1. 文件的偏移

#include <stdio.h>

int fseek(FILE \*stream, long offset, int whence);

功能：光标的移动

参数：stream 流

offset 偏移量 正数—文件结尾方向

负数—文件开头方向

whence SEEK\_SET 文件开始位置

SEEK\_CUR 文件当前位置

SEEK\_END 文件结尾位置

返回值： 成功0 失败-1

long ftell(FILE \*stream);

功能：检测光标所在位置

参数：流

返回值：成功偏移量，失败-1

void rewind(FILE \*stream);

功能：光标移动到文件开始位置

1. 出错打印

#include <string.h>

char \*strerror(int errnum);

功能：打印出错信息

参数：errno (系统定义好了，包含头文件<errno.h>就可以了)

返回值：错误信息首地址

#include <stdio.h>

void perror(const char \*s);

功能：打印错误信息

参数：s 自己的错误信息

1. 文件描述符

对于内核而言，所有打开文件都由文件描述符引用。

文件描述符是一个**非负整数**。当打开一个现存文件或创建一个新文件时，内核向进程返回一个文件描述符。

当读、写一个文件时，用open或creat返回的文件描述符标识该文件，将其作为参数传送给read或write。

1. open

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

int open(const char \*pathname, int flags);

int open(const char \*pathname, int flags, mode\_t mode);

功能：获得文件描述符

参数：pathname 文件的路径

flags O\_RDONLY 只读

O\_WRONLY 只写

O\_RDWR 可读可写

（以上三个互斥）

O\_CREAT 创建（此时需要第三个参数）

O\_EXCL 创建时如果文件存在报错（错误原因已经存在）

O\_TRUNC 清空

mode 权限

返回值：成功文件描述符 失败-1

1. close

#include <unistd.h>

int close(int fd);

功能：关闭文件描述符

参数：fd 文件描述符

返回值：成功0 失败-1

1. 读写

#include <unistd.h>

ssize\_t write(int fd, const void \*buf, size\_t count);

功能：向文件中写

参数：fd 文件描述符

buf 要写入内容的首地址

count 期望值（字节数）

返回值：成功写入的字节数 失败-1

#include <unistd.h>

ssize\_t read(int fd, void \*buf, size\_t count);

功能：从文件中读

参数：fd 文件描述符

buf 读到数据的存放位置

count 期望值（字节数）

返回值：成功实际读到的字节数 失败-1

1. 文件偏移

#include <sys/types.h>

#include <unistd.h>

off\_t lseek(int fd, off\_t offset, int whence);

功能：光标的偏移

参数：fd 文件描述符

offset 偏移量

whence SEEK\_SET 开头位置

SEEK\_CUR 当前位置

SEEK\_END 结尾位置

返回值：成功偏移量 失败

1. 文件属性

#include <sys/types.h>

#include <sys/stat.h>

#include <unistd.h>

int stat(const char \*path, struct stat \*buf);

int fstat(int fd, struct stat \*buf);

int lstat(const char \*path, struct stat \*buf);

* 三个函数的返回：若成功则为0，若出错则为-1,并且设置errno.
* 给定一个pathname的情况下：
  + stat函数返回一个与此命名文件有关的信息结构
  + fstat函数获得已在描述符filedes上打开的文件的有关信息

lstat函数类似于stat，但是当命名的文件是一个符号连接时，lstat返回该符号连接的有关信息，而不是由该符号连接引用的文件的信息

1. 目录相关函数

#include <sys/types.h>

#include <dirent.h>

DIR \*opendir(const char \*name);

功能：获得目录流

参数：目录

返回值：成功目录流 失败NULL

#include <sys/types.h>

#include <dirent.h>

int closedir(DIR \*dirp);

功能：关闭目录流

参数：dirp 目录流

返回值：成功0 失败-1

#include <dirent.h>

struct dirent \*readdir(DIR \*dirp);

功能：读目录

参数：目录流

返回值：成功读到文件的信息 失败NULL

1. 库
2. linux下的库有两种：静态库和共享库（动态库）。二者的不同点在于代码被载入的时刻不同。
   1. 静态库在程序编译时会被连接到目标代码中，程序运行时将不再需要该静态库，因此体积较大。

优点：移植性好

缺点：库升级不方便，浪费空间和资源

* 1. 动态库在程序编译时并不会被连接到目标代码中，而是在程序运行是才被载入，因此在程序运行时还需要动态库存在，因此代码体积较小。

优点：库升级方便

缺点：移植性差

1. 静态库的制作
2. 生成目标文件

gcc -c fun.c -o fun.o

1. 生成库

ar crs libfun.a fun.o //(fun1.o)

lib.a是静态库的固定格式

1. 编译

gcc main.c –o main -L. –lfun

-L 指定库的路径

-l 指定库的名字

【13】动态库的制作

1. 生成目标文件

gcc –c –fPIC fun.c –o fun.o

1. 生成库

gcc –shared –o libfun.so fun.o

1. 编译

gcc main.c –o main –L. –lfun

1. 加载库的方式

(1) 把库拷贝到/usr/lib和/lib目录下。

(2) 在LD\_LIBRARY\_PATH环境变量中加上库所在路径。

export  LD\_LIBRARY\_PATH=/home/linux/day\_2/dynamic/

export LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:.

(3) 添加/etc/ld.so.conf.d/\*.conf文件，把库所在的路径加到文件末尾，并执行ldconfig刷新。