学习目标

1. 掌握/open/read/write/lseek/close 函数的使用
2. 掌握 stat/lstat 函数的使用
3. 掌握目录遍历相关函数的使用
4. 掌握 dup、dup2 函数的使用
5. 掌握 fcntl 函数的使用

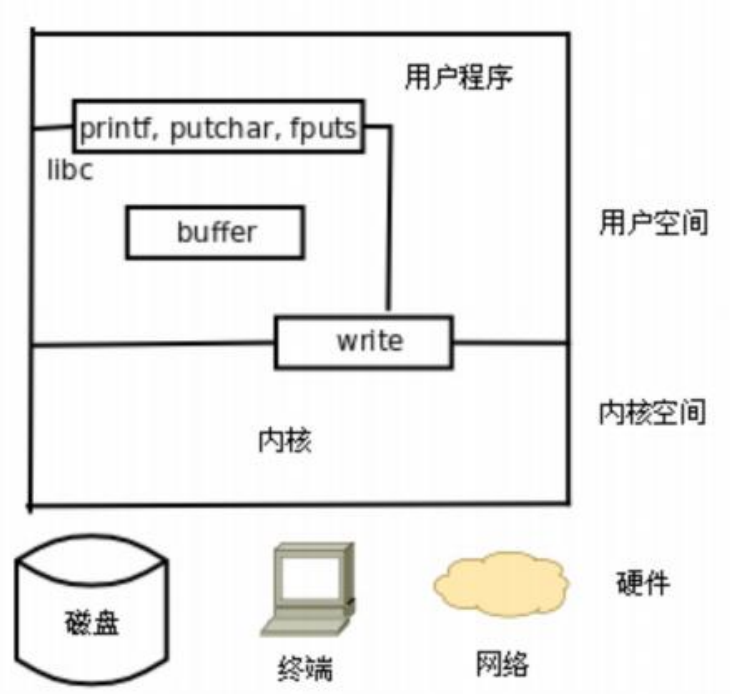
文件 IO

从本章开始学习各种 Linux 系统函数,这些函数的用法必须结合 Linux 内核的工作原理来理解,因为系统函数正是内核提供给应用程序的接口，而要理解内核的工作原理,必须熟练掌握 C 语言,因为内核也是用 C 语言写的，我们在描述内核工作原理时必然要用“指针”、“结构体”、“链表”这些名词来组织语言，就像只有掌握了英语才能看懂英文书一样，只有学好了 C 语言才能看懂我描述的内核工作原理。

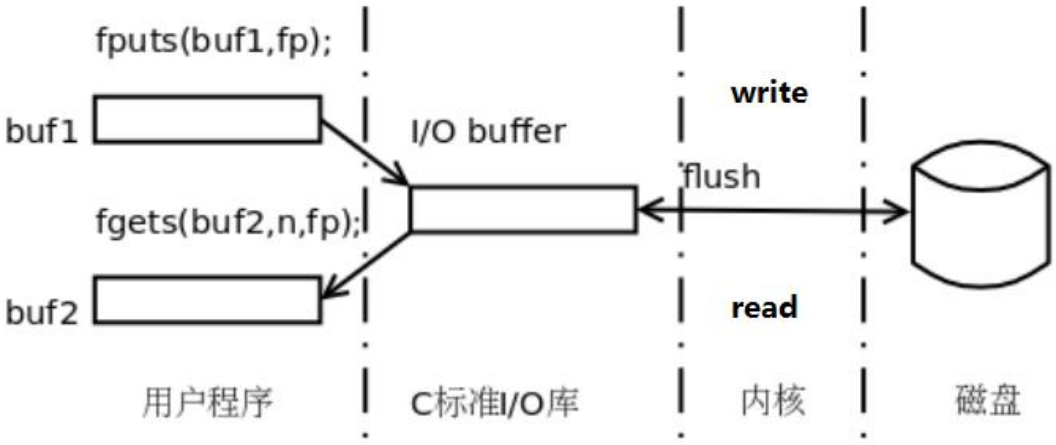
C 标准函数与系统函数的区别

什么是系统调用

由操作系统实现并提供给外部应用程序的编程接口。(Application Programming Interface，API)。是应用程序同系统之间数据交互的桥梁。
一个 helloworld 如何打印到屏幕。



每一个 FILE 文件流（标准 C 库函数）都有一个缓冲区 buffer，默认大小 8192Byte。Linux 系统的 IO 函数默认是没有缓冲区.



open/close

文件描述符

一个进程启动之后，默认打开三个文件描述符：

```
#define   STDIN_FILENO          0
#define   STDOUT_FILENO         1
#define   STDERR_FILENO         2
```

新打开文件返回文件描述符表中未使用的最小文件描述符，调用 open 函数可以打开或创建一个文件，得到一个文件描述符.

open 函数

- 函数描述：打开或者新建一个文件
- 函数原型：

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

  int open(const char *pathname, int flags, mode_t mode);
- 函数参数：
  - pathname 参数是要打开或创建的文件名,和 fopen 一样, pathname 既可以是相对路径也可以是绝对路径。
  - flags 参数有一系列常数值可供选择，可以同时选择多个常数用按位或运算符连接起来，所以这些常数的宏定义都以 O_开头,表示 or。
    - 必选项:以下三个常数中必须指定一个，且仅允许指定一个。
      - O_RDONLY 只读打开
      - O_WRONLY 只写打开
      - O_RDWR 可读可写打开
    - 以下可选项可以同时指定 0 个或多个，和必选项按位或起来作为 flags 参数。可选项有很多，这里只介绍几个常用选项：
      - O_APPEND 表示追加。如果文件已有内容，这次打开文件所写的数据附加到文件的末尾而不覆盖原来的内容。
      - O_CREAT 若此文件不存在则创建它。使用此选项时需要提供第三个参数 mode，表示该文件的访问权限。
        - ★ **文件最终权限：mode & ~umask**
      - O_EXCL 如果同时指定了 O_CREAT,并且文件已存在,则出错返回。
      - O_TRUNC 如果文件已存在，将其长度截断为为 0 字节。
      - O_NONBLOCK 对于设备文件，以 O_NONBLOCK 方式打开可以做非阻塞 I/O(NonblockI/O),非阻塞 I/O。
- **函数返回值:**
  - 成功：返回一个最小且未被占用的文件描述符
  - 失败：返回-1，并设置 errno 值.

close 函数

- 函数描述：关闭文件
- 函数原型： int close(int fd);
- 函数参数： fd 文件描述符
- 函数返回值:
  - 成功返回 0
  - 失败返回-1，并设置 errno 值.

需要说明的是,当一个进程终止时，内核对该进程所有尚未关闭的文件描述符调用 close 关闭，所以即使用户程序不调用 close，在终止时内核也会自动关闭它打开的所有文件。但是对于一个长年累月运行的程序(比如网络服务器)，打开的文件描述符一定要记得关闭，否则随着打开的文件越来越多，会占用大量文件描述符和系统资源。

read/write

read 函数

- 函数描述：从打开的设备或文件中读取数据
- 函数原型：ssize_t read(int fd, void *buf, size_t count);
- 函数参数：
  - fd：文件描述符
  - buf：读上来的数据保存在缓冲区 buf 中
  - count: buf 缓冲区存放的最大字节数
- 函数返回值：
  - >0：读取到的字节数
  - =0：文件读取完毕
  - -1： 出错，并设置 errno

write

- 函数描述：向打开的设备或文件中写数据
- 函数原型：ssize_t write(int fd, const void *buf, size_t count);
- 函数参数：
  - fd：文件描述符
  - buf：缓冲区，要写入文件或设备的数据
  - count：buf 中数据的长度
- 函数返回值：
  - 成功：返回写入的字节数
  - 错误：返回-1 并设置 errno

lseek

所有打开的文件都有一个当前文件偏移量(current file offset),以下简称为 cfo. cfo 通常是一个非负整数，用于表明文件开始处到文件当前位置的字节数. 读写操作通常开始于 cfo，并且使 cfo 增大，增量为读写的字节数. 文件被打开时，cfo 会被初始化为 0，除非使用了 O_APPEND.

使用 lseek 函数可以改变文件的 cfo.

```
#include <sys/types.h>
#include <unistd.h>
off_t lseek(int fd, off_t offset, int whence);
```

- 函数描述：移动文件指针
- 函数原型：off_t lseek(int fd, off_t offset, int whence);
- 函数参数：
  - fd：文件描述符

- 参数 offset 的含义取决于参数 whence：
  - 如果 whence 是 SEEK_SET，文件偏移量将设置为 offset。
  - 如果 whence 是 SEEK_CUR，文件偏移量将被设置为 cfo 加上 offset，offset 可以为正也可以为负。
  - 如果 whence 是 SEEK_END，文件偏移量将被设置为文件长度加上 offset，offset 可以为正也可以为负。

- 函数返回值：若 lseek 成功执行，则返回新的偏移量。
- lseek 函数常用操作
  - 文件指针移动到头部

    lseek(fd, 0, SEEK_SET);
  - 获取文件指针当前位置

    int len = lseek(fd, 0, SEEK_CUR);
  - 获取文件长度

    int len = lseek(fd, 0, SEEK_END);
  - lseek 实现文件拓展

```
off_t currpos;
// 从文件尾部开始向后拓展 1000 个字节
currpos = lseek(fd, 1000, SEEK_END);
// 额外执行一次写操作，否则文件无法完成拓展
write(fd, “a”, 1);    // 数据随便写
```

练习：

1 编写简单的 IO 函数读写文件的代码

2 使用 lseek 函数获取文件大小

3 使用 lseek 函数实现文件拓展

perror 和 errno

errno 是一个全局变量，当系统调用后若出错会将 errno 进行设置，perror 可以将 errno 对应的描述信息打印出来.

如:perror("open"); 如果报错的话打印: open:(空格)错误信息

练习:编写简单的例子，测试 perror 和 errno.

阻塞和非阻塞:

思考：阻塞和非阻塞是文件的属性还是 read 函数的属性?

- 普通文件：hello.c
  - 默认是非阻塞的
- 终端设备：如 /dev/tty
  - 默认阻塞
- 管道和套接字
  - 默认阻塞

练习：

1 测试普通文件是阻塞还是非阻塞的?

2 测试终端设备文件/dev/tty 是阻塞还是非阻塞的.

得出结论：阻塞和非阻塞是文件本身的属性，不是 read 函数的属性.

文件和目录

文件操作相关函数

stat/lstat 函数

- 函数描述：获取文件属性
- 函数原型: int stat(const char *pathname, struct stat *buf);
  int lstat(const char *pathname, struct stat *buf);
- 函数返回值：
  - 成功返回 0
  - 失败返回 -1

```
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

If (st_mode & S_IRUSR) -----为真表明可读

If (st_mode & S_IWUSR) ------为真表明可写

If (st_mode & S_IXUSR) ------为真表明可执行

- 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

If ((st_mode & S_IFMT)==S_IFREG) ----为真普通文件

if(S_ISREG(st_mode)) ------为真表示普通文件

if(S_ISDIR(st.st_mode)) ------为真表示目录文件

stat 函数和 lstat 函数的区别

- 对于普通文件，这两个函数没有区别，是一样的.
- 对于连接文件,调用 lstat 函数获取的是链接文件本身的属性信息;
  而 stat 函数获取的是链接文件指向的文件的属性信息.

练习:

1 stat 函数获取文件大小

2 stat 函数获取文件类型和文件权限

3 lstat 函数获取连接文件的属性(文件大小)

目录操作相关函数

opendir 函数

- 函数描述:打开一个目录
- 函数原型: DIR *opendir(const char *name);
- 函数返回值: 指向目录的指针
- 函数参数: 要遍历的目录(相对路径或者绝对路径)

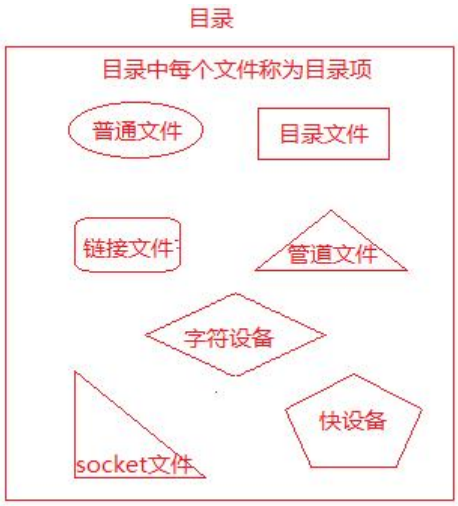
readdir 函数

- 函数描述：读取目录内容--目录项
- 函数原型：struct dirent *readdir(DIR *dirp);
- 函数返回值：读取的目录项指针
- 函数参数：opendir 函数的返回值

```
struct dirent
{
    ino_t d_ino;                // 此目录进入点的 inode
    off_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
    char d_name[256];           // 文件名
};
```

d_type 的取值：

- 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



closedir 函数

- 函数描述：关闭目录
- 函数原型：int closedir(DIR *dirp);
- 函数返回值：成功返回 0，失败返回-1

- 函数参数: opendir 函数的返回值

读取目录内容的一般步骤

```
1 DIR *pDir = opendir("dir");    //打开目录
2 while((p=readdir(pDir))!=NULL){}  //循环读取文件
3 closedir(pDir);  //关闭目录
```

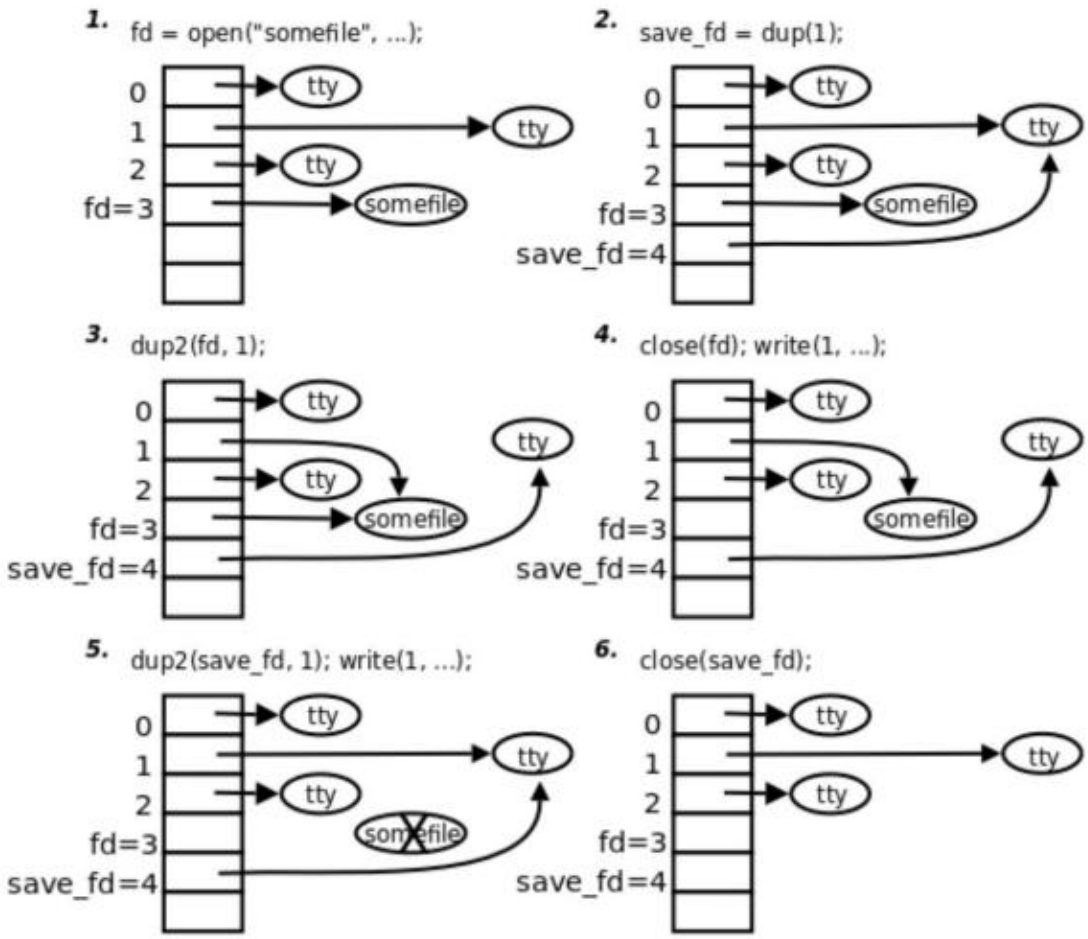
练习

1 遍历指定目录下的所有文件，并判断文件类型.

2 递归遍历目录下所有的文件，并判断文件类型.

特别注意: 递归遍历指定目录下的所有文件的时候，要过滤掉.和..文件，否则会进入死循环

dup/dup2/fcntl



dup 函数

- 函数描述：复制文件描述符
- 函数原型：int dup(int oldfd);
- 函数参数：oldfd -要复制的文件描述符
- 函数返回值：
  - 成功：返回最小且没被占用的文件描述符
  - 失败：返回-1，设置 errno 值

练习：编写程序，测试 dup 函数.

dup2 函数

- 函数描述：复制文件描述符
- 函数原型：int dup2(int oldfd, int newfd);
- 函数参数：
  - oldfd-原来的文件描述符
  - newfd-复制成的新的文件描述符
- 函数返回值：
  - 成功：将 oldfd 复制给 newfd，两个文件描述符指向同一个文件
  - 失败：返回-1，设置 errno 值
- 假设 newfd 已经指向了一个文件，首先 close 原来打开的文件，然后 newfd 指向 oldfd 指向的文件.

  若 newfd 没有被占用，newfd 指向 oldfd 指向的文件.

练习：

1 编写程序，测试 dup2 函数实现文件描述符的复制.

2 编写程序，完成终端标准输出重定向到文件中

fcntl 函数

- 函数描述：改变已经打开的文件的属性
- 函数原型：int fcntl(int fd, int cmd, ... /* arg */);
  - 若 cmd 为 F_DUPFD，复制文件描述符，与 dup 相同
  - 若 cmd 为 F_GETFL，获取文件描述符的 flag 属性值
  - 若 cmd 为 F_SETFL，设置文件描述符的 flag 属性
- 函数返回值:返回值取决于 cmd
  - 成功
    - 若 cmd 为 F_DUPFD，返回一个新的文件描述符
    - 若 cmd 为 F_GETFL，返回文件描述符的 flags 值
    - 若 cmd 为 F_SETFL，返回 0
  - 失败返回-1，并设置 errno 值.

- fcntl 函数常用的操作:

  1 复制一个新的文件描述符:

  int newfd = fcntl(fd, F_DUPFD, 0);

  2 获取文件的属性标志

  int flag = fcntl(fd, F_GETFL, 0)

  3 设置文件状态标志

  flag = flag | O_APPEND;

  fcntl(fd, F_SETFL, flag)

  4 常用的属性标志

  O_APPEND-----设置文件打开为末尾添加

  O_NONBLOCK-----设置打开的文件描述符为非阻塞

练习:

1 使用 fcntl 函数实现复制文件描述符

2 使用 fcntl 函数设置在打开的文件末尾添加内容.