	标准 IO	文件 IO
定义	在 C 库中定义输入输出的函数	在 posix 中定义的输入输出的函数
特点	有缓冲机制 围绕流操作，FILE* 默认打开三个流：stdin/stdout/stderr 只能操作普通文件	没有缓冲机制 围绕文件描述符操作，int 非负整数 默认打开三个文件描述符：0/1/2 除 d 外其他任意类型文件
函数接口	打开文件：fopen/freopen 读写文件：fgetc/fputc、fgets/fputs、fread/fwrite 关闭文件：fclose 文件定位：rewind、fseek、ftell	打开文件：open 读写文件：read、write 关闭文件：close 文件定位：lseek

文件属性获取

```
int stat(const char *path, struct stat *buf);
功能：获取文件属性
参数：path：文件路径名
        buf：保存文件属性信息的结构体
返回值：成功：0
       失败：-1
struct stat {
         ino_t     st_ino;      /* inode 号 */
         mode_t    st_mode;     /* 文件类型和权限 */
         nlink_t   st_nlink;    /* 硬链接数 */
         uid_t     st_uid;      /* 用户 ID */
         gid_t     st_gid;      /* 组 ID */
         off_t     st_size;     /* 大小 */
         time_t    st_atime;    /* 最后访问时间 */
         time_t    st_mtime;    /* 最后修改时间 */
         time_t    st_ctime;   /* 最后状态改变时间 */
    };
```

目录操作

围绕目录流进行操作，DIR *

由于 windows 和 linux 的本质不同，因此二者库的二进制是不兼容的

2. 库的分类

静态库和动态库，本质区别是代码被载入时刻不同。

1）静态库在程序编译时会被连接到目标代码中。

优点：程序运行时将不再需要该静态库；运行时无需加载库，运行速度更快

缺点：静态库中的代码复制到了程序中，因此体积较大；

静态库升级后，程序需要重新编译链接

2）动态库是在程序运行时才被载入代码中。

优点：程序在执行时加载动态库，代码体积小；

程序升级更简单；

不同应用程序如果调用相同的库，那么在内存里只需要有一份该共享库的实例。

缺点：运行时还需要动态库的存在，移植性较差

3. 库的制作

3.1 静态库的制作

1-将源文件编译生成目标文件

gcc -c add.c -o add.o

2-创建静态库用 ar 命令，它将很多.o 转换成.a

ar crs libmyadd.a add.o

静态库文件名的命名规范是以 lib 为前缀，紧接着跟静态库名，扩展名为.a

3-测试使用静态库：

gcc main.c -L. -lmyadd // -L 指定库的路径 -l 指定库名

执行./a.out

3.2 动态库的制作

1-我们用 gcc 来创建共享库

gcc -fPIC -c hello.c -o hello.o

-fPIC 创建与地址无关的编译程序

gcc -shared -o libmyhello.so hello.o

2-测试动态库使用

gcc main.c -L. -lmyhello

可以正常编译通过，但是运行时报错./a.out: error while loading shared libraries: libmyadd.so: cannot open shared object file: No such file or directory

原因：当加载动态库时，系统会默认从/lib 或/usr/lib 路径下查找库文件

解决方法（有三种）:

(1)把库拷贝到/usr/lib 和/lib 目录下。(此方法编译时不需要指定库的路径)

(2)在 LD_LIBRARY_PATH 环境变量中加上库所在路径。

export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:.

（终端关闭，环境变量就没在了）

(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

sudo vi xx.conf

添加动态库存在的路径,如：

/home/hq/teach/22092/day3/dynamic

补充：

头文件：

放在当前目录：#include "xx.h"，从当前路径下查找文件，如果没有再从系统目录下查找

放在系统目录：#include <xx.h>，默认从系统路径下查找，系统路径：/usr/include

放在其他目录：#include "xx.h"，在用 gcc 编译代码时加选项-I(i 的大写)指定头文件的路径

gcc main.c -I 头文件路径

库文件：动态库放在系统目录

系统路径：/usr/lib 和 /lib

gcc 编译时需要添加选项

-L 路径：指定库的路径

-l 库名：(小写的 L)指定库名

-I 路径：(大写的 i)指定头文件的路径

进程

1. 概念：

1.1 程序和进程区别：

程序：编译好的可执行文件

存放在磁盘上的指令和数据的有序集合（文件）

程序是静态的，没有任何执行的概念

进程：一个独立的可调度的任务

执行一个程序所分配的资源的总称

进程是程序的一次执行过程

进程是动态的，包括创建、调度、执行和消亡

1.2 特点

1. 系统会为每个进程分配 0-4g 的虚拟空间，其中 0-3g 是用户空间，每个进程独有；3g-4g 是内核空间，所有进程共享
2. 轮转调度：时间片，系统为每个进程分配时间片(几毫秒~几十毫秒),当一个进程时间片用完时，CPU 调度另一个进程，从而实现进程调度的切换

1.3. 进程段：

Linux 中的进程包含三个段：

“数据段”存放的是全局变量、常数以及动态数据分配的数据空间(如 malloc 函数取得的空间)等。

“正文段”存放的是程序中的代码

“堆栈段”存放的是函数的返回地址、函数的参数以及程序中的局部变量

1.4. 进程分类：

交互进程：该类进程是由 shell 控制和运行的。交互进程既可以在前台运行，也可以在后台运行。该类进程经常与用户进行交互，需要等待用户的输入，当接收到用户的输入后，该类进程会立刻响应，典型的交互式进程有：shell 命令进程、文本编辑器等

批处理进程：该类进程不属于某个终端，它被提交到一个队列中以便顺序执行。

守护进程：该类进程在后台运行。它一般在 Linux 启动时开始执行，系统关闭时才结束

1.5. 进程状态：

1）运行态（TASK_RUNNING）：R

指正在被 CPU 运行或者就绪的状态。这样的进程被成为 runnning 进程。

2）睡眠态(等待态)：

可中断睡眠态（TASK_INTERRUPTIBLE）S：处于等待状态中的进程，一旦被该进程等待的资源被释放，那么该进程就会进入运行状态。

不可中断睡眠态（TASK_UNINTERRUPTIBLE）D：该状态的进程只能用 wake_up()函数唤醒。

3）暂停态（TASK_STOPPED）:T

当进程收到信号 SIGSTOP、SIGTSTP、SIGTTIN 或 SIGTTOU 时就会进入暂停状态。可向其发送 SIGCONT 信号让进程转换到可运行状态。

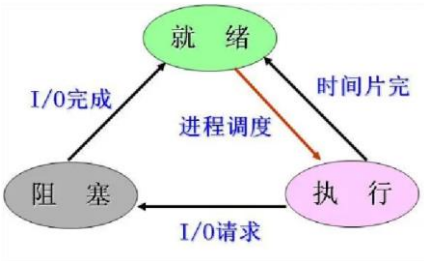
4）死亡态：进程结束 X

5）僵尸态：Z 当进程已经终止运行，但还占用系统资源，要避免僵尸态的产生

< 高优先级
N 低优先级
s 会话组组长
l 多线程
+ 前台进程

1.6. 进程状态切换图

进程创建后，进程进入就绪态，当 CPU 调度到此进程时进入运行态，当时间片用完时，此进程会进入就绪态，如果此进程正在执行一些 IO 操作(阻塞操作)会进入阻塞态，完成 IO 操作（阻塞结束）后又可进入就绪态，等待 CPU 的调度，当进程运行结束即进入结束态。



2. 函数：

2.1 创建进程

```
pid_t fork(void);
功能：创建子进程
返回值：
    成功：在父进程中：返回子进程的进程号 >0
          在子进程中：返回值为 0
    失败：-1 并设置 errno
```

特点：

1）子进程几乎拷贝了父进程的全部内容。包括代码、数据、系统数据段中的 pc 值、栈中的数据、父进程中打开的文件等；但它们的 PID、PPID 是不同的。

2）父子进程有独立的地址空间，互不影响；当在相应的进程中改变全局变量、静态变量，都互不影响。

3）若父进程先结束，子进程成为孤儿进程，被 init 进程收养，子进程变成后台进程。

4）若子进程先结束，父进程如果没有及时回收，子进程变成僵尸进程（要避免僵尸进程产生）

2.2 回收进程资源

```
pid_t wait(int *status);
功能：回收子进程资源，阻塞函数，等待子进程退出后结束阻塞
参数：status：子进程退出状态，不接受子进程状态设为 NULL
返回值：成功：回收的子进程的进程号
                失败：-1
```

作业：

1. 梳理今天学习内容
2. 编程实现 ls -l 功能。

```
//获取用户名和组名
 //getpwuid();//将用户 ID 转换成用户名
//getgrgid();//将组 ID 转换成组名
//时间
 //st_mtime:最后一次修改时间,单位是秒
 //localtime();
```