（1）在使用无名与有名管道时，一开始使用的是 fgetc & fputc，发现并不能对管道进行操作，只能够使用 read & write 来读写管道，因此浪费了很长时间。

（2）一开始对 read & write 和 fread & fwrite 参数把握并不是很到位，都是使用固定的字节数来传参，每次程序都会报错，后来发现，需要使用 read/fread 返回的字节数作为 write/fwrite 的输入参数，因为可能在读文件结尾时，并没有满足正好 1024 个字节，因此要以实际读写字节为准。

（3）在使用 fscanf & fprintf、fgetc & fputc、fgets & fputs 时，操作并不是很熟练，对于换行符的把握并不是很好，比如 fscanf & fprintf 不可以读换行符，只能够使用正则表达式匹配至行结尾，利用 fgetc 消去换行符，手动再加入换行符；fgetc & fputc 则比较通用，可以读任何字符，读到 EOF 就是文件结尾了，fgets & fputs 的话，一次匹配一个字符串，可以把换行符包含进入，也不用进行过多的处理。

2、体会

在学习操作系统之前，我对于文件的处理一般都是用首先用 fopen()，然后使用 fgetc & fputs 进行 txt 文件的复制与操作，从没有想过文件的操作竟然还有这么多学问。

这次的实验中学习了 read & write 与 fread & fwrite 之间的区别，同时也尝试了 fscanf & fprintf、fgetc & fputc、fgets & fputs 对文件的读写操作，了解了系统调用与用户态库函数的区别，也辨析了不同的行结构文件处理函数的异同，发现 fgets & fputs 相对来说操作比较方便，而且效率也很高，以后

在使用 C 语言进行文件操作的时候，我一定会好好利用这些函数，发挥其特点，提高代码的效率。

本次实验多亏了网络上的一些博客资料，清楚地解决了我的所有问题，而且讲解也都比较详细，让我对理论知识有了进一步的认识与拓展。

同时，这一次实验中学习了使用无名管道与有名管道，当初学习理论知识的时候，感觉这一种设计非常的精巧；在实验中亲身编码体验以后，更加感觉到了操作系统蕴含了大量的知识与科学家们智慧的结晶，需要我一直去探索其中的奥秘。

七、参考文献

1) 系统调用 read，write 和标准库 fread，fwrite 的区别 https://blog.csdn.net/ixiaochouyu/article/details/48340579
2) linux 编程学习 6-文件操作之用 read、write 实现文件复制拷贝功能 https://blog.csdn.net/d704791892/article/details/46391397
3) c 文件拷贝 fopen fwrite fread https://blog.csdn.net/u014338577/article/details/50997435
4) read 函数---------详解 https://blog.csdn.net/csdn_logo/article/details/46500065
5) C 语言 fread()与 fwrite()函数说明与示例 https://www.cnblogs.com/xudong-bupt/p/3478297.html

八、源代码及其注释

1. 调用 read & write、fread & fwrite 函数复制文件

使用 read & write 并且是一次一个字节模式

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <fcntl.h>
#include <time.h>

int main(int argc, char *argv[]) {
    clock_t begin_time, end_time;
    if (argc != 3) {    // 判断参数是否齐全
        printf("input param error\n");
```

```
        return -1;
    }

    int s_fd = open(argv[1], O_RDONLY);
    if (s_fd == -1) {    // 打开待复制文件
        printf("open %s error\n", argv[1]);
        return -1;
    }

    int d_fd = open(argv[2], O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
    if (d_fd == -1) {    // 打开复制到的文件
        printf("open %s error\n", argv[2]);
        return -1;
    }

    begin_time = clock(); // 记录文件复制前的时间
    char ch;
    while (true) {
        int rdRes = read(s_fd, &ch, 1);    // 逐个字符读文件
        if (rdRes == -1) {    // 读文件出错
            printf("read %s error\n", argv[1]);
            return -1;
        } else if (rdRes == 0) {    // 读文件完成
            printf("copy %s success\n", argv[1]);
            break;
        } else if (rdRes == 1) {    // 读文件过程中
            int wrRes = write(d_fd, &ch, 1);
            if (wrRes != 1) {    // 写文件出错
                printf("write %s error\n", argv[2]);
                return -1;
            }
        } else {
            printf("unknow error\n");
            return -1;
        }
    }
    end_time = clock(); // 记录文件复制后的文件
    // 打印使用了多少时间
    printf("timespan: %f seconds.\n", (double)(end_time-begin_time)/1000000);
    return 0;
}
```

使用 read & write 并且是一次 1024 字节模式

```
#include <unistd.h>
```

```
#include <sys/types.h>
#include <sys/stat.h>
#include <stdio.h>
#include <fcntl.h>
#include <time.h>
#define BUFE_SIZE 1024

int main(int argc, char *argv[]) {
    clock_t begin_time, end_time;
    if (argc != 3) {    // 判断参数是否齐全
        printf("input param error\n");
        return -1;
    }

    int s_fd = open(argv[1], O_RDONLY);
    if (s_fd == -1) {    // 打开待复制文件
        printf("open %s error\n", argv[1]);
        return -1;
    }

    int d_fd = open(argv[2], O_WRONLY | O_CREAT, S_IRUSR | S_IWUSR);
    if (d_fd == -1) {    // 打开复制到的文件
        printf("open %s error\n", argv[2]);
        return -1;
    }

    begin_time = clock(); // 记录文件复制前的时间
    buf[BUFE_SIZE];
    while (true) {
        int rdRes = read(s_fd, buf, sizeof(buf));    // 按照 buf 读文件
        if (rdRes == -1) {    // 读文件出错
            printf("read %s error\n", argv[1]);
            return -1;
        } else if (rdRes == 0) {    // 读文件完成
            printf("copy %s success\n", argv[1]);
            break;
        } else if (rdRes > 1) {    // 读文件过程中
            int wrRes = write(d_fd, buf, rdRes);
            if (wrRes != rdRes) {    // 写文件出错
                printf("write %s error\n", argv[2]);
                return -1;
            }
        } else {
            printf("unknow error\n");
```

```
                return -1;
            }
        }
        end_time = clock(); //  记录文件复制后的文件
        //  打印使用了多少时间
        printf("timespan: %f seconds.\n", (double)(end_time-begin_time)/1000000);
        return 0;
}
```

使用 fread & fwrite 并且是一次一字节模式

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <time.h>
#define READ_BUFF 1024

int main(int argc,char *argv[]){
        clock_t begin_time, end_time;
        if(argc < 3)    //  判断参数是否齐全
        {
            printf("input param error\n");
            return -1;
        }

        FILE * fileSourceHandler = NULL;
        FILE * fileDestHandler = NULL;
        fileSourceHandler = fopen(argv[1],"r");
        fileDestHandler = fopen(argv[2],"w");
        if(fileSourceHandler == NULL || fileDestHandler == NULL)    //  文件打开是否正常
        {
            printf("open %s or %s failed:%s\n",argv[1],argv[2],strerror(errno));
            return -2;
        }

        char buf[READ_BUFF];
        int nread;
        begin_time = clock();    //  记录文件复制前的时间
        while( nread = fread(buf,1,1,fileSourceHandler))    //  逐个字符读文件
        {
            fwrite(buf,1,nread,fileDestHandler);    //  逐个字符写文件
        }
        printf("copy %s success\n", argv[1]); //  写完成
        end_time = clock(); //  记录文件复制后的文件
```

```
    fclose(fileDestHandler);
    fclose(fileSourceHandler);
    // 打印使用了多少时间
    printf("timespan: %f seconds.\n", (double)(end_time-begin_time)/1000000);
    return 0;
}
```

使用 fread & fwrite 并且是一次 1024 字节模式

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <time.h>
#define READ_BUFF 1024

int main(int argc,char *argv[]){
    clock_t begin_time, end_time;
    if(argc < 3)   // 判断参数是否齐全
    {
        printf("./mycp sour dest\n");
        return -1;
    }

    FILE * fileSourceHandler = NULL;
    FILE * fileDestHandler = NULL;
    fileSourceHandler = fopen(argv[1],"r");
    fileDestHandler = fopen(argv[2],"w");
    if(fileSourceHandler == NULL || fileDestHandler == NULL)   // 文件打开是否正常
    {
        printf("open %s or %s failed:%s\n",argv[1],argv[2],strerror(errno));
        return -2;
    }

    char buf[READ_BUFF];
    int nread;
    begin_time = clock(); // 记录文件复制前的时间
    while( nread = fread(buf,1024,1,fileSourceHandler) )   // 1024 个字符读文件
    {
        fwrite(buf,1024,nread,fileDestHandler);     // 1024 个字符写文件
    }
    printf("copy %s success\n", argv[1]); // 写完成
    end_time = clock(); // 记录文件复制后的文件
    fclose(fileDestHandler);
    fclose(fileSourceHandler);
```

```
    // 打印使用了多少时间
    printf("timespan: %f seconds.\n", (double)(end_time-begin_time)/1000000);
    return 0;
}
```

2. 调用 fscanf & fprintf、fgetc & fputc、fgets & fputs 函数复制行文件

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
int main()
{
    FILE *fr; // 输入文件的标识字
    char ch;
    char buf[1024];
    clock_t begin_time, end_time;

    //########## fgets & fputs ###########
    FILE *fp1;
    if((fr=fopen("input.txt","r"))==NULL)
    {
        printf("error\n");
        exit(-1);
    }
    if((fp1=fopen("output1.txt","w"))==NULL)
    {
        printf("error\n");
        exit(-1);
    }
    printf("using fgets & fputs:\n");
    printf("##############################\n");
    begin_time = clock();
    while(fgets(buf,1024,fr)){
        printf("%s",buf);
        fputs(buf,fp1);
    }
    end_time = clock();
    printf("##############################\n");
    printf("timespan: %f seconds.\n\n", (double)(end_time-begin_time)/1000000);
    fclose(fr);
    fclose(fp1);

    //########## fscanf & fprintf ###########
    FILE *fp2;
    if((fr=fopen("input.txt","r"))==NULL)
```

```
{
	printf("error\n");
	exit(-1);
}
if((fp2=fopen("output2.txt","w"))==NULL)
{
	printf("error\n");
	exit(-1);
}
printf("using fscanf & fprintf:\n");
printf("############################\n");
begin_time = clock();
while(fscanf(fr,"%[^\n]",buf)>0){
	printf("%s\n",buf);
	fprintf(fp2,"%s",buf);
	fgetc(fr);
	fprintf(fp2,"%s","\n");
}
end_time = clock();
printf("############################\n");
printf("timespan: %f seconds.\n\n", (double)(end_time-begin_time)/1000000);
fclose(fr);
fclose(fp2);

//########## fgetc & fputs ###########
FILE *fp3;
if((fr=fopen("input.txt","r"))==NULL)
{
	printf("error\n");
	exit(-1);
}
if((fp3=fopen("output3.txt","w"))==NULL)
{
	printf("error\n");
	exit(-1);
}
printf("using fgetc & fputc:\n");
printf("############################\n");
begin_time = clock();
while((ch=fgetc(fr))!=EOF){
	printf("%c",ch);
	fputc(ch,fp3);
}
end_time = clock();
```

```
    printf("################################\n");
    printf("timespan: %f seconds.\n\n", (double)(end_time-begin_time)/1000000);
    fclose(fr);
    fclose(fp3);
    return 0;
}
```

Server：

```
#include <stdio.h>
#include <stdlib.h>
#include "msgcom.h"
int main()
{
    struct msgtype buf;      // create a message buffer
    int qid;
    if((qid=msgget(MSGKEY,IPC_CREAT|0666))==-1)       // get the message ID
        return(-1);
    while(msgrcv(qid,&buf,1000,5679,MSG_NOERROR)){ // receive message
        FILE *FileOpen=fopen("output4.txt","a");       // open the receive file
        if(FileOpen){
            printf("Server has received some data from the Client:\n");
            printf("######################################################\n");
            printf("%s",buf.msg);                          // print the message
            printf("######################################################\n\n");
            fputs(buf.msg,FileOpen);                       // put the message to the file
            fclose(FileOpen);
        }
        else{ // open file failure
            printf("Open file failed.\n");
            exit(1);
        }
    }
}
```

Client：

```
#include <stdio.h>
#include <stdlib.h>
#include "msgcom.h"
#include <time.h>
int main()
{
    struct msgtype buf; // create a buffer
    clock_t begin_time, end_time;
    int qid;
```

```
    qid=msgget(MSGKEY,IPC_CREAT|0666);              // get the queue of message
    buf.mtype=5679;
    FILE *FileOpen=fopen("input.txt","r");       // open the message file
    begin_time = clock();
    printf("Client has sent the following Message to the Server.\n");
    printf("##############################\n");
    if(FileOpen){
        while(fgets(buf.msg, 1000, FileOpen)){ // read message by line
            printf("%s",buf.msg);
            msgsnd(qid,&buf,sizeof(buf.msg),0); // send the message
            }
    }
    else{
        printf("Open file failed.\n");
        exit(1);
    }
    end_time = clock();
    printf("##############################\n");
    printf("timespan: %f seconds.\n", (double)(end_time-begin_time)/1000000);
    fclose(FileOpen);
}
```

Msgcom.h:

```
#include<errno.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/msg.h>
#define MSGKEY 5678
struct msgtype{
      long mtype;           // message type definition
      char msg[1000];    // message block
};
```

3. 无名管道传输数据

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <time.h>
int main(){
            FILE* fr=fopen("input.txt", "r");
            FILE* fw=fopen("output5.txt","w");
            clock_t begin_time, end_time;
            int fd[2];
            char buf[1024];
```

```
        pipe(fd);
        if(fork())
        {           //父进程
                close(fd[0]);     //关闭管道读端
                printf("##### unnamed pipe #######\n");
                printf("##############################\n");
                begin_time =clock();
                while(fgets(buf,1024,fr)){
                        write(fd[1],buf,1024);   // 在管道写
                        printf("%s",buf);
                }
                end_time = clock();
                printf("##############################\n");
                printf("timespan: %f seconds.\n\n", (double)(end_time-begin_time)/1000000);
                close(fd[1]);     //关闭管道写端
                fclose(fr);
        }
        else{     //子进程
                close(fd[1]);     //关闭管道写端
                while(read(fd[0],buf,1024)){       // 从管道读
                        fputs(buf,fw);
                }
                close(fd[0]);     //关闭管道读端
                fclose(fw);
        }
}
```

4. 有名管道传输数据

Process1：

```
#include <fcntl.h>
#include <sys/stat.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>
#define BUFSIZE 1024
int main(){
    int fd;
    char buf[BUFSIZE];
    mknod("named_pipe",S_IFIFO|0666,0); // 创建一个有名管道
    fd=open("named_pipe",O_WRONLY); // 打开有名管道，只写模式
    FILE* fp=fopen("input.txt","r"); // 从 input.txt 读入数据
    while(fgets(buf,sizeof(buf),fp)){
        write(fd,buf,strlen(buf)+1); // 数据写入管道
    }
```

```
    close(fd);
    fclose(fp);
}
```

Process2：

```
#include <fcntl.h>
#include <unistd.h>
#include <stdio.h>
#include <time.h>

#define BUFSIZE 1024
int main(){
    int fd,n;
    clock_t begin_time, end_time;
    char buf[BUFSIZE];
    fd=open("named_pipe",O_RDONLY); // 打开有名管道，只写模式
    FILE* fp=fopen("output6.txt","w"); // 数据流出文件
    printf("####### named pipe ##############\n");
    printf("################################\n");
    begin_time = clock();
    while((n=read(fd,buf,sizeof(buf)))>0){ // 从有名管道读入数据
        fputs(buf,fp);  // 输出到文件
        printf("%s",buf);
    }
    end_time = clock();
    printf("################################\n");
    printf("timespan: %f seconds.\n\n", (double)(end_time-begin_time)/1000000);
    close(fd);
    fclose(fp);
}
```