1. 系统调用方法
2. 库函数调用方法

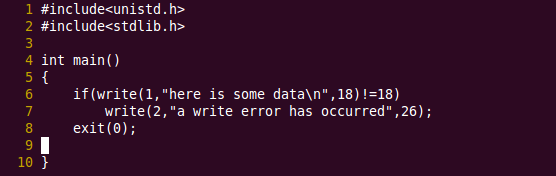
## 【系统调用】

### 文件描述符

０：标准输入

１：标准输出

２：标准错误



### write系统调用

size\_t write(int fildes, const void \*buf, size\_t nbytes);

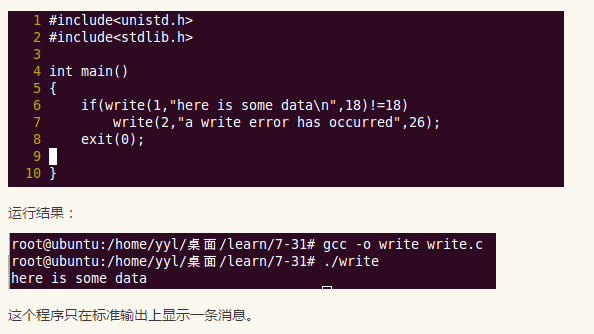
参数说明：

fildes：文件描述符，标识了要写入的目标文件。例如:fildes的值为1，就像标准输出写数据，也就是在显示屏上显示数据；如果为 2 ，则想标注错误写数据。

\*buf：待写入的文件，是一个字符串指针。

nbytes：要写入的字符数。

函数返回值：size\_t 返回成功写入文件的字符数。需要指出的是，write可能会报告说他写入的字节比你所要求的少。这并不一定是个错误。在程序中，你需要检查



### read系统调用

size\_t read(int fildes, void \*buf, size\_t nbytes);

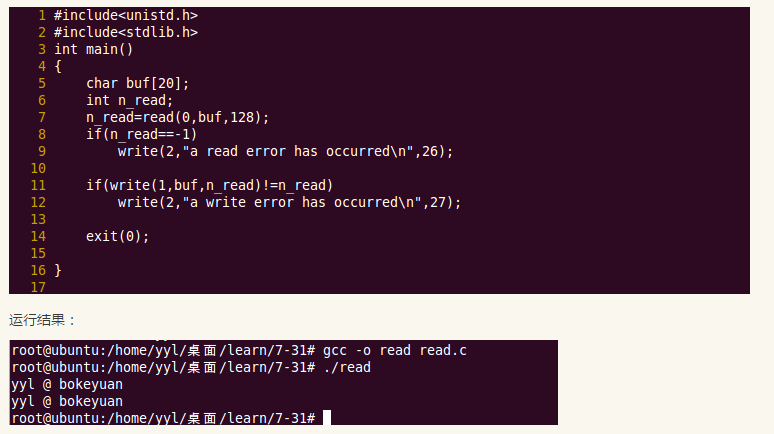
参数说明：

fildes：文件描述符，标识要读取的文件。如果为0，则从标准输入读数据。类似于scanf（）的功能。

\*buf：缓冲区，用来存储读入的数据。

nbytes：要读取的字符数。

返回值：size\_t返回成功读取的字符数，它可能会小于请求的字节数。



### open系统调用

int open(const char \*path, int oflags);

int open(const char \*path, int oflags, mode\_t mode );

O\_RDONLY: 以只读方式打开文件

O\_WRONLY：以只写的方式打开文件

O\_RDWR：以读写的方式打开文件

O\_CREAT：若打开的文件不存在，则创建该文件

O\_EXCL：如果打开文件是设置了O\_CREAT，但是该文件存在，则导致调用失败

O\_TRUNC：如果以只写或只读方式打开一个已存在的文件，将该文件截至0

O\_APPEND：追加的方式打开文件

O\_NONBLOCK：用于非堵塞接口i/o

O\_NODELAY

O\_SYNC:当数据被写入外存或者其他设备后，操作才返回。

其中mode是打开的权限

S\_IRWXU 00700 设置文件所有者的读，写，执行权限

S\_IRWXG 00070 设置文件所在用户组的读，写，执行权限

S\_IRWXO 00007 设置其他用户的读，写，执行权限

S\_IRUSR 00400 设置文件所有者的读权限

S\_IWUSR 00200 设置文件所有的写权限

S\_IXUSR 00100 设置文件所有者的执行权限

S\_IRGRP 00040 设置用户组的读权限

S\_IWGRP 00020 设置用户组的写权限

S\_IXGRP 00010 设置用户组的执行权限

S\_IROTH 00004 设置其他用户的读权限

S\_IWOTH 00002 设置其他用户的写权限

S\_IXOTH 00001 设置其他用户的执行权限



### close系统调用

close系统调用用于“关闭”一个文件，close调用终止一个文件描述符fildes以其文件之间的关联。文件描述符被释放，并能够重新使用。

close成功返回1，出错返回-1.

#Include<unistd.h>

int close(int fildes);

### ioctl系统调用

ioctl提供了一个用于控制设备及其描述符行为和配置底层服务的接口。终端、文件描述符、甚至磁带机都可以又为他们定义的ioctl，具体

细节可以参考特定设备的使用手册。

下面是ioctl 的函数原型

#include<unistd.h>

int ioctl(int fildes, int cmd,,,,,,);

ioctl对描述符fildes指定的对象执行cmd 参数中所给出的操作。

1. 其他和文件管理有关的系统调用

还有许多其他的系统调用能对文件进行操作。

几个常用的如：lseek（）对文件描述符fildes指定文件的读写指针进行设置，也就是说，它可以设置文件的下一个读写位置。

fstat,stat,lstat 是和文件描述符相关的函数操作，这里就不做介绍。

dup，dup2系统调用。dup提供了复制文件描述符的方法，使我们能够通过两个或者更多个不同的文件描述符来访问同一个文件。这可以用于

在文件的不同位置对数据进行读写。

# 检测例题：

#include <fcntl.h>  
#include <stdio.h>  
#include <unistd.h>  
  
void write\_file(int fd){  
    char buf[]="abcde\n";  
    write(fd,buf,sizeof(buf));  
  
    close(fd);  
}  
  
void read\_file(){  
    int fd;  
    char \*path="/home/zhf/test1.txt";  
    char result[20];  
    fd=open(path,O\_RDONLY);  
    read(fd,result,10);  
    printf("The content is %s",result);  
    close(fd);  
}  
  
int create\_file(){  
    int fd;  
    char \*path="/home/zhf/test1.txt";  
    fd=creat(path,00777);  
    return fd;  
}

## 【库函数调用】

### fopen函数

FILE \*fopen(const char \*filename, const char \*mode);

参数说明：

\*filename：打开文件的文件名

\*mode：打开的方式

r 以只读方式打开文件，该文件必须存在。

　　r+ 以可读写方式打开文件，该文件必须存在。

　　rb+ 读写打开一个二进制文件，允许读数据。

　　rw+ 读写打开一个文本文件，允许读和写。

　　w 打开只写文件，若文件存在则文件长度清为0，即该文件内容会消失。若文件不存在则建立该文件

　　w+ 打开可读写文件，若文件存在则文件长度清为零，即该文件内容会消失。若文件不存在则建立该文件。

fopen在成功是返回一个非空的FILE \*指针。失败返回NULL

### fread/fwrite函数

fread函数从文件流中读取数据，对应于系统调用中的read；fwrite函数从文件流中写数据，对应于系统调用中的write

函数原型：

#include<stdio.h>

size\_t fread(void \*ptr, size\_t size, size\_t nitems, FILE \*stream);

参数说明：

\*ptr 要读取数据的缓冲区，也就是要存放读取数据的地方。

size：指定每个数据记录的长度。

nitems： 计数，给出要传输的记录个数。

返回值：成功读取到数据缓冲区的记录个数，当到达文件尾时，他的返回值可能会消耗与nitems，甚至可以是0

size\_t fwrite(const coid \*ptr, size\_t size , size\_t nitimes, FILE \*stream);

他从指定的数据缓冲区ptr中把数据写入文件流，返回成功写入的记录个数。

### fclose函数

fclose函数关闭指定的文件流stream，这个操作会使所有未写出的数据都写出。因为stdio库函数会对数据进行缓冲，所有调用fclose函数是很重要的。

如果程序需要确保数据已经全部写出，就应该调用fclose函数。虽然程序正常结束时，也会自动的调用fclose函数，但这样就不能检测出调用fclose所产生的错误了。

函数原型如下：

#include<stdio,h>

int fclose(FILE \*stream);

### fflush函数

fflush函数的作用是把文件流中所有未写出的数据全部写出。 处于效率考虑，在使用库函数的时候会使用数据缓冲区，当缓冲区满的时候才进行写操作。使用fflush函数

可以将缓冲区的数据全部写出，而不关心缓冲区是否满。fclose的执行隐含调用了fflush函数，所以不必再fclose执行之前调用fflush。

函数原型：

#include<stdio.h>

int fflush(FILE \*stream);

# 检测题：

void fopen\_file()

{

int i;

char ch;

char \*str;

FILE \*fp;

char \*path="/home/zhf/zhf/c\_prj/c\_test.txt";

char result[100];

fp=fopen(path,"r");

str=fgets(result,20,fp);

printf("%s\n",str);

fclose(fp);

}