Chapter 1

文件系统（File System）

- 操作系统中负责存取和管理文件的部分
- 一个文件及其某些属性的集合。它为这些文件的序列号提供了一个名称空间
- 类型
  - VFS，（Virtual File System）虚拟文件系统，与以下的磁盘文件系统（即文件的分区格式不同），为底层的文件系统提供了统一的抽象
  - EXT2，EXT3，EXT4，FAT32，ExFAT

Boot Loader

- Boot Loader加载和启动操作系统内核（注意这是个不特定与具体操作系统的概念）。Linux下使用的Boot Loader是GRUB

虚拟终端（Virtual Terminal）

- Linux下6个虚拟终端

命令提示符

- 可以自行配置
- $：当前登录身份是普通用户
- #: root用户

文件和目录

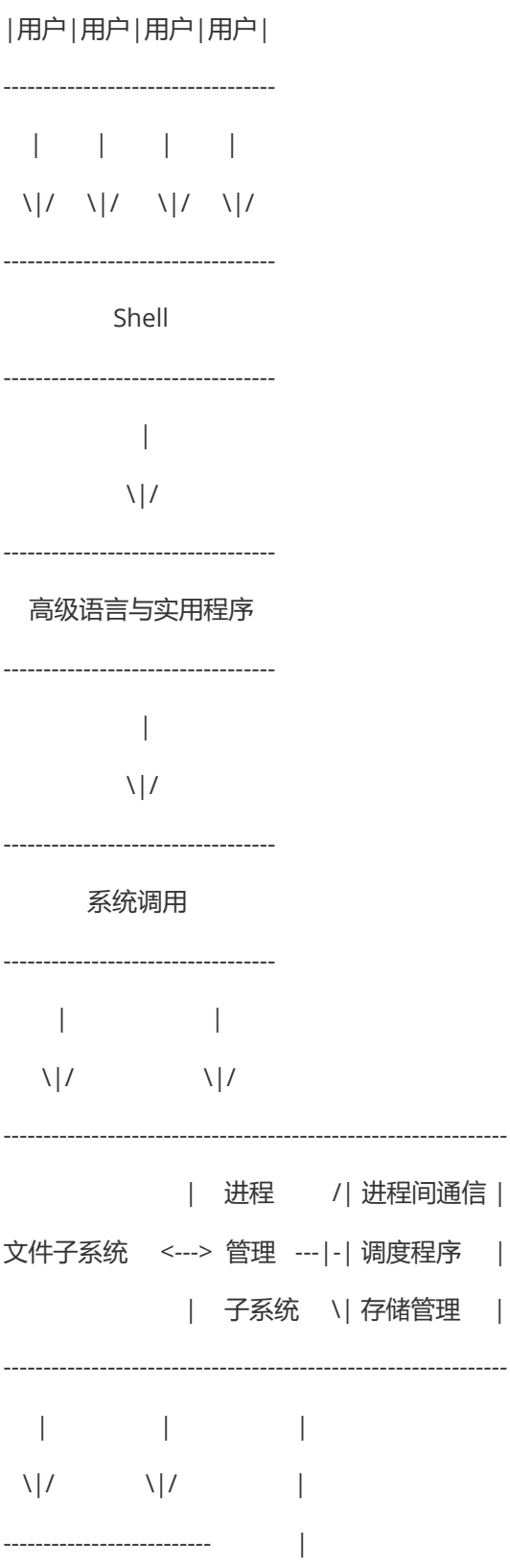
- 文件
  - 文件是数据的集合
  - 文件结构：字节流、记录序列（Record Sequence）、记录树（Record Tree）。Linux下为字节流。
  - Linux的文件类型
    - 普通文件（-）：纯文本文件、二进制文件、数据格式文件
    - 字符设备文件（c, character）：与设备进行交互的文件。按字符进行I/O。如：终端文件（ttyN）
    - 块设备文件（b, block）：同字符设备文件，但按块进行I/O。如：硬盘
    - 套接字文件（s, socket）：表示一个socket连接。可以通过这个文件来与连接的对方（peer）进行通信。
    - 管道文件/FIFO文件（p, pipe)：用于进程间通信。一个进程可以从这个文件中读取另一个进程此前写入的数据
    - 符号链接文件（l, link）
    - 目录文件（d, directory）
  - 目录结构
    - Linux中所有的目录均包含在一个统一的、虚拟的统一文件系统（Unified File System）中。
    - 物理设备被抽象为文件，挂载到指定的挂载点。没有Windows下的盘符的概念
    - 根目录下各个文件夹的作用：
      - /bin：系统必要的命令的二进制文件。包含了会被会被系统管理者和用户使用的命令。大部分常用的命令都在这里。
      - /boot：Boot Loader相关的静态文件。包含了所有需要在系统引导阶段使用的文件（如内核镜像等）。
      - /dev：设备对应的虚拟文件。
      - /etc：系统和软件的配置文件。
      - /lib：必要的共享库文件（如.so）或内核模块。
      - /media：外部设备通用挂载点的父目录。
      - /mnt：临时文件系统的挂载点的父目录。
      - /opt：额外的应用软件包安装目录
      - /sbin：只有管理员可以使用的命令的二进制文件。是与系统相关的基本命令，如shutdown，reboot等
      - /srv：系统提供的有关服务的数据
      - /tmp：临时文件
      - /usr：Unix System Resources，不是user的简写。用于存放共享、只读的数据。子目录包括/bin，/etc，/lib，/sbin，/tmp等，与根目录下对应的目录对比，这些目录是給后来安装的软件的使用的（而不是系统自带的）。还有/include，/src等文件夹，存放系统编程所需的头文件和源码等。
      - /home：用户的家目录的父目录
      - /root：root用户的家目录

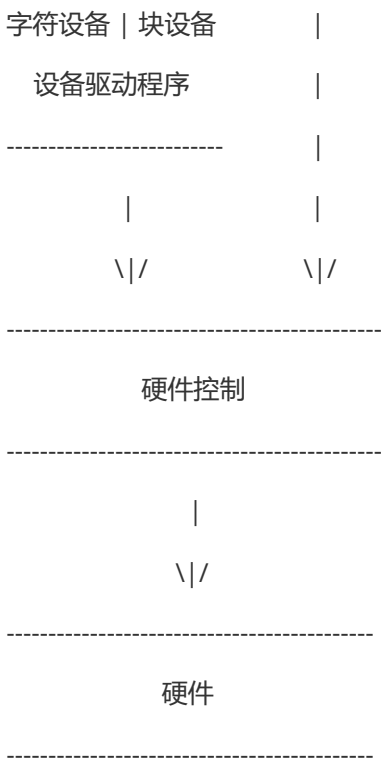
- 文件权限
  - 分为三个层次：所有者，所有者所在组，其他用户
  - 每个层次又分三个类型：读、写、执行
  - 更改权限：参见chmod

进程

进程是一个正在执行的程序实例。由程序体，当前值（？），状态信息，以及操作系统用于管理此进程执行情况的资源组成

Unix的系统结构





重定向与管道

重定向

- 标准输入、标准输出、标准错误
  - 文件描述符：0,1,2
  - C库（stdio.h）的流指针：stdin，stdout，stderr
- 命令行操作符
  - >：将程序的输出重定向到一个文件设备文件，覆盖原来的文件
  - >!：同上，但是强制覆盖
  - >>：同上，但是不覆盖而是在末尾追加
  - <：将程序的输入重定向为某个程序

管道

- |：将一个进程的标准输出作为另一个进程的标准输入

环境变量

- 操作环境的参数
- 相关命令
  - env：显示所有环境变量
  - echo $VAR_NAME：显示某个变量的值
  - set：显示本地定义的变量
  - export：将用户变量（只在当前shell可见变量）输出为环境变量
- 常用环境变量
  - HOME：当前用户的家目录
  - PATH：可执行文件的搜索路径
  - TERM：终端的类型

- UID：当前用户的UID
- PWD：当前工作目录
- PS1：主提示符

Chapter 2

执行脚本的方式

1. `sh/bash/csh/some_bash script_file`
2. `chmod +x script_file`

   `./scirpt_file`
3. `source script_file`或`. script_file`

用户变量

- 在Shell脚本中定义的变量，运行时只会在当前Shell可见
- 赋值：`var_name=value`（不加$，不能有空格）
- 使用：`$var_name`或`${var_name}`
- 删除：`unset var_name`（不加$）
- read：
  - read var_name
  - 带提示：read -p "prompt_here" var_name

引号

- 单引号内所有字符都是字面量（不会被转义或求值）
- 双引号内$、``（反引号）会被求值，\会被转义

参数变量和内部变量

调用脚本时，如果带有参数，会产生一些额外的变量

- $#：参数个数
- $0：脚本程序名
- $1，$2...：按调用时指定的顺序排列的参数
- $*：全部参数连接成的一个字符串，以环境变量$IFS的第一个字符分隔
- $@：全部参数组成的列表（就是数据结构的那个列表）

条件测试

- `test expression`或`[ expression ]`（注意方括号内部两侧都必须要有空格）。这两种方式是一个命令调用，通过命令的退出值表明真伪。0表示真，1表示假。使用举例：

```
$ test 1 -lt 2
$ echo $?
0
$ test 1 -gt 2
$ echo $?
1
```

比较运算（注意关系算符和操作数之间都要有空格）：

- 字符串比较：
    1. str1 = str2
    2. str1 != str2
    3. -z str：字符串是否为空
    4. -n str：字符串是否不为空
- 算术比较：expr1 *op* expr2，下面列举的是op
    1. -eq：相等
    2. -ne：不等
    3. -gt：大于
    4. -ge：大于等于
    5. -lt：小于
    6. -le：小于等于
- 文件测试：op file，下面列举的是op
    1. -e file：是否存在
    2. -d file：是否是目录
    3. -f file：是否是常规文件
    4. -s file：文件长度是否不为0
    5. -r file：是否可读
    6. -w file：是否可写
    7. -x file：是否可执行
- 逻辑操作：
    1. ! expr：逻辑取反
    2. expr1 -a expr2：逻辑且
    3. expr1 -o expr2：逻辑或

- 算术拓展：`$((expression))`。这是一个求值表达式，可以赋给变量。0表示假，1表示真。（算术拓展详见下文）

算术运算

算术拓展

- 使用举例：

```
$ a=$((0 < 1))
$ echo $a
1
$ echo $((0 >= 1))
0
```

  算术拓展也完全可以用于算术运算

```
$ a=$((1 + 2))
$ echo $a
3
$ echo $((9 * 8))
72
$ echo $((9 ** 2))
81
$ unset a
$ echo $((10 + a))
10
$ echo $((10 + $a))
syntax error
```

另外还有`(())`的形式（即不带`$`）。表示对变量自己做运算，如

```
$ unset a
$ ((a = 99))
$ echo $a
99
$ ((a++))
$ echo $a
100
$ ((a /= 10))
$ echo $a
10
$ ((a = a > 8 ? a - 1 : a))
$ echo $a
9
```

let指令、$[]、expr指令

和算术拓展基本一样

参数拓展

1. `${var_name=default}`，如果var_name不存在，则返回default的值，并把var_name赋值为default
2. `${var_name:=default}`，如果var_name不存在或者值为空，则返回default的值并把var_name赋值为default
3. `${var_name-default}`，如果var_name不存在，则返回default的值
4. `${var_name:-default}`，如果var_name不存在或者值为空，则返回default的值
5. `${var_name+default}`，如果var_name存在，则返回default的值
6. `${var_name:+default}`，如果var_name存在且不为空，则返回default的值
7. `${var_name?default}`，如果var_name不存在，则报错并输出default的值
8. `${var_name:?default}`，如果var_name不存在或者值为空，则报错并default的值
9. `${#var_name}`，var_name的值的长度
10. `${var_name#regex}`，从左侧开始，去掉var_name的之中匹配regex的部分（非贪婪模式）
11. `${var_name##regex}`，从左侧开始，去掉var_name的之中匹配regex的部分（贪婪模式）
12. `${var_name%regex}`，从右侧开始，去掉var_name的之中匹配regex的部分（非贪婪模式）

13. `${var_name##regex}`，从右侧开始，去掉var_name的之中匹配regex的部分（贪婪模式）
14. `${var_name:x:y}`，提取var_name的值
    1. 如果x
        1. 是非负数：从左侧开始第x个字节开始
        2. 是负数：
    2. 如果y
        1. 是非负数：的连续y个字节的内容
        2. 是负数：到从右侧开始数y个字节的内容
15. `${var_name/regex/sub}`，替换var_name中的内容，语法和sed及vim的替换指令一样，只替换一次
16. `${var_name/regex/sub}`，替换var_name中的内容，语法和sed及vim的替换指令一样，替换全部

替换

``和$()中的命令会优先在子shell中执行，并用其标准输出的内容替换原命令中的部分

Shell脚本

if语句

```
if [ expr ]
then
  stmts
elif [ expr ]
then
  stmts
else
  stmts
fi
```

then可以写在同一行，但需要用分号分割，即`if [ expr ]; then`

for语句

```
for var in list
do
  stmts
done
```

也可以写成c风格（其他语句同理）

```
for ((i = 1; i < 5; i++))
do
  stmts
done
```

while语句

```
while condition
do
  stmts
done
```

until语句

```
until condition
do
  stmts
done
```

select语句

```
select item in itemlist
do
  stmts
done
```

这个的具体用法请仔细搜索

case语句

```
case val in
  v1) stmts;;
  v2 | v3) stmts;;
  *) stmts::
esac
```

函数

声明:

```
func_name() {
  stmts
  [return]
}
```

调用

`func_name [params]`

函数体中类似脚本变量$1，$2的方式获取参数

用 $?获取返回值

其他

break，continue：同其他语言

exit n：以退出码n退出执行

return：函数返回

trap：收到信号时的动作

`:`（冒号命令）：空命令

`.`（点号命令）或source：在当前shell中执行命令（即不开子shell）

Chapter 3-0

gcc

用于编译、连接

- -E：只进行预处理

- -S：只进行预处理、编译
- -c：预处理、编译、汇编
- -o [output_file]：指定输出文件名
- -g：产生符号文件（可用于调试工具）
- -On：优化等价，n可以取0,1,2,3
- -Wall：显示所有警告信息
- -Idir：指定额外的头文件搜索路径
- -Ldir：指定额外的库文件搜索路径
- -lname：指定链接时搜索的库文件（name要去掉lib，如要链接pthread，pthread的静态库名为libpthread.a，则参数写成-lpthread）
- -DMACROT[=DEFN]：定义宏

make

不会重点考，稍微了解，自己看课件或者博客

编译链接过程

源文件(.c/.cpp)--预处理-->纯C代码--编译(cc)-->汇编程序(.s)--汇编(as)-->目标文件(.o)--链接(ld)-->可执行文件

Chapter 3-1

虚拟文件系统

- 只存在于内存中（这句话不完全准确。如有兴趣请深入理解一遍Linux的文件系统）
- 四种对象
  - super block：超级块。一个超级块对应一个文件系统。
  - inode：索引节点。一个实际存在的文件实体只有一个inode。inode对象全系统共用。
  - dentry：目录项。一个目录项对应一个dentry，就是ls -a列出来的每一项就是一个dentry。dentry中有指向inode的指针。多个dentry可以对应同一个inode。dentry对象全系统共用。
  - file：文件对象。一个打开了的文件对应一个file。file中有指向dentry的指针。文件对象是进程私有的（会以copy-on-write的方式与子进程共享）。

上课讲的文件系统的内容很少而且很。文件系统又是很复杂的一个子系统，三言两语难以说清。要理解的话得自己多查资料。

- 软连接（符号链接）和硬连接
  - 软连接
    - 是确确实实存在的一个文件，有自己的inode号
    - 文件中存放被链接的文件的路径
    - 可以跨越文件系统
    - 对应系统调用symlink
  - 硬链接
    - 与被链接的文件共享同一个inode，dentry不同
    - 不能跨越文件系统
    - 对应系统调用link

系统调用与库函数

- 都以C函数的形式出现
- 系统调用：是Linux内核与用户层程序交互的唯一接口。提供最小的接口，需要陷入内核态运行。不可移植。
- 库函数：依赖系统调用，是对系统调用的封装和组合，提供较为复杂的功能。可移植。

IO系统调用

IO系统调用围绕文件描述符fd，一个非负整数进行。标准输入、标准输出、标准错误对应的fd分别是STDIN_FILENO(0)，STDOUT_FILENO(1)，STDERR_FILENO(2)

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

  `int open(const char *pathname, int flags, mode_t mode);`

  `int creat(const char *pathname, mode_t mode)`

  pathname：文件路径

  flags：文件打开模式。位域。可选值O_RDONLY、O_WRONLY、O_RDWR、O_APPEND、O_TRUNC（清空文件原来的内容）、O_CREAT（如果不存在则创建）、O_EXCL（和O_CREAT一起使用时，如果原来存在则报错）、O_NONBLOCK（非阻塞模式）

  mode：创建文件时的权限，无符号整数，同chmod的值

  返回值：文件描述符；失败时则-1

- `int close(int fd)`

  fd：文件描述符

  返回值：0；失败则-1

- `ssize_t read(int fd, void *buf, size_t count);`

  buf：缓冲区

  size_t：要读取的字节数

  返回值：已读取的字节数；若此次调用前已达到文件末尾，则0；出错则-1

- `ssize_t write(int fd, const void *buf, size_t count);`

  类比read

- `off_t lseek(int fd, off_t offset, int whence)`

  offset：偏移量

whence：SEEK_SET：相对文件头偏移+offset处(这里offset不可以为负值)

SEEK_CUR：相对当前位置偏移+offset处（可以为负值）

SEEK_END：偏移到文件末尾+offset处（可以为负值）

返回值：偏移量；失败则-1

- `int dup(int oldfd);`

  `int dup2(int oldfd, int newfd);`

  dup复制一个文件文件描述符，返回新的

  dup2复制oldfd到newfd，之前newfd对应的文件将被关闭。

  返回：新的文件描述符；出错则-1

- `int fcntl(int fd, int cmd);`

  `int fcntl(int fd, int cmd, long arg);`

  `int fcntl(int fd, int cmd, struct flock *lock);`

  cmd:

  - F_DUPFD：复制文件描述符，返回新的文件描述符
  - F_GETFD/F_SETFD：获取/设置文件描述符标识（目前只有close-on-exec，表示子进程在执行exec族命令时释放对应的文件描述符）。
  - F_GETFL/F_SETFL：获得/设置文件状态标识（open/creat中的flags参数），目前只能更改O_APPEND，O_ASYNC，O_DIRECT，O_NOATIME，O_NONBLOCK
  - F_GETOWN/F_SETOWN：管理I/O可用相关的信号。获得或设置当前文件描述符会接受SIGIO和SIGURG信号的进程或进程组编号
  - F_GETLK/F_SETLK/F_SETLKW：获得/设置文件锁，设置为F_GETLK需要传入flock的指针用于存放锁信息。S_SETLK也传入flock指针表示需要改变的锁的内容，如果不能设置则立即返回EAGAIN。S_SETLKW同S_SETLK，但在无法设置时会阻塞当前进程直到成功

- `int stat(const char *filename, struct *buf);`

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

  `int lstat(const char *filename, struct stat *buf);`

  获取文件的属性。最后一个是遇到符号链接时，能取到被链接的文件的属性（其他的只能取到链接文件自己的属性）。

  返回值：0；失败则-1

```
struct stat {
    mode_t st_mode;
    ino_t st_ino;
    dev_t st_rdev;
    nlink_t st_nlink;
    uid_t st_uid;
    gid_t st_gid;
    off_t st_size;
    time_t st_atime;
    time_t st_mtime;
    time_t st_ctime;
    long st_blksize;
    long st_blocks;
}
```

  `st_mode` 里存放了类型、权限等信息。

  另外注意下，这几个 `time_t` 是时间戳也就是long，不是C库里那个time_t

- `int access(const char *path, int mode);`

  根据当前的用户ID和实际组ID测试文件的存取权限

  mode：R_OK，W_OK，X_OK，F_OK（文件是否存在）

  返回值：0；失败则-1

- `int chmod(const char *path, mode_t mode);`

  `int fchmod(int fd, mode_t mode);`

  mode与st_mode中的第九位相同。

  返回值：0；失败则-1

- `int chown(const char *path, uid_t owner, gid_t group);`

  `int fchown(int fd, uid_t owner, gid_t group);`

  `int lchown(const char *path, uid_t owner, gid_t group);`

  更改文件的拥有者和组

  返回值：0；失败则-1

- `mode_t umask(mode_t mask);`

  更改存取权限屏蔽字（默认为022）

  返回值：之前的值

- `int link(const char *oldpath, const char *newpath);`

  `int unlink(const char *pathname);`

  创建/删除一个文件的硬链接

  返回值：0；失败则-1

- `int symlink(const char *oldpath, const char *newpath);`

  `int readlink(const char *path, char *buf, size_t bufsize);`

  创建/读取符号链接的值

  返回值：0；失败则-1

- `int mkdir(const char *pathname, mode_t mode);`

  `int rmdir(const char *pathname);`

  创建/删除空目录

  返回值：0；失败则-1

- `int chdir(const char *path);`

  `int fchdir(int fd);`

  更改当前工作目录

  返回值：0；失败则-1

- `char *getcwd(char *buf, size_t size);`

  获取当前工作目录

  返回值：buf；失败则NULL

- `DIR *opendir(const char *name);`

  打开目录

  返回值：DIR指针，类似FILE；失败则NULL

- `int closedir(DIR *dir);`

`struct dirent *readdir(DIR *dir);`

`off_t telldir(DIR *dir);`

`void seekdir(DIR *dir, off_t offset);`

不赘述

```
struct dirent {
    long d_ino;
    off_t d_off;
    unsigned short d_reclen;
    unsigned char d_type;
    char d_name [NAME_MAX + 1];
}
```

d_reclen不是文件名的长度，课件上这里是错的。表示的是这个记录的长度，计算方式：4(d_ino) + 4(d_off) + 2(d_reclen) + 1(d_type) + 1(padding) + 4N(d_name) = 12 + ceil(length_of(d_name))。d_name会自动补齐到4的倍数。如1.jpg和1234.jpg都是8，12345.jpg是12。

- `int lockf(int fd, int cmd, off_t len);`

  cmd：指定的操作类型

  - F_LOCK：给文件夹互斥锁。若已被加锁则阻塞直到成功
  - F_TLOCK：同上，但不会阻塞，直接失败
  - F_ULOCK：解锁
  - F_TEST：测试是否上锁。未上锁则0，否则-1

  len：从当前位置开始要锁住多长

  这个函数是对fcntl的一层封装

标准I/O库

标准库中的I/O围绕FILE对象，也就是流指针进行。预定义三个流指针，即标准输入stdin，标注你输出stdout，标准错误stderr

- 缓冲模式
  - 块缓冲（全缓冲，full buffered，block bufferd）
  - 行缓冲
  - 无缓冲
- `void setbuf(FILE *stream, char *buf);`

  `int setvbuf(FILE *stream, char *buf, int mode, size_t size);`

  mode：缓冲模式，_IOFBF（全缓冲），_IOLBF（行缓冲），_IONBF（无缓冲）

  buf：缓冲区，如果为NULL且mode不是_IONBF，库会调用malloc分配由size指定的大小的空间

  返回值：0；失败则非0
- `FILE *fopen(const char *filename, const char *mode)`

  mode：打开的模式："r"只读；"w"覆盖写；"a"追加；"r+"读写；"w+"读+覆盖写，且在文件不存在时自动创建；"a+"读+追加写，且在文件不存在时自动创建；"t"文本模式；"b"二进制模式。最后两个可以和之前的组合，如"rb"，"at+"等

  返回值：流指针；失败则NULL
- `int fclose(FILE *stream)`

  返回值：0；失败则EOF

- `int getc(FILE *fp);`

  `int fgetc(FILE *fp);`

  `int getchar(void);`

  `getchar`从标准输入读取。

  `getc`使用宏来实现的，所以要注意其参数不能有副作用。但效率会略高于`fgetc`

  返回值：转换成unsigned int的char值；读取到末尾或出错则EOF

- `int putc(int c, FILE *fp);`

  `int fputc(int c, FILE *fp);`

  `int putchar(int c)`

  返回值：写入的字符值；出错则-1

- `char *fgets(char *s, int size, FILE *stream);`

  `char *gets(char *s);`

  s：缓冲区

  后者不推荐，很容易溢出

  注意，会读取size - 1个字符，并在末尾添加\0。遇到文件尾或换行符会停止

  返回值：缓冲区头

- `int fputs(const char *s, FILE *stream);`

  `int puts(const char *s);`

  批量写入直到第一个\0（\0本身不写入）

  返回值：非负整数；出错则EOF

- `size_t fread(void *buf, size_t size, size nmemb, FILE *stream);`

  `size_t fwrite(const void *buf, size_t size, size_t nmemb, FILE *stream);`

  size：每次读/写的字节数

  nmemb：总共读/写几次。也就是说总共写入的字节数是size * nmemb。**注意这里课件上的解释是错的**

  返回值：成功读/写的次数

- `int scanf(const char *format, ...);`

  `int fscanf(FILE *stream, const char *format, ...)`

  `int sscanf(const char *str, const char *format, ...);`

  分别从标准输入，流，字符串扫描输入。注意后面的`...`是指可变参数。

  返回值：正确读取的变量个数

- `int printf(const char *format, ...)`

  `int fprintf(FILE *stream, const char *format);`

  `int sprintf(char *str, const char *format);`

  分别格式化输出到标准输出，流，字符串（包括一个\0）。返回值为写入的字符数，包括\0。

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

  和`lseek`差不多

- `long ftell(FILE *stream);`

  返回当前位置的偏移量

- `void rewind(FILE *stream);`

  将流指针移到文件开头

- `int fgetpos(FILE *fp, fpos_t *pos);`

  `int fsetpost(FILE *fp, const fpos_t *pos);`

也用来获取/移动位置，向/从pos参数存放/读取位置信息。新增这两个函数是为了处理大到超出long int范围的文件

返回值：0；失败则一个非零值

- `int fflush(FIILE *stream);`

  返回值：0；失败则EOF

- `int fileno(FILE *fp)`

  获取流指针对应的文件描述符

- `FILE *fdopen(int fd, const char *mode);`

  用已打开的文件描述符创建一个流

- `char *tmpnam(char *s);`

  返回一个当前未被使用的文件名

- `FILE *tmpfile(void);`

  创建一个临时文件

Chapter 4

内核

- 操作系统是一系列程序的集合，最重要的部分构成内核
- 单内核/微内核
  - 单内核又称宏内核，是一个整体，可以分成模块，运行时是一个独立的二进制文件，模块间通过直接调用函数进行通信
  - 微内核的各个部分作为独立的进程在特权模式下运行，通过消息传递进行通信
- Linux内核的能力
  - 文件管理，内存管理，进程管理，抢占式多线程支持，多处理器支持
- Linux区别于其他UNIX商业内核的优点
  - 单内核，模块支持
  - 免费，开源
  - 支持多种CPU，硬件支持能力非常强的
  - Linux开发者都是非常出色的程序员
  - 通过学习Linux内核可以了解现代操作系统的实现原理

建立初始化程序

`mkinitrd /boot/initrd.img $(uname -r)`

`mkinitramfs -o /boot/initrd.img $(uname -r)`

`update-initramfs -u`

内核模块

- 操作模块
  - 底层：`insmod`、`rmmod`
  - 高层：`modprobe`、`modprobe -r`
  - `moddep`、`lsmod`、`modinfo`

	C程序	内核模块
运行	用户空间	内核空间
入口	main	module_init()指定
出口	无	module_exit()指定
运行	直接运行	insmod
调试	gdb	kdbug，kdb，kgdb等

开发驱动的注意事项

- 不能用C库
- 没有内存保护
- 小内核栈
- 要考虑并发