基础IO：标准库的回顾/系统调用接口/文件描述符/文件系统/动态库与静态库的打包与使用

标准库的接口回顾：

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

mode：r　r+

w/w+ ：这个操作会清空文件原有内容(若不想清空原有内容可以使用r+/a+）

a/a+：总是将数据写到文件末尾，追加，a文件不存在会创建新文件

fread(char *buffer,size_t bsize,FILE *fp)

返回的读到的块的个数，如果读到的数据不足完整的一块，则返回0；(通常会设置块大小为1，快个数是想要的数据长度）

fwrite(char *data,size_t bsize,size_t bnum,FILE *fp)

fwrite返回的是块个数，当块大小为1的时候返回的是写入文件的字节长度

fwrite/fread：nsize--块大小，nmem--块个数，nsize*nmeme--真正要读写的数据大小；返回值是成功操作的块的个数。

推荐将块大小设置为1---返回值就是实际参数的数据大小与将文件读取到末尾的返回值就区分出来

fseek(FILE *fp,long offset,int whence);跳转文件读写位置

whence:SEEK_SET/SEEK_CUR/SEEK_END

fclose(FILE *fp)；关闭文件

rr+ww+：打开文件后，文件读写位置是在起始位置，注意不要覆盖写人

文件内容中游客可能会有空字符（\0)，从文件读取数据进行处理，需要注意；

fgets/fprintf　printf/sprintf/fprintf

学习系统调用IO接口：open/read/write/lseek/close

int **open**(char *filename,int flag,int mode)

filename:要打开的文件名称

flag：选项参数　　mode：文件的给定权限

必选参数：O_RDONLY / O_WRONLY / O_RDWR 只能选择其一

可选参数：

O_CREAT:文件存在则打开，不存在则创建新的
O_EXCL:通常与O_CREAT同时使用，表若文件不存在则创建，已经存在则会报错返回
O_TRUNC：打开文件的同时，清空文件的原有内容
O_APPEND：写入数据时，总是写入文件末尾（追加写）

r+：O_RDWR
w+：O_RDWR|O_TRUNC|O_CREAT
a+：O_RDWR|O_APPEND|O_CREAT

返回值：正整数--文件描述符--后期对文件操作的操作句柄，失败返回-1

ssize_t **write**(int fd，char* data，size_t count):有符号整形
fd:文件描述符--open返回的文件操作句柄
data：想要写入的数据首地址
count：想要写人的数据长度
返回值：>0表示实际写入的数据长度（有可能与想要写入的长度不符）；-1表示出错
size_t:无符号整形数据int，4个字节

off_t **fseek**(int fd,long offset,int whence)
lseek返回值是跳转后的位置相对于文件起始位置的偏移量

ssize_t **read**((int fd，char* data，size_t count):
fd:文件描述符--open返回的文件操作句柄
data：一块缓冲区的首地址，用于存储读取的数据
count：想要读取的数据长度
返回值：>0表示实际读取的数据长度；==0：读到文件末尾；-1：出错

close(int fd);

注意事项：

1、如果open使用O_CREAT，那么就一定要加上第三个权限参数

2、权限参数-0664，不要忘了前边这个0；

文件描述符：

探讨文件描述符是什么：

文件描述符实际上就是内核中一个文件描述信息结构体数组的下标

进程通过pcd找到files_struct，进而找到数组fd_arry[]，在通过描述符找到具体的文件描述信息

一个程序运行起来之后，默认会打开三个文件：

标准输入	标准输出	标准错误
0	1	2
stdin	stdout	stderr

文件描述符与文件流指针的关系：

文件流指针是一个结构体，并且这个结构体中封装了一个成员就是文件描述符

我们通常所说的缓冲区其实也是每一个FILE结构体中自带的缓冲区--用户态缓冲区

若当前进程运行的是程序员这自己写的代码，则进程运行在用户态

若当前进程运行的程序时系统调用/操作系统实现的功能，则进程运行在内核态

重定向原理：

重定向的实现：要操作的stdout/文件描述符都没有，但是通过改变这个文件描述符对应下标的文件描述信息，进而实现改变当前所操作的文件的目的

int dup2(int oldfd,int newfd);

将newfd这个描述符下标对应的文件描述信息

在minishell中如何实现重定向?

>> 追加

> 清空

重定向的文件描述符不会随着程序替换而初始化

文件系统：磁盘中的文件存储管理

存储一个文件的流程：

1、通过超级块获取到databitmap的地址，通过databitmap获取到空闲的磁盘块号，将文件数据写入指定的磁盘块；

2、通过超级块获取到inodebitmap的地址，通过inodebitmap获取一个空闲的inode节点，将文件的元信息写入其中

3、将文件的文件名以及inode节点号的对应信息(目录项)，写入到这个文件所在目录中

获取一个文件数据的流程：

1、通过文件名，在所在目录中找到自己的目录项(包含当前文件的inode节点号)

2、通过超级块找到inode-table区域，通过inode节点号找到相应的inode节点

3、通过inode节点，就能找到文件数据存储的磁盘块号

4、找到磁盘块，读取数据

软链接文件和硬链接文件：访问源文件

软链接文件：ln -s sflie dfile

硬链接文件：ls sfile dfile

区别：

软链接文件：一个独立的文件，文件数据中保存的是源文件的路径，通过路径访问源文件

硬链接文件：跟源文件没有什么区别，与源文件共用同一个inode节点，通过访问同一个inode，实现获取文件数据

软链接就是一种类似于win下边快捷方式的文件，而硬链接就是一个文件别名

删除源文件，则软链接文件无效，而硬链接文件依然可以访问文件数据
软链接可以跨分区建立，但是硬链接不可以
软链接可以对目录创建，但是硬连接不可以

动态库与静态库的生成与使用
生成：将大量代码进行打包
动态库：1、gcc -c child.c -o child.o　2、gcc -shared child.o -o libmychild.so
-fPIC：产生与位置无关代码
-c：只进行预处理，编译，汇编，但是不链接
-shared：生成的是动态库而不是可执行程序
动态库的命名方式：以lib为前缀，以.so为后置，中间是名称

静态库：1、gcc -c child.c -o child.o　2、 ar -rc libmychild.a child.o
ar 生成静态库的命令　-c：创建　-r：模块替换
命名方式：以lib为前缀，以.a为后置，中间是名称
使用：
生成可执行程序的时候链接使用：
1、将库文件放到指定的路径下：/lib64　　/usr/lib64
2、设置环境变量：export LIBRARY_PATH=${LIBRARY_PATH}:./
3、设置gcc选项： gcc main.c -o main –L./ -lmychild
gcc _L选项，用于指定库的链接搜索路径

运行可执行程序的时候加载使用：仅针对动态链接生成的可执行程序
1、将库文件放到指定的路径下：/lib64　　/usr/lib64
2、设置环境变量：export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:./

使用第三方库，这个库通常只有你一个程序使用，因此更多的会使用静态库

复习：

基础IO:文件描述符/文件系统

回顾标准库IO: fopen/fread/fwrite/fseek/fclose

fopen: "r/r+ /w/w+/a/a+"

注意事项:

fread/fwrite这两个函数是按块操作文件数据;返回值返回的是完整操作的块个数

系统调用IO: open/read/write/lseek/close/umask

open(const chat *filename, int flag, mode_ t mode)

flag: O_RDONLY/O_WRONLY/O_RDWR | O _CREAT|O_TRUNC|O_APPEND

mode:文件给定的创建权限　mode & (~ mask)

文件描述符: open返回的文件操作句柄

文件描述符:实际上是内核中文件描述信息表中的-一个数组下标

文件描述符:系统调用的操作句柄

文件流指针:库函数的操作句柄

文件流指针是一个结构体， 内部封装了文件描述符

文件描述符的重定向:通过改变文件描述符这个下标所对应的文件描述信息结构，来改变所操作的文件dup2(int oldfd, int newfd)

文件系统:管理磁盘分区中的文件存取

ext2文件系统将磁盘分区分成了多个区域:超级块/inode_ _bitmap/data_ bitmap/inode/data

文件的存储流程:通过超级块找个各个区域的起始地址，通过inode _bitmap快速找到空闲inode节点，通过data_ bitmap快速找到空闲数据块;空闲数据块存储文件数据，inode节 点存放文件的元信息以及数据存储的块号;组织-一个目录项信息,写入到所在目录的文件中;

文件数据的获取流程:通过文件的路径名,到所在目录文件中查找目录项，获取到文件的inode节点号,在inode区域中找到指定的inode节点，获取到文件的元信息以及数据块号，进而从指定的数据块获取文件数据

软链接文件/硬链接文件:给源文件创建一个软/硬链接文件，通过这两个文件可以访问源文件软连接文件:是一一个独立的文件，有自己的inode节点，文件数据中保存的是源文件路径

硬链接文件:只是文件的一个别名(目录项)，与源文件没什么区别，都是文件的目录项，共用同一个inode节点

动态库与静态库的生成与使用
本质:将一堆已经实现的代码汇编完毕后，打包起来
1.将所有的源码文件进行编译汇编完成gcc -fPIC -C child.c -o child.o
2.动态库的打包: gcc --share child.o -o libmychild.so
3.静态库的打包: ar -cr libmychild.a child.o

生成可执行程序时链接使用:gcc main.c -o main -lmychild
1.将库文件放置到指定路径下: /lib64
2.设置环境变量: export LIBRARY PATH= $LIBRARY_ PATH./lib
3.使用gcc选项指定库文件所在路径: gcc main.c -0 main -L/ib -lmychild
运行可执行程序时加载使用:针对使用动态库的程序
1.将库文件放置到指定路径下: /ib64
2.设置环境变量: export LD_ LIBRARY PATH=$LIBRARY PATH:/lib