1. open()函数

功能描述：用于打开或创建文件，在打开或创建文件时可以指定文件的属性及用户的权限等各种参数

所需头文件：#include <sys/types.h>, #include <sys/stat.h>,
#include <fcntl.h>

函数原型：int open(const char *pathname, int flags, int perms)

参数：

(1) pathname: 被打开的文件名（可包括路径名，如"dev/ttyS0"）

(2) flags：文件打开方式

O_RDONLY：以只读方式打开文件；

O_WRONLY：以只写方式打开文件；

O_RDWR：以读写方式打开文件；

O_CREAT：如果打开的文件不存在，创建一个新的文件，并用第三个参数为其设置权限；

O_EXCL：如果使用 O_CREAT 时文件存在，则返回错误消息。这一参数可测试文件是否存在。此时 open 是原子操作，防止多个进程同时创建同一个文件；

O_NOCTTY：使用本参数时，若文件为终端，那么该终端不会成为调用 open()的那个进程的控制终端；

O_TRUNC：若文件已经存在，那么会删除文件中的全部原有数据，并且设置文件大小为 0；

O_APPEND：以添加方式打开文件，在打开文件的同时，文件指针指向文件的末尾，即将写入的数据添加到文件的末尾；

O_NONBLOCK：如果 pathname 指的是一个 FIFO、一个块特殊文件或一个字符特殊文件，则此选择项为此文件的本次打开操作和后续的 I/O 操作设置非阻塞方式；

O_SYNC：使每次 write 都等到物理 I/O 操作完成；

O_RSYNC：read 等待所有写入同一区域的写操作完成后再进行；

在 open()函数中，falgs 参数可以通过“|”组合构成，但前 3 个标准常量（O_RDONLY，O_WRONLY，和 O_RDWR）不能互相组合。

(3) perms：被打开文件的存取权限，可以用两种方法表示，可以用一组宏定义：

S_I(R/W/X)(USR/GRP/OTH)，其中 R/W/X 表示读写执行权限，

USR/GRP/OTH 分别表示文件的所有者/文件所属组/其他用户，如

S_IRUUR|S_IWUUR|S_IXUUR,（-rex------）；

也可用八进制 800 表示同样的权限

返回值：

成功：文件描述符 fd

失败：-1

当前文件偏移量 cfo(current file offset) in Linux：

cfo 表示文件开始处到文件当前位置的字节数，一般是一个非负整数。对文件的 read()、write()通常始于 cfo，即根据 cfo 指向的文件位置，进行读写操作，并使 cfo 值增大，cfo 的增量为读写的字节数。

使用 open()操作打开文件时，文件的 cfo 初始化为 0，指向文件开始位置，除非参数 flags=O_APPEND；使用系统调用 lseek()可以改变文件的 cfo，即改变文件读写指针位置。

2. close()函数

功能描述：用于关闭一个被打开的的文件

所需头文件： #include <unistd.h>

函数原型：int close(int fd)

参数：fd 文件描述符

函数返回值：0，成功，-1，出错

3. read()函数

功能描述：从文件读取数据

所需头文件：#include <unistd.h>

函数原型：ssize_t read(int fd, void *buf, size_t count)

参数：

(1) fd：读取数据的文件的文件描述符；

(2) buf：内存缓冲区，用于存放从文件读取的数据；

(3) count：执行一次 read 操作，应该读取的字节数；

返回值：

返回所读取的字节数；

0，读到文件尾部 EOF；-1，出错。

读操作从指针 cfo 处开始，在成功返回之前，cfo 增加，增量为实际读取到的字节数。

以下下述情况将导致读取到的字节数小于 count：

A. 读取普通文件时，读到文件末尾时已读取的字节数还不够 count。例如，文件只有 30 bytes，而 count=100 bytes，那么实际读到的只有 30 bytes，read()返回 30;

B. 从终端设备(terminal device)读取时，一般情况下每次只能读取一行；

C. 从网络读取时，网络缓存可能导致读取的字节数小于 count 字节；

D. 读取 pipe 或者 FIFO 时，pipe 或 FIFO 中的字节数可能小于 count；

E. 从面向记录(record-oriented)的设备读取时，某些面向记录的设备（如磁带）每次最多只能返回一个记录。

F. 在读取了部分数据时被信号 signal 中断。

4. write()函数

功能描述：向文件写入数据

所需头文件：#include <unistd.h>

函数原型：ssize_t write(int fd, void *buf, size_t count)

参数：

(1) fd：写入数据的文件的文件描述符；

(2) buf：内存缓冲区，用于存放写入文件的数据；

(3) count：调用 write 操作，应该写入文件的字节数。

返回值：

成功：写入文件的字节数；

失败：-1，当磁盘空间满，或者写入数据后超过文件大小限制。

对于普通文件，写操作始于文件读写指针 cfo 。如果打开文件时使用了 O_APPEND（从尾部追加写入），则每次写操作都将数据写入文件末尾。成功写入后，cfo 增加，增量为实际写入的字节数。

5. lseek()函数

功能描述： 在文件描述符 fd 指定的文件中，将文件读写指针 cfo 定位到相应位置

所需头文件： #include <unistd.h>，#include <sys/types.h>

函数原型：off_t lseek(int fd, off_t offset, int whence)

参数：

(1) fd：文件描述符；

(2) offset：偏移量，读写操作所需要移动的距离，单位是字节 byte。值为正，指针移向文件尾部；值为正，指针移向文件头部；

(3) whence：定位参照点

SEEK_SET：参照点为文件开始处 0，定位后 cfo 新位置为偏移量 offset，即 cfo=offset；

SEEK_CUR：参照点为读写指针 cfo 的当前位置，定位后 cfo 新位置为其当前位置加上偏移量 offset，即 cfo=cfo+offset；

SEEK_END：参照点为文件结尾，新位置为文件大小加上偏移量 offset，即 cfo=size-of(fd)+offset。

返回值：

成功：cfo 的新位置，即返回当前读取位置在文件中的绝对位置

失败：-1

6. 示例程序

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include<sys/types.h>
#include<sys/stat.h>
#include <errno.h>

#define BUFFER_SIZE 128                    //每次读写缓存大小，影响运行效率
#define SRC_FILE_NAME "src_file.txt"       //源文件名
#define DEST_FILE_NAME "dest_file.txt"     //目标文件名
#define OFFSET 0                           //文件指针偏移量

int main()
{
    int src_file,dest_file;
    unsigned char src_buff[BUFFER_SIZE];
    unsigned char dest_buff[BUFFER_SIZE];
    int real_read_len = 0;
    char str[BUFFER_SIZE] = "this is a testabout\nopen()\nclose()\nwrite()\nread()\nlseek()\nend
                            of the file\n";

    //创建源文件
    src_file=open(SRC_FILE_NAME, O_RDWR|O_CREAT,
                  S_IRUSR|S_IWUSR|S_IRGRP|S_IROTH);
    if(src_file<0)
    {
        printf("open file error!!!\n");
        exit(1);
    }
    //向源文件中写数据
    write(src_file, str, sizeof(str));
    //创建目标文件
```

```
dest_file=open(DEST_FILE_NAME, O_RDWR|O_CREAT,
                                    S_IRUSR|S_IWUSR|S_IRGRP|S_IROTH);
 if(dest_file<0)
 {
        printf("open file error!!!\n");
        exit(1);
  }

     lseek(src_file, OFFSET, SEEK_SET); //将源文件的读写指针移到起始位置
     while((real_read_len=read(src_file, src_buff, sizeof(src_buff)))>0)
     {
          printf("src_file:%s", src_buff);
          write(dest_file,src_buff,real_read_len);
      }
      lseek(dest_file, OFFSET, SEEK_SET); //将目标文件的读写指针移到起始位置
      while((real_read_len=read(dest_file,dest_buff,sizeof(dest_buff)))>0); //读取目标文件的
内容
      printf("dest_file:%s", dest_buff);
      close(src_file);
      close(dest_file);
      return 0;
      }
```

结果 如下：

```
src_file: this is a test about
open()
close()
write()
read()
lseek()
end of the file
dest_file: this is a test about
open()
close()
write()
read()
lseek()
```

end of the file