学习目标

1. 掌握read/write/lseek函数的使用
2. 掌握stat/lstat函数的使用
3. 了解文件属性相关的函数使用
4. 了解目录操作相关的函数的使用
5. 掌握目录遍历相关函数的使用
6. 掌握dup、dup2函数的使用
7. 掌握fcntl函数的使用

01-文件IO

1. open/close
   - 函数原型:
     - int open(const char *pathname, int flags);
     - int open(const char *pathname, int flags, mode_t mode);
   - 参数:
     - flags
       - 必选项 O_RDONLY, O_WRONLY, O_RDWR
       - 可选项
         - 创建文件: O_CREAT
           - 创建文件时检测文件是否存在: O_EXCL
           - 如果文件存在, 返回-1
           - 必须与O_CREAT一起使用
         - 追加文件: O_APPEND
         - 文件截断: O_TRUNC
         - 设置非阻塞: O_NONBLOCK
     - mode -- 指定0777
       - 八进制数
       - 最终权限: mode & ~umaks
       - umask 0002
         - 000000010
         - ~
         - 111111101
         - 111111111

         &

         111111101

         775

2. read
   - 函数原型: ssize_t read(int fd, void *buf, size_t count);
     - 参数:

        - fd -- open的返回值
        - buf - 缓冲区，存放读取的数据
        - count -- 缓冲区的最大容量 sizeof(buf)
    - 返回值：
        - -1: 失败
        - >0: 读出的字节数
        - =0: 文件读完了

3. write
    - 函数原型：ssize_t write(int fd, const void *buf, size_t count);
        - 参数：
            - fd: 文件描述符，open 返回值
            - buf: 要往文件中写的数据
            - count: 有效数据的长度
        - 返回值：
            - -1: 失败
            - >0: 写入的字节数

4. lseek
    - 函数原型：off_t lseek(int fd, off_t offset, int whence);
        - SEEK_SET
        - SEEK_CUR
        - SEEK_END
    - 使用：

    a. 文件指针移动到头部：

    lseek(fd, 0, SEEK_SET);

    b. 获取文件指针当前的位置：

    int len = lseek(fd, 0, SEEK_CUR);

    c. 获取文件长度：

    int len = lseek(fd, 0, SEEK_END);

        - 文件拓展
            - 文件原大小100k， 拓展为1100k

            lseek(fd, 1000, SEE_END);

            - 最后做一次写操作

```
write(fd, "a", 1);
```

5. 阻塞和非阻塞
    - 阻塞和非阻塞是文件的属性还是read函数的属性?
        - 文件的属性
        - 普通文件：hello.c
            - 默认不阻塞
        - 终端设备：/dev/tty
            - 默认阻塞
            - 管道
            - 套接字

02-stat/lstat函数

1. 获取文件属性

- struct stat {
  dev_t st_dev; //文件的设备编号
  ino_t st_ino; //节点
  mode_t st_mode; //文件的类型和存取的权限
  nlink_t st_nlink; //连到该文件的硬连接数目，刚建立的文件值为1
  uid_t st_uid; //用户ID
  gid_t st_gid; //组ID
  dev_t st_rdev; //(设备类型)若此文件为设备文件，则为其设备编号
  off_t st_size; //文件字节数(文件大小)
  blksize_t st_blksize; //块大小(文件系统的I/O 缓冲区大小)
  blkcnt_t st_blocks; //块数
  time_t st_atime; //最后一次访问时间
  time_t st_mtime; //最后一次修改时间
  time_t st_ctime; //最后一次改变时间(指属性)
  };
- st_mode -- 16位整数
  - 0-2 bit -- 其他人权限
    - S_IROTH 00004 读权限
    - S_IWOTH 00002 写权限
    - S_IXOTH 00001 执行权限
    - S_IRWXO 00007 掩码，过滤st_mode中除其他人权限以外的信息
  - 3-5 bit -- 所属组权限
    - S_IRGRP 00040 读权限
    - S_IWGRP 00020 写权限
    - S_IXGRP 00010 执行权限
    - S_IRWXG 00070 掩码，过滤st_mode中除所属组权限以外的信息
  - 6-8 bit -- 文件所有者权限
    - S_IRUSR 00400 读权限
    - S_IWUSR 00200 写权限
    - S_IXUSR 00100 执行权限
    - S_IRWXU 00700 掩码，过滤st_mode中除文件所有者权限以外的信息
  - 12-15 bit -- 文件类型

        - S_IFSOCK 0140000 套接字
        - S_IFLNK 0120000 符号链接（软链接）
        - S_IFREG 0100000 普通文件
        - S_IFBLK 0060000 块设备
        - S_IFDIR 0040000 目录
        - S_IFCHR 0020000 字符设备
        - S_IFIFO 0010000 管道
        - S_IFMT 0170000 掩码,过滤 st_mode 中除文件类型以外的信息
          (st_mode & S_IFMT) == S_IFREG
    - int stat(const char *path, struct stat *buf);
    - int lstat(const char *path, struct stat *buf);
        - lstat读取的链接文件本身的属性
        - stat读取的是连接文件指向的文件的属性
            - 追踪，穿透

03-文件属性函数

1. 测试当前用户指定文件是否具有某种属性
    - 当前用户，使用哪个用户调用这个函数，这个用户就是当前用户
    - int access(const char *pathname, int mode);
        - 参数：
          pathname: 文件名
          mode: 4种权限
              R_OK -- 读
              W_OK -- 写
              X_OK -- 执行
              F_OK -- 文件是否存在
        - 返回值：
              0 - 有某种权限，或者文件存在
              -1 - 没有，或文件不存在
2. 修改文件权限
    - int chmod(const char *filename, int mode);
        - 参数：
            - filename: 文件名
            - mode: 文件权限，八进制数
3. 修改文件所有者和所属组
    - int chown(const char *path, uid_t owner, gid_t group);
    - 函数参数：
        - path -- 文件路径
        - owner -- 整形值，用户ID
            - /etc/passwd
        - group -- ....., 组ID

        - /etc/group

4. 修改文件大小
    - int truncate(const char *path, off_t length);
    - 参数:
        - path -- 文件名
        - length -- 文件的最终大小
            1. 比原来小, 删掉后边的部分
            2. 比原来大, 向后拓展

04-目录操作相关函数

1. 文件重命名
   - int rename(const char *oldpath, const char *newpath);
2. 修改当前进程(应用程序)的路径 cd
   - int chdir(const char *path);
     - 参数: 切换的路径
3. 获取当前进程的工作目录 pwd
   - char *getcwd(char *buf, size_t size);
     - 返回值:
       - 成功: 当前的工作目录
       - 失败: NULL
     - 参数:
       - buf: 缓冲区, 存储当前的工作目录
       - size: 缓冲区大小
4. 创建目录 mkdir
   - int mkdir(const char *pathname, mode_t mode);
     - 参数:
       - pathname: 创建的目录名
       - mode: 目录权限, 八进制的数, 实际权限: mode & ~umask
5. 删除一个空目录
   - int rmdir(const char *pathname);
     - 参数: 空目录的名字

05-目录遍历相关函数

1. 打开一个目录
   - DIR *opendir(const char *name);
     - 参数：目录名
     - 返回值：指向目录的指针
   - FILE* fp = fopen（）
   - fread（buf， len， len，fp）；
2. 读目录

```
struct dirent
{
   ino_t d_ino;                  // 此目录进入点的inode
   ff_t d_off;                   // 目录文件开头至此目录进入点的位移
   signed short int d_reclen; // d_name 的长度，不包含NULL 字符
   unsigned char d_type;     // d_name 所指的文件类型
   har d_name[256];          // 文件名
};
```

d_type
   - DT_BLK - 块设备
   - DT_CHR - 字符设备
   - DT_DIR - 目录
   - DT_LNK - 软连接
   - DT_FIFO - 管道
   - DT_REG - 普通文件
   - DT_SOCK - 套接字
   - DT_UNKNOWN - 未知

   - struct dirent *readdir(DIR *dirp);
     - 参数：opendir的返回值
     - 返回值：目录项结构体
3. 关闭目录
   - int closedir(DIR *dirp);
4. 独立完成递归读目录中指定类型文件个数的操作.

06-dup-dup2-fcntl

1. 复制文件描述符
   - int dup(int oldfd);
     - oldfd - 要复制的文件描述符
     - 返回值：新的文件描述符
     - dup调用成功：
       - 有两个文件描述符指向同一个文件
     - 返回值：取最小的且没被占用的文件描述符
   - int dup2(int oldfd, int newfd);
     - oldfd -》 hello
     - newfd -》 world
       - 假设newfd已经指向了一个文件，首先断开close与那个文件的链接，newfd指向oldfd指向的文件
         - 文件描述符重定向
         - oldfd和newfd指向同一个文件
       - newfd没有被占用，newfd指向oldfd指向的文件
2. 改变已经打开的文件的属性：fcntl
   - 变参函数
   - 复制一个已有的文件描述符

     int ret = fcntl(fd, F_DUPFD);
   - 获取/设置文件状态标志
     - open 的flags参数

     1. 获取文件状态标识

        int flag = fcntl(fd, F_GETFL)

     2. 设置文件状态标识
        - flag = flag | O_APPEND;
        - fcntl(fd, F_SETFL, flag)
   - 可以更改的几个标识：O_APPEND、O_NONBLOCK（常用）

作业

2016年11月28日 17:30

1. ls -l
    a. stat(lstat)
2. 验证dup和dup2
    a. 验证fcntl

复习

2017年3月18日 9:09

1. makefile
   - 一个规则

     目标：依赖

     （tab缩进）命令
     - 可以有多个规则
       - make - 终极目标
         - 第一个规则对应的目标
       - make 子目标的名字
     - 工作原理：
       - 依赖不存在：
         - 向下查找子规则，找到了用来生成依赖的规则，执行规则中的命令
       - 依赖存在->更新检查：
         - 更新条件：目标时间<依赖的时间
   - 两个函数：
     - wildcard
       - src = $(wildcard ./src/*.c)
       - src=./src/a.c ./src/b.c ./src/c.c
     - patsubst
       - obj=$(patsubst %.c, %.o, $(src))
       - obj=./src/a.o ./src/b.o ./src/c.o
   - 三个自动变量
     - $@: 规则中的目标
     - $<: 规则中的依赖（第一个）
     - $^: 规则中的所有的依赖
       - 只能在命令中使用

app： a.o b.o c.o

gcc a.o b.o c.o -o app

- 模式规则：

%.o:%.c

gcc -c $< -o $@

- 清空项目：
  - 声明伪目标：

.PHONY:clean

clean：

rm *.o

2017年3月18日 10:16

系统函数库

open
read
write
lseek

全局变量：errno

不同的值，对应不同的错误信息

perror（）；

2017年3月18日 10:56

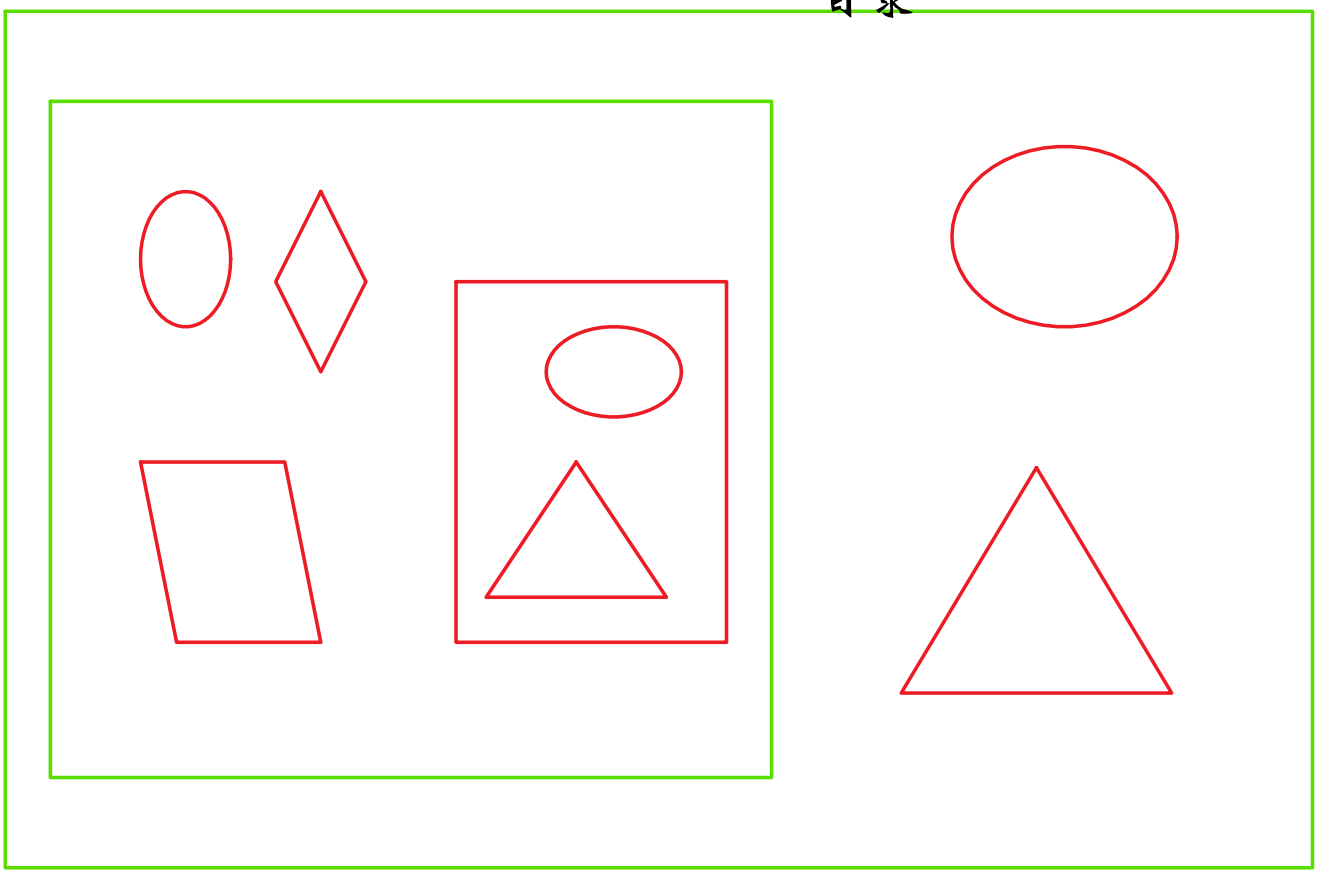
终端

默认bash是前台程序
./a.out - 启动了一个程序，前台程序变成了./a.out, bash变成了后台程序

./a.out等待用户输入10个字符
实际输入的个数>10, 剩下的还在缓冲区
read解除阻塞读缓冲区数据，write，程序结束

bash从后台程序变为前台程序，检测到了缓冲区数据，将缓冲区作为shell命令去做了解析

目录



进到目录之后：opendir，读内容

需要循环的读目录中的内容

```
while（readdir）
{
}
```

最后关闭目录

closedir（dir）；