标*非重点，配合理解

标#为掌握，知道即可

ch1-1 Linux-Basics

一、谈谈自己对linux的理解（#）

定义：根据GNU通用许可证开发的免费Unix类型的操作系统

特点：开源、受欢迎、支持大多数平台

历史：Unix 1969——GNU 1984——第一版 1991——发行版 1992

创始人：linus

发行版：RedHat（RHEL，CentOS），Debian（Ubuntu），Suse等。

二、分区形式（#）

1.MBR

512字节，0磁道第1扇区。

446字节引导代码

64字节磁盘分区表，最多3条目

2字节的0X55AA

2.GPT和GUID

GPT与MBR并称两种分区类型。

GPT的全称是GUID Partition Table Scheme，即全局唯一标识分区表。

3.分区的限制和好处

限制：分区数目限制（4主分区或3主1扩），分区大小限制（受文件系统和操作系统限制）。

好处：数据保护、数据整理、系统管理、多操作系统。

三、文件系统的概念（#）

操作系统中负责访问和管理文件的部分

Linux常用文件系统：VFS、EXT2、EXT3、FAT32……

（这里我实在受不了了，想吐槽一句，这PPT至少得有20年了，最常用的是EXT4，XFS，SWAP，EFI，VFAT等）

四、分区（*）

分区参数 浏览文件 扇区编辑

卷标	序号(状态)	文件系统	标识	起始柱面	磁头	扇区	终止柱面	磁头	扇区	容量	属性
efi(F:)	0	FAT32	0B	0	32	33	131	13	10	1.0GB	A
主分区(1)	1	Linux Swap	82	131	13	11	2219	215	44	16.0GB	
/(2)	2	EXT4	83	2219	215	45	23106	163	33	160.0GB	
/home(3)	3	EXT4	83	23106	163	34	62260	121	5	299.9GB	

至少需要/（序号2）和swap分区（序号1）

推荐/boot（这里存在efi中）

还可以添加如/home等

五、Bootloader、LILO、Grub（*）

Bootloader（引导）的作用：引导程序加载并启动linux内核、传递启动参数、选择加载初始根磁盘、启动其它操作系统。

通用引导包括LILO和GRUB。

Grub其实是GRand Unified Bootloader，汉语为GRand统一引导加载程序。
存储在MBR中（第一阶段）和/boot/grub（1.5阶段和2阶段）；
配置文件boot/grub/grub.conf
通过grub-install安装在MBR

六、安装软件（#）

从tar安装（开源软件源代码）

```
1. tar zxvf application.tar.gz
2. cd application
3. ./configure
4. make
5. su -
6. make install
```

自动安装

```
apt-get command *（update 获取最新软件包 upgrade 升级到已安装所有软件包）
dpkg 手动安装
仅需了解：aptitude、yum、rpm
```

七、命令行提示符和命令

```
$代表普通用户、#代表root用户
```

基本命令：
- passwd：更改密码
- mkpasswd：生成随机密码
- date，cal:生成日期和日历
- who、finger：找出谁在系统上处于活跃状态
- clear：清屏
- echo：屏幕上写一条消息
- write、fall、talk、mesg
  - write：给其他用户发信息
  - wall：write all给所有登录到系统的用户发信息
  - talk：建立聊天session
  - mesg：可以屏蔽用户发来的信息

目录相关命令：
- pwd：打印工作目录
- cd：更改目录
- mkdir：创建目录
- rmdir：删除目录
- ls：列出目录的内容
  - -l：使用较长格式列出
  - -a：打印.开始的目录
  - -R:递归显示子目录

文件相关命令：
- touch：更新文件的访问和/或修改时间 OR 创建新文件
- cp：复制文件
- mv：移动并重命名文件
- ln：链接文件
  - -s：软链接，指向被链接文件的inode（inode不一样）
  - 默认硬链接，inode相同
- rm：删除文件
- cat：打印文件内容
- more/less：逐页显示文件，more不可以回退，less可以回退
- chmod:修改权限
- find：寻找文件

进程相关命令：
- ps：报告进程状态
  - 示例：ps -aux|grep 。。。
  - -e：列出所有进程
  - -aux：列示所有进程并且列出详细信息
- pstree：显示进程树
- jobs，fg，bg，：作业控制
  - bg：后台执行
  - fg：前台执行
- kill：杀死进程 -9表示强制杀死
- nohup：运行命令，忽略挂断信号
- nice：设置进程优先级
- top：显示最热门的CPU进程

八、七种文件类型

```
普通文件
特殊文件：字符特殊文件、块特殊文件
socket文件
符号链接：软连接、硬链接
目录文件
```

九、目录结构

```
/boot 内核相关 /etc 配置文件 /bin 二进制程序 /mnt 临时挂载 /root root用户 /home  用户目录
```

十、文件权限

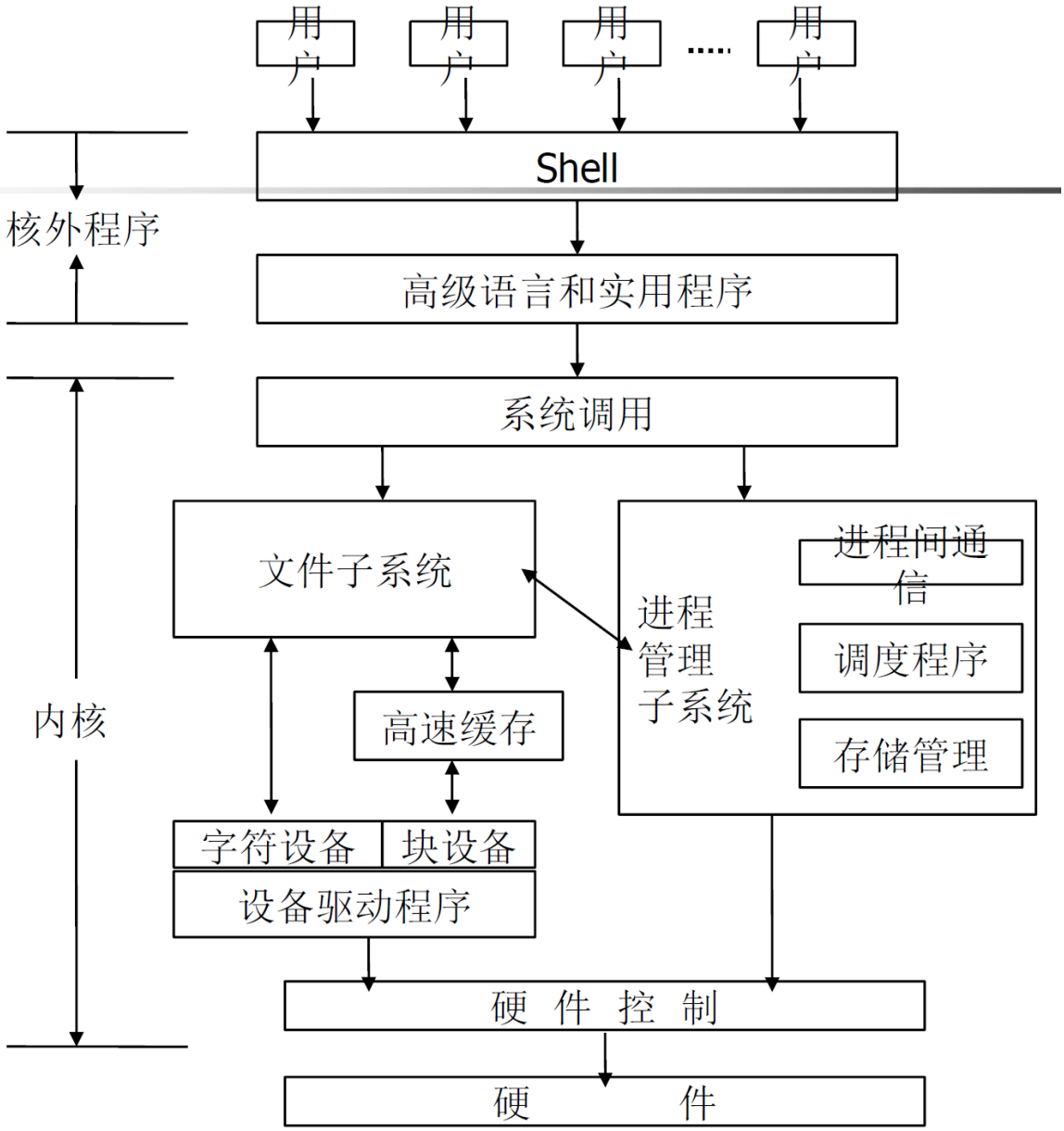
```
文件权限有三个权限等级 用户User 组Group 其它Others
有三个权限 读r 写w 执行r，对应二进制三位数，如chmod 777 把所有权限赋给所有用户
```

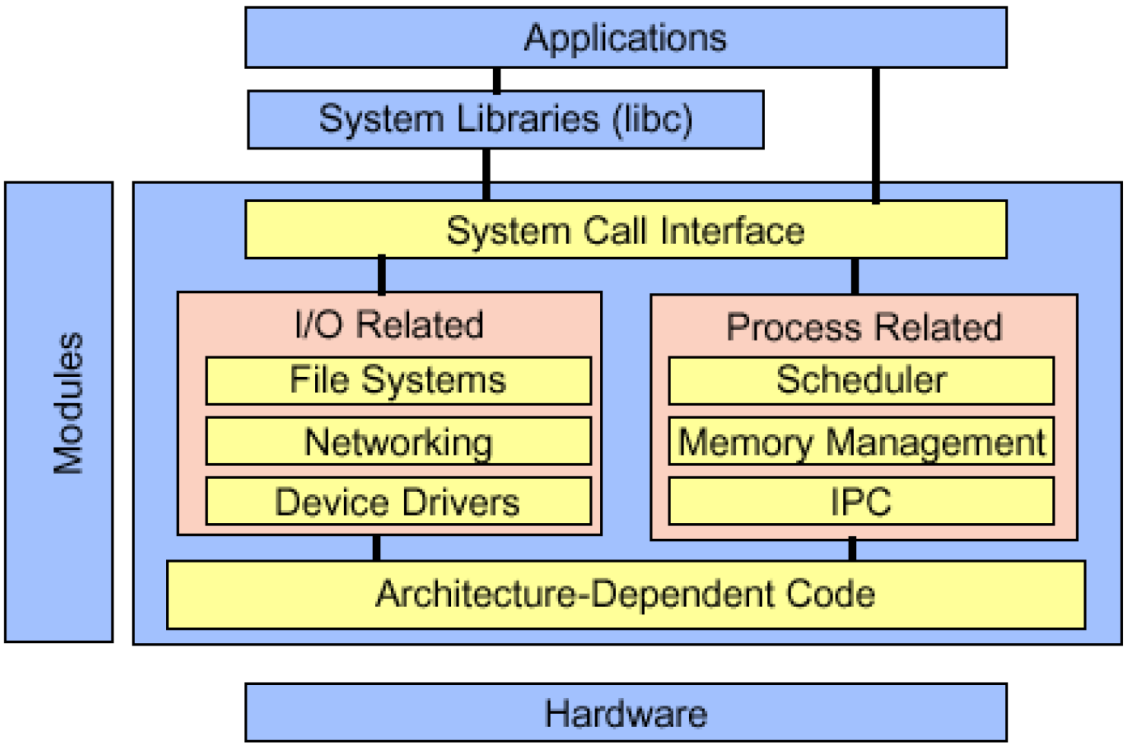
十一、进程概念（*）

一个正在执行的程序实例。由执行程序、当前值、状态信息、以及通过操作系统管理此进程执行情况的资源组成。

ch1-2 Linux-Basics

一、层次图（#）


用户
用户
用户
.....
用户
Shell
核外程序
高级语言和实用程序
系统调用
文件子系统
进程间通信
进程管理子系统
调度程序
存储管理
内核
高速缓存
字符设备
块设备
设备驱动程序
硬 件 控 制
硬 件


Applications
System Libraries (libc)
Modules
System Call Interface
I/O Related
File Systems
Networking
Device Drivers
Process Related
Scheduler
Memory Management
IPC
Architecture-Dependent Code
Hardware

二、重定向

重定向是一种将命令的输出或输入从一个默认位置转移到另一个位置的方式。

比如，将标准输出导向一个文件或追加到一个文件中。

```
command > file.txt //运行command命令，并将其输出重定向到名为file.txt的文件中，而不是在终端上打印
command < file.txt //运行command命令，并从名为file.txt的文件中读取输入，而不是从终端获取
ls >> file。txt //运行ls命令，并将结果追加到file.txt文件的末尾
同理，不同的组合表示不同的意思：
< 输入重定向
> 输出重定向
>! 输出重定向外加强制覆盖
<< 追加输入重定向 你可能不理解 但我放一条别的代码就理解了 cout << "Hello World!" << endl
>> 追加输出重定向 不清楚文件内容而是追加
已知，标准输入、标准输出、标准错误对应的文件描述符为0、1、2，所以还有：
2> 错误重定向：将错误的信息输入到后面的文件中
2>> 错误追加重定向：将错误信息追加到后面文件中
下面举几个例子：
command > file 2>&1 //执行command命令，第一个>代表标准输出重定向到文件file，2>&1代表将将标准错误输出（文件描述符为 2）重定向到标准输出（文件描述符为 1）的位置，实现同时输出stdout和stderr。
command < file1 > file2 //stdin重定向到file1 stdout重定向到file2
```

三、管道

将一个进程的输出作为另一个进程输入

```
例如，假设我们有一个名为 "input.txt" 的文件，并且我们想要将其内容按行排序并只显示唯一的行。
sort input.txt | uniq
这将使用管道将 "input.txt" 文件中的内容传递给 "sort" 命令，然后将 "sort" 命令的输出传递给 "uniq" 命令。

如 ls / | grep "str"
具体来说，/ 表示根目录，ls / 命令会列出根目录下的所有文件和目录。管道符 | 将 ls 命令的标准输出传递给 grep 命令作为标准输入，grep 命令会在其中搜索包含指定字符串的行，并将结果输出到标准输出中。... 是要搜索的字符串，可以是任何字符串。

如 ls | wc -l
具体来说，ls 命令会列出当前目录下的所有文件和目录，并将它们的名称输出到标准输出中。管道符 | 将 ls 命令的标准输出传递给 wc 命令作为标准输入，wc 命令会统计标准输入中的行数、单词数和字符数，并将结果输出到标准输出中。-l 选项指定只统计行数。
```

四、环境变量

分为用户环境变量和系统环境变量

echo命令

```
echo $HOME //打印当前用户的主目录路径
HOME还可改为PATH、SHELL等,分别代表“当前用户的默认shell,如bin/bash”，“告诉shell在哪里寻找命令或可执行文件，
如/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin”
```

env命令

```
使用env打印所有环境变量及其值，如：
$ env
LANG=en_US.UTF-8
HOME=/home/user
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/us
r/local/games:/snap/bin
```

set命令

```
列出所有环境变量、其它shell变量的值，如：
$ set
SHELL=/bin/bash
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/us
r/local/games:/snap/bin
HOME=/home/user
USER=user
LANG=en_US.UTF-8
PWD=/home/user
```

设置环境变量

```
export name="value"//退出shell便失效
```

系统环境变量

```
/etc/profile
/etc/bashrc
```

用户环境变量

```
.bash_profile
.bashrc
```

五、简单的正则与高级命令（#）

这里老师原话：只考最简单的，也不知道他说的最简单指的是啥.....

```
https://blog.csdn.net/qq_48391148/article/details/125566389
```

OK

linux find命令

`find` 是一个在 Linux 系统中用于查找文件和目录的强大命令。它可以按照文件名、文件类型、文件大小、文件日期等多个条件来查找文件，并支持通过执行命令对查找结果进行进一步处理。

以下是一些常用的 `find` 命令选项：

- `-name`：按照文件名查找，可以使用通配符。
- `-type`：按照文件类型查找，可以指定文件类型，例如 `f` 表示普通文件，`d` 表示目录。
- `-size`：按照文件大小查找，可以指定文件大小范围，例如 `+10M` 表示查找大于 10MB 的文件。
- `-mtime`：按照文件修改时间查找，可以指定天数范围，例如 `-mtime -7` 表示查找 7 天内修改过的文件。
- `-exec`：对查找结果执行指定的命令，例如 `-exec rm {} \;` 表示删除所有查找到的文件。

以下是一些 `find` 命令的例子：

- `find / -name "*.txt"`：在根目录下查找所有扩展名为 `.txt` 的文件。
- `find /home -type f -size +10M`：在 `/home` 目录下查找所有大于 10MB 的普通文件。
- `find /var/log -type f -mtime -7 -exec gzip {} \;`：在 `/var/log` 目录下查找所有 7 天内修改过的普通文件，并将其压缩为 `.gz` 格式。

OK

linux sed命令

`sed`是一个在Linux和Unix系统上常用的命令行工具，它用于文本处理和编辑。下面是一些常见的`sed`命令：

1. 替换字符串：`sed 's/old/new/g' file.txt`
   这个命令将会把文件file.txt中所有出现的"old"字符串替换成"new"。
2. 打印文件内容：`sed 'p' file.txt`
   这个命令会将文件file.txt的内容输出两次。
3. 删除指定行：`sed '5d' file.txt`
   这个命令将会删除文件file.txt的第5行。
4. 在指定行前或后添加文本：`sed '5i\This is a new line.' file.txt`或`sed '5a\This is a new line.' file.txt`
   这个命令将在文件file.txt的第5行前（i）或后（a）添加文本"This is a new line."。
5. 删除匹配的行：`sed '/pattern/d' file.txt`
   这个命令将删除文件file.txt中匹配"pattern"的所有行。
6. 只对某些行进行操作：`sed '2,4s/old/new/g' file.txt`
   这个命令将只对文件file.txt的第2到第4行进行替换操作。
7. 将匹配到的文本进行替换：`sed 's/\(old\)/\1 new/g' file.txt`
   这个命令将文件file.txt中所有出现的"old"替换为"old new"，其中"\1"表示第一个括号内匹配到的文本。

这些都是`sed`命令中最常见的一些，`sed`命令非常强大，可以通过多种方式组合使用，以实现不同的文本处理和编辑操作。

OK linux grep命令

`grep`是一种强大的文本搜索工具，用于在文本文件中查找特定的字符串模式。以下是一些常用的`grep`命令：

1. 查找包含特定字符串的行

```perl
grep "string" file.txt
```

这将在`file.txt`文件中查找包含`string`字符串的所有行，并将它们打印到终端。

2. 查找不包含特定字符串的行

```perl
grep -v "string" file.txt
```

这将在`file.txt`文件中查找不包含`string`字符串的所有行，并将它们打印到终端。

3. 指定多个文件进行搜索

```perl
grep "string" file1.txt file2.txt
```

这将在`file1.txt`和`file2.txt`文件中查找包含`string`字符串的所有行，并将它们打印到终端。

4. 搜索包含特定字符串的文件

```perl
grep -l "string" *
```

这将在当前目录中搜索所有包含`string`字符串的文件，并将它们的文件名打印到终端。

5. 搜索指定类型的文件

```perl
grep "string" *.txt
```

这将在当前目录中搜索所有扩展名为`.txt`的文件，并查找包含`string`字符串的所有行，并将它们打印到终端。

6. 忽略字母大小写进行搜索

```perl
grep -i "string" file.txt
```

这将在`file.txt`文件中查找包含`string`字符串的所有行，并忽略大小写。

这些只是`grep`命令的一些常见用法，它还有许多其他选项和用法。可以通过`man grep`命令来查看完整的帮助文档。

ch2 Shell Programming

一、常见的shell（*）

```
ash ash.static bsh bash sh csh tcsh  ksh
```

二、编写脚本文件

预备知识

binbash

```
#!/bin/bash是一个shebang的特殊指令，告诉操作系统用哪个解释器来执行文件
```

注释

```
以#开头
```

退出码

```
exit 0
```

执行脚本文件

```
sh script_file
chmod +x filename then ./script_file
source script_file or ./script_file
```

用户环境

```
.bash_profile 用户登录被读取 包含的命令被bash执行
.bashrc 启动一个新的shell时读取并执行
.bash_logout 登录退出时读取执行
alias/unalias 命名/解除命名
export MY_VAR="Hello World"
```

变量

用户变量

是用户在shell脚本里定义的变量

```
var = value
echo $var
```

read命令

```
直接读取并存入变量
echo -n "Enter your name:" //-n不换行
read name

在read命令行中直接指定一个提示 -p
read -p "Enter your name:" name

read后面什么也不跟会存到reply变量中
read -p "Enter a number"
echo $REPLY

指定输入时间 -t
if read -t 5 -p "please enter your name:" name
then
  echo "hello $name， welcome to my script"
else
  echo "sorry, too slow"
fi

其它参数
-s 不显示输入
-a 将分裂后的字段存储到指定数组中
-d 指定读取行的结束符号
-n 限制读取n个字符自动结束，回车和换行可以结束
-N 强制读满n个字符结束，换行算一个字符
-r 禁止转义
```

引号&转义符号

```
单引号：所有字符保持本身字符的意思
双引号：除了$、''、\，双引号内的所有字符将保持字符本身的含义不被bash解释
转义符号：改变下一个字符的含义，将它们变为最原始的状态
```

环境变量

shell环境提供的变量。通常使用大写字母做名字

```
$HOME 当前用户的登陆目录
$PATH 以冒号分隔的用来搜索命令的目录清单
$PS1 命令行提示符 通常是$
$PS2 辅助提示符 通常是>
$IFS 输入区分隔符 通常是空格、制表符、换行符等
```

参数变量和内部变量

```
$# 传递到脚本程序的参数个数
$0 脚本程序的名字
$1,$2…… 脚本程序的参数
$* 全体参数组成的清单，它是一个独立的变量，各个参数之间用环境变量IFS中的第一个字符分隔开
$@ $*的一种变体，不使用IFS环境变量
```

命令行输入 `./test.sh apple orange banana`

```
#!/bin/bash
echo "The first parameter is $1"
echo "The second parameter is $2"
echo "The third parameter is $3"
echo "The number of parameters is $#"
echo "All the parameters are $*"
echo "The script name is $0"
exit 0
```

条件测试

退出码 exit

test命令:字符串比较、算术比较；文件相关条件测试、逻辑操作。

字符串比较

```
- str = str
- str != str
- -z str:字符串为空
- -n str：字符串为非空
```

算术比较

```
expr1 operator expr2
-eq 相等结果为真
-ne 不等结果为真
-gt （greater than）exp1大于exp2
-ge （greater than or euqal）exp1大于等于exp2
-lt -le同理
```

文件相关

```
-e file 文件存在则结果为真
-d file 文件是一个子目录则结果为真
-f file 文件是一个普通文件则结果为真
-s file 文件的长度不为0则结果为真
-r/-w/-x file 文件可读/写/执行结果为真
```

逻辑操作

```
!expr 逻辑表达式求反
expr1 -a expr2 与
expr1 -o expr2 或
```

条件语句

if

```
if [ expression ]
then
statements
elif [ expression ]
then
statements
elif …
else
statements
fi
```

```
if [ "$answer" = "yes" ]; then
  echo "Good morning"
elif [ "$answer" = "no" ]; then
  echo "Good afternoon"
else
  echo "Sorry, $answer not recognized. Enter yes or no"
  exit 1
fi
```

case

```
case str in
  str1 | str2) statements;;
  str3 | str4) statements;;
  *) statements;;
esac
```

```
#!/bin/sh
echo “Is this morning? Please answer yes or no.”
read answer
case “$answer” in
yes | y | Yes | YES) echo “Good morning!” ;;
no | n | No | NO) echo “Good afternoon!” ;;
*) echo “Sorry, answer not recognized.” ;;
esac
exit 0
```

循环语句

for

```
for var in list
do
  statements
done

#!/bin/sh
for file in $(ls f*.sh); do
  lpr $file
done
exit 0
```

while

```
while condition
do
  statements
done

a=0
while [ "$a" -le "$LIMIT" ]
do
  a=$(($a+1))
  if [ "$a" -gt 2 ]
  then
    break # Skip entire rest of loop.
  fi
  echo -n "$a"
done
```

until（不推荐）

```
until condition
do
  statements
done
```

select（生成菜单列表）

```
select item in itemlist
do
  statements
done

#!/bin/bash
clear
select item in Continue Finish
do
  case "$item" in
    Continue) ;;
    Finish) break ;;
    *) echo "Wrong choice! Please select again!" ;;
  esac
done
```

命令表和语句块

命令表

```
分号串联：command1;command2;……
条件组合：
    AND命令表：stmt1 && stmt2 && ……
    OR命令表：stmt1 || stmt2 || ……
```

语句块

```
{
statement1
statement2
...
}
```

语句块的主要载体——函数

```
func()
{
statements
}

例子：
yesno()
{
msg=“$1”
def=“$2”
while true; do
echo ” ”
echo “$msg”
read answer
if [ -n “$answer” ]; then
    case “$answer” in
    y|Y|yes|YES)
    return 0
    ;;
    n|N|no|NO)
    return 1
    ;;
    *)
    echo “ ”
    echo “ERROR: Invalid response,
    expected \”yes\” or \”no\”.”
    continue
    ;;
    esac
else
return $def
fi
done
}
```

msg=“$1”：将传入的第一个参数作为提示信息保存到 msg 变量中。
def=“$2”：将传入的第二个参数作为默认值保存到 def 变量中。
while true; do：无限循环开始。
echo ” ”：打印一个空行。
echo “$msg”：打印提示信息。
read answer：读取用户输入并将其保存到 answer 变量中。
if [-n “$answer”]; then：如果用户输入不为空，则执行下面的代码块。
case “$answer” in：使用 case 语句检查用户输入，并根据不同的情况执行不同的代码块。
y|Y|yes|YES)：如果用户输入是 "y"、"Y"、"yes" 或 "YES"，则执行下面的代码块。
return 0：返回值为 0，表示用户选择了"yes"。
n|N|no|NO)：如果用户输入是 "n"、"N"、"no" 或 "NO"，则执行下面的代码块。
return 1：返回值为 1，表示用户选择了"no"。

*)：如果用户输入不是 "y"、"Y"、"yes"、"YES"、"n"、"N"、"no" 或 "NO"，则执行下面的代码块。
echo “ ”：打印一个空行。
echo “ERROR: Invalid response, expected \”yes\” or \”no\”.：打印错误消息，提示用户输入了无效的值。
continue：继续下一轮循环。
else：如果用户输入为空，则执行下面的代码块。
return $def：返回默认值，即传入的第二个参数。
done：循环结束。

其它

杂项命令

- break: 从for/while/until循环退出
- continue: 跳到下一个循环继续执行
- exit n: 以退出码"n"退出脚本运行
- return: 函数返回
- export: 将变量导出到shell，使之成为shell的环境变量
- set: 为shell设置参数变量
- unset: 从环境中删除变量或函数
- trap: 指定在收到操作系统信号后执行的动作
- “:”(冒号命令): 空命令
- “.”(句点命令)或source: 在当前shell中执行命令

捕获命令输出

语法：$(command)或`command`

```
#!/bin/sh
echo “The current directory is $PWD”
echo “The current directory is $(pwd)”
exit 0
```

第一个语句 echo “The current directory is $PWD” 使用了 Shell 变量 $PWD。$PWD 是一个特殊变量，保存着当前工作目录的路径。在该语句中，使用了双引号包裹了字符串和变量，Shell 会将 $PWD 变量替换为当前工作目录的路径，并将整个字符串打印到屏幕上。

第二个语句 echo “The current directory is $(pwd)” 使用了命令替换。在这个语句中，$(pwd) 表示执行 pwd 命令并将其输出作为字符串嵌入到整个 echo 语句中。Shell 在执行该语句之前会先执行 pwd 命令，获取当前工作目录的路径，并将路径字符串嵌入到 echo 语句中。

因此，这两个语句都可以打印当前工作目录的路径，但是使用了不同的方式。第一个语句使用了变量 $PWD，而第二个语句使用了命令替换 $(pwd)。在大多数情况下，这两种方式都可以得到相同的结果，但是在某些情况下可能会有细微的差别。例如，在一个子 Shell 中运行时，$PWD 可能会返回父 Shell 的工作目录，而 $(pwd) 则会返回当前 Shell 的工作目录。

（经验证，结果是一样的）

算术扩展——$((.....))

```
#!/bin/sh
x=0
while [ “$x” -ne 10 ]; do
echo $x
x=$(($x+1))
done
exit 0
```

参考扩展——批处理文件时用

```
#!/bin/sh
i=0
while [ "$i" -ne 10 ]; do
  touch "${i}_tmp"
  i=$(($i+1))
done
exit 0
```

here document

```
#!/bin/bash
cat >> file.txt << !CATINPUT!
Hello, this is a here document.
!CATINPUT!
```

ch3-0 Programming Prerequisite

一、编程工具（*）

```
编辑工具：vi emacs
编译链接：gcc
调试工具：gdb
make命令
版本控制工具：cvs
```

二、编程语言（*）

脚本语言

shell:sh/bash,csh,ksh等

Perl,Python,tcl/tk,sed,awk等

高级语言

C/CPP，JAVA，Fortran等

ELF：linux下的可执行文件封装格式

执行方式

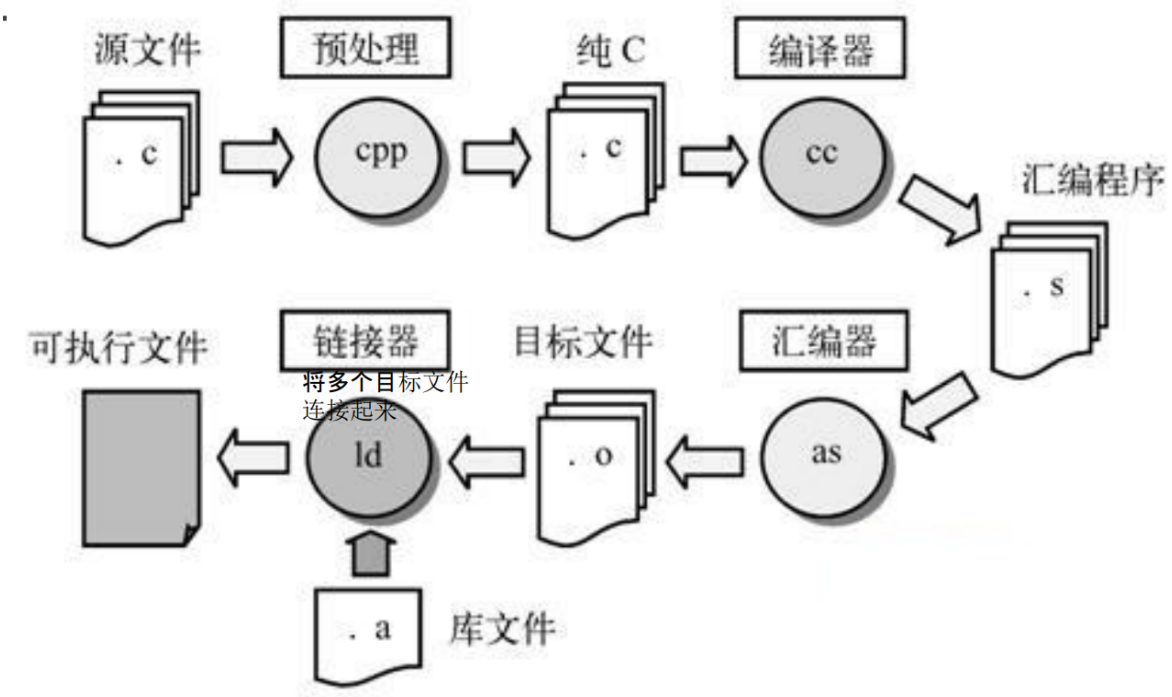
编译执行：先编译成本地字节码，再配备相应解释器，解释为CPU可以执行的二进制码，然后放到CPU执行，比如**C、C++、Java、Go、Rust、Swift**

解释执行：不编译，读一行执行一行，比如**Python、Ruby、JavaScript、Perl、PHP**

三、gdb功能

调试工具，用于调试C、C++、汇编等编程语言编写的程序

四、编译链接图解



如何处理头文件：在预处理阶段，按文件名找到头文件，用头文件的文本内容代替头文件

动态库与静态库：动态库不放在可执行文件，升级方便，但是会有版本冲突；静态库开发推进，程序变大，需要静态库降低复杂度，会导致复用性降低。

链接的作用：将所有目标文件链接成一个可执行文件，并将符号表和重定位表等信息写入可执行文件中。

五、GCC & 文件扩展名

GCC

-E：只对源程序进行预处理

-S：只对源程序进行预处理、编译

-c：执行预处理、编译、汇编但不链接

```
gcc -c hello.c
```

-o output_file：-o创建可执行二进制文件

```
//链接多个.o成为二进制文件
gcc file1.o file2.o -o myprogram
//编译但不链接
gcc -c -o main.o main.c
```

-g：产生调试符号信息

-O/On：在程序编译、链接过程中进行优化处理

-Wall：显示所有警告

不常考的不放了

文件扩展名（*）

.c .cc .cp .cpp .CPP .c++ .C .cxx 需要预处理的源代码

.i .ii 不需要预处理的源代码

.h .H .hh 头文件

.s .S 汇编代码 后者必须预处理

.o 目标文件

.a 静态库

.so 动态库

六、make、makefile & 预定义变量

make

命令根据makefile对程序进行管理和维护，判断被维护文件的时序关系

命令格式：make [-f Makefile] [option] [target]

```
# automake方式
./configure #生成新的makefile
make
make install
make uninstall
make clean
make distclean# 退回到configure之前(删除makefile)
```

makefile语法

```
target ... : prerequisites ...
    command
```

顶格为规则 缩进为命令

target是一个目标文件

prerequisities是生成target所需的文件或目标

command是make需要执行的命令

（命令默认回显 不要回显在命令前添加@）

（%.o表明要所有以.o为结尾的目标）

```
hello : main.o kbd.o
  gcc -o hello main.o kbd.o
main.o : main.c defs.h
  cc -c main.c
kbd.o : kbd.c defs.h command.h
  cc -c kbd.c
clean :
  rm edit main.o kbd.o # 伪目标
```

判断执行次序

找到makefile文件

查找第一个target

如果target不存在或者target依赖文件修改时间比target新，执行command来生成新的target

target所依赖的.o文件不存在，递归调用

伪目标

```
clean :
  rm edit main.o kbd.o # 伪目标
```

make无法生成它的依赖关系，无法决定它是否要执行，只能通过显式指明目标让其生效

伪目标不能和文件名重名，可以使用.PHONY标记指明为伪目标

常用的伪目标：

```
all：编译所有的目标
clean：清理生成的文件
install：安装程序
uninstall：卸载程序
check：运行测试
```

多目标（*）

```
bigoutput littleoutput : text.g
generate text.g -$(subst output,,$@) > $@

#上述规则等价于
bigoutput : text.g
  generate text.g -big > bigoutput
littleoutput : text.g
  generate text.g -little > littleoutput
```

预定义变量（*）

- $< 第一个依赖文件的名称
- $? 所有的依赖文件，以空格分开，这些依赖文件的修改日期比目标的创建日期晚
- $+ 所有的依赖文件，以空格分开，并以出现的先后为序，可能包含重复的依赖文件
- $^ 所有的依赖文件，以空格分开，不包含重复的依赖文件
- $* 不包括扩展名的目标文件名称
- $@ 目标的完整名称
- $% 如果目标是归档成员，则该变量表示目标的归档成员名称

ch3-1 System Programming

一、文件系统

本部分较长，重要的有：文件系统概念、VFS、硬软链接、系统调用和库函数、IO等。

文件和文件系统

文件

文件：能读、能写或二者兼备的对象

文件系统

文件系统：

1.一种特定的文件格式

2.按特定格式进行了格式化的块存储介质

3.**操作系统中用来管理文件系统以及对文件进行操作的机制和实现**

VFS & VFS MODEL

VFS

作用：屏蔽系统空间中各种文件系统之间的差异

VFS MODEL

一定有一道题，即下面的四个组件及其作用

super block:某一磁盘某一分区的文件系统的信息，包括类型和参数

inode对象:记录真正的文件，文件存储在磁盘上时按照索引号访问文件

file对象:记录通过open创建的文件描述符而不是真的文件，文件需要close之后才能释放文件对象

dentry对象:表示文件或目录在目录中的信息

硬链接与软链接

其中软链接又称符号链接。

硬链接

在文件中创建一个新的目录项，将其与原文件所在的存储区域关联起来，形成一个新的文件名，新文件与原文件共享同一个inode和数据块，即它们指向同一个物理数据。不能链接目录。

删除原文件不影响使用

不能跨文件系统

对应系统调用link

软链接

包含了指向另一个文件的路径名，而不是直接包含数据。既可以链接目录，又可以链接不同文件系统中的文件。

删除原文件不能使用

对应系统调用symlink

系统调用和库函数

均以C函数形式出现

系统调用

系统调用是linux内核对外的接口，也是用户程序和内核之间唯一的接口，提供最小接口

库函数

库函数依赖于系统调用，提供复杂功能

无缓冲IO和缓冲IO

无缓冲IO

read/write命令直接通往系统调用

使用文件描述符进行读写操作

并不是ANSI C，而是POSIX.1 和 XPG3

有缓冲IO

通过标准IO库实现

处理细节

字节流是指向FILE的指针

IO系统调用

文件描述符

小的非负整数

unistd.h中包含了与文件描述符相关的函数原型和常量定义

文件操作的一般步骤：打开-读写-寻道-关闭

追求及格的话后面的函数不要追求全背下来，太多了

open/creat，close，read，write，lseek

```
#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);
int creat(const char *pathname, mode_t mode);
//成功返回文件描述符，不成功返回-1
```

open函数用来打开一个文件，如果文件不存在则创建该文件。其中，pathname是要打开的文件路径，flags是打开文件的选项，mode是文件权限。

其中，flags参数可以设置为以下标志的按位或：

- O_RDONLY：只读方式打开文件
- O_WRONLY：只写方式打开文件
- O_RDWR：读写方式打开文件
- O_APPEND：在文件末尾追加数据
- O_CREAT：如果文件不存在则创建文件
- O_TRUNC：清空文件内容
- O_EXCL：如果同时设置了O_CREAT，那么当文件已经存在时open会失败，而O_EXCL选项可以保证在打开文件时不会覆盖已经存在的文件。

mode参数用于指定新创建文件的权限，当flags参数包含O_CREAT选项时，mode必须提供。
mode_t实际上是一个八进制数字，他的后三位代表了拥有者权限、组权限、其它用户权限。
1代表可执行，2代表可写，4代表可读。

1 可执行，2 可写，4 可读

The value of parameter "mode"

取值 前面两个0有含义！！	含义
S_IRUSR(00400)	Read by owner
S_IWUSR(00200)	Write by owner
S_IXUSR(00100)	Execute by owner
S_IRWXU(00700)	Read, write and execute by owner
S_IRGRP 00040	Read by group
S_IWGRP 00020	Write by group
S_IXGRP 00010	Execute by group
S_IRWXG 00070	Read, write and execute by group
S_IROTH 00004	Read by others
S_IWOTH 00002	Write by others
S_IXOTH 00001	Execute by others
S_IRWXO 00007	Read, write and execute by others

creat是一种特殊的open，相当于open的flag参数设置为O_WRONLY|O_CREAT|O_TRUNC。

mode参数会和umask构成保护机制

- umask: a file protection mechanism
- The initial access mode of a new file
  - mode & ~umask

regular files:

default permissions	rw-rw-rw-	666
umask (-)	----w--w-	022
resulting permissions	rw-r--r--	644

directories:

default permissions	rwxrwxrwx	777
umask (-)	----w--w-	022
resulting permissions	rwxr-xr-x	755

```
#include <unistd.h>
int close(int fd);
//0关闭成功 1关闭失败
```

```
#include <unistd.h>

ssize_t read(int fd, void *buf, size_t count);
//fd 文件描述符 buf 写入的数据缓冲区 count 写入的字节数
//(返回值：实际读到的字节数，若已到文件尾返回0，若出错为-1)

ssize_t write(int fd, const void *buf, size_t count);
//(返回值：若成功为已写的字节数，若出错为-1)

while ((n = read(STDIN_FILENO, buf, BUFSIZE)) > 0)
   if (write(STDOUT_FILENO, buf, n) != n)
      err_sys("write error");
   if (n<0)
      err_sys("read error");
```

```
#include <unistd.h>

off_t lseek(int fd, off_t offset, int whence);
//fd 文件描述符 offset 偏移量 whence 偏移起始位置
//将文件读写位置从whence位置开始向前或向后移动offset个字节，返回新的读写位置。
//成功返回新的读写位置 失败返回-1
whence有以下三个取值：
SEEK_SET：从文件开头开始偏移。
SEEK_CUR：从当前读写位置开始偏移。
SEEK_END：从文件结尾开始偏移。
```

dup/dup2

```
#include <unistd.h>
int dup(int oldfd);
//返回新的文件描述符，和oldfd指向同一个文件 错误返回-1
int dup2(int oldfd, int newfd);
//如果newfd被打开，关闭newfd对应的文件，执行dup操作，成功返回newfd，失败返回-1

使用dup完成重定向
int fd=-1;
char buffer[100];
int len=0;
fd=open("4.txt",O_CREAT|O_RDWR);
dup2(fd,STDOUT_FILENO);
printf("hello world");//prtinf-stdout-fd-4.txt
close(fd);
```

fcntl（*)

```
用于控制文件描述符——复制、获取/设置 文件描述符/文件状态 标志、获取/设置文件锁
#include <unistd.h>
#include <fcntl.h>
int fcntl(int fd, int cmd);
int fcntl(int fd, int cmd, long arg);
int fcntl(int fd, int cmd, struct flock *lock);
F_DUPFD：复制文件描述符。
F_GETFD/F_SETFD：获取/设置文件描述符标志。
F_GETFL/F_SETFL：获取/设置文件状态标志。
F_GETLK/F_SETLK/F_SETLKW：获取/设置文件锁。
```

ioctl（*)

```
进行设备控制
#include <sys/ioctl.h>
int ioctl(int d, int request, ...);
fd 文件描述符 request 设备控制命令序号
```

标准IO库

```
//1.开闭文件
#include <stdio.h>
FILE *fopen(const char *filename, const char *mode);//成功返回文件指针 失败null
int fclose(FILE *stream);//0成功，-1失败
r 只读/不存在失败
w 打开一个文本文件并清空/不存在就创建新文件
a 追加/不存在就创建新文件
r+ r的基础上允许写
w+ w的基础上允许读
a+ a的基础上允许读

//2.读取或写入一个字符
int getc(FILE *fp);
int fgetc(FILE *fp);//读取文件流的下一个字符
int getchar(void);//读取输入
```

```
int putc(int c, FILE *fp);//向文件流写一个字符
int fputc(int c, FILE *fp);
int putchar(int c);//输出一个字符

//3.读取或写入多个字符
char *fgets(char *s, int size, FILE *stream);//读取size-1字符存入s
char *gets(char *s);
int fputs(const char *s, FILE *stream);
int puts(const char *s);

//4.读写二进制数据 && 举例
size_t fread(void *ptr, size_t size, size_t nmemb, FILE *stream);
ptr是指向读取数据缓冲区的指针，size是每个数据项的大小并以字节为单位，nmemb是要读取的数据项的数量，stream是指向要读取的文件流的指针，用于从指定的文件流中读取二进制数据
size fwrite(const void *ptr, size_t size, size_t nmemb, FILE *stream);
同理，向指定文件流写入二进制数据
    float data[10];
    if ( fwrite(&data[2], sizeof(float), 4, fp) != 4 )
       err_sys("fwrite•error");
    struct {
       short count; long total; char name[NAMESIZE];
    }item;
    if ( fwrite(&item, sizeof(item), 1, fp) != 1)
       err_sys("fwrite•error");

//5.标准输入输出
int scanf(const char *format, ...);
int fscanf(FILE *stream, const char *format, ...);从文件流读取格式化数据
int sscanf(const char *str, const char *format, ...);从指定字符串读取格式化数据
int printf(const char *format, ...);写到这想起阿水了
int fprintf(FILE *stream, const char *format, ...);将按指定格式输出内容到文件流stream
int sprintf(char *str, const char *format, ...);将按指定格式输出内容到字符串str
一个fprintf的例子：fprintf(fp, "%s\n", str);

//6.流的重定位 控制文件流中的指针位置
int fseek(FILE *stream, long int offset, int whence);和lseek基本一致
long ftell(FILE *stream);获取文件指针当前位置
void rewind(FILE *stream);将文件指针移动到文件开头

//7.将缓冲区的数据立刻写入文件
int fflush(FILE *stream);

//8.确定流使用的底层fd（*）
int fileno(FILE *fp);

//9.根据打开的文件描述符创建流（*）
FILE *fdopen(int fildes, const char *mode);mode表示打开方式
```

其它系统调用

```
//1.获取文件信息
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
int stat(const char *filename, struct stat *buf);filename的信息存到buf指向的内存
int fstat(int filedes, struct stat *buf);文件描述符对应文件信息存到……
int lstat(const char *file_name, struct stat *buf);返回的是链接本身而不是指向文件信息
```

```
//2.struct stat 标红蓝的重点属性如下
struct stat {
   mode_t st_mode; /*file type & mode 权限和文件类型(Linux 系统中的文件类型)*/
   ino_t st_ino; /*inode*/
   nlink_t st_nlink; /*link count 硬链接计数*/
   uid_t st_uid; /*user ID of owner*/
   gid_t st_gid; /*group ID of owner*/
   time_t st_atime; /*上一次访问时间*/
   time_t st_mtime; /*上一次修改时间*/
   time_t st_ctime; /*上一次status修改时间*/
};
判断文件是否是目录?
    struct stat sb;
    const char* filename;
    stat(filename,stat);
    if(S_ISDIR(stat.st_mode)){
    .....
    }不用背 只是举例
这些函数怎么来的?
    S_ISREG()
    S_ISDIR()
    S_ISCHAR()
    S_ISBLK()
    S_ISFIFO()
    S_ISLINK()
    S_ISSOCK()
    分别代表了七种文件类型

//3.文件权限拓展
SUID权限：将用户提升到root权限（设置了SUID的程序文件，用户执行时，用户的权限是该程序文件属主的权限）
SGID权限：将用户提升到组权限

//4.access测试存取权限
int access(const char *pathname, int mode);mode指定了需要检查的权限，如F_OK代表是否存在

//5.chmod fchmod改变文件权限
int chmod(const char *path, mode_t mode);
int fchmod(int fildes, mode_t mode);f开头的变量输入文件描述符

//6.chown fchown lchown更改拥有者
int chown(const char *path, uid_t owner, gid_t group);将指定路径的所有者和组进行更改
int fchown(int fd, uid_t owner, gid_t group);文件描述符
int lchown(const char *path, uid_t owner, gid_t group);不跟随符号链接

//7.umask
mode_t umask(mode_t mask);
umask(S_IWGRP | S_IWOTH);
上述代码将权限掩码中的“组用户可写”和“其他用户可写”两个位设为 1，表示在创建新文件或目录时不会赋予这两个权限，而其他权限则会被保留。

//8.软硬链接
//硬链接
int link(const char *oldpath, const char *newpath);
int unlink(const char *pathname);删除指定路径的链接
//软链接
int symlink(const char *oldpath, const char *newpath);
```

```
int readlink(const char *path, char *buf, size_t bufsiz);
path代表符号链接的路径名 buf代表存储符号链接目标路径名的缓冲区 bufsiz代表缓冲区大小

//9.目录操作（*）
int mkdir(const char *pathname, mode_t mode);
int rmdir(const char *pathname);
int chdir(const char *path);
int fchdir(int fd);
char *getcwd(char *buf, size_t size);cwd代表绝对路径
接下来的操作需要引入dirent.h
DIR *opendir(const char *name);
int closedir(DIR *dir);
struct dirent *readdir(DIR *dir);
off_t telldir(DIR *dir);// 查看当前目录项的偏移量
void seekdir(DIR *dir, off_t offset); // 跳转至下一个目录项，可指定偏移量

//10.目录扫描程序（*）
DIR *dp;
struct dirent *entry;
if ( (dp = opendir(dir)) == NULL )
   err_sys(…);
while ( (entry = readdir(dp)) != NULL ) {
   lstat(entry->d_name, &statbuf);
   if ( S_ISDIR(statbuf.st_mode) )
   …
   else
   …
}
closedir(dp);
```

二、文件锁（#）

在多个进程同时操作一个文件时可以上锁

文件锁分类

记录锁：将某个文件进行锁定，防止其它事务进程或删除

劝告锁：不会实际锁定数据，而是向其它进程发出建议

强制锁：由操作系统提供支持，防止其它进程访问

共享锁：允许一起读，不允许它们修改或删除

排他锁：只允许一个进程进行修改/删除

大胆一点，文件锁函数不考，考的话太傻逼了

ch4-2 Kernel Driver

一、内核相关概念

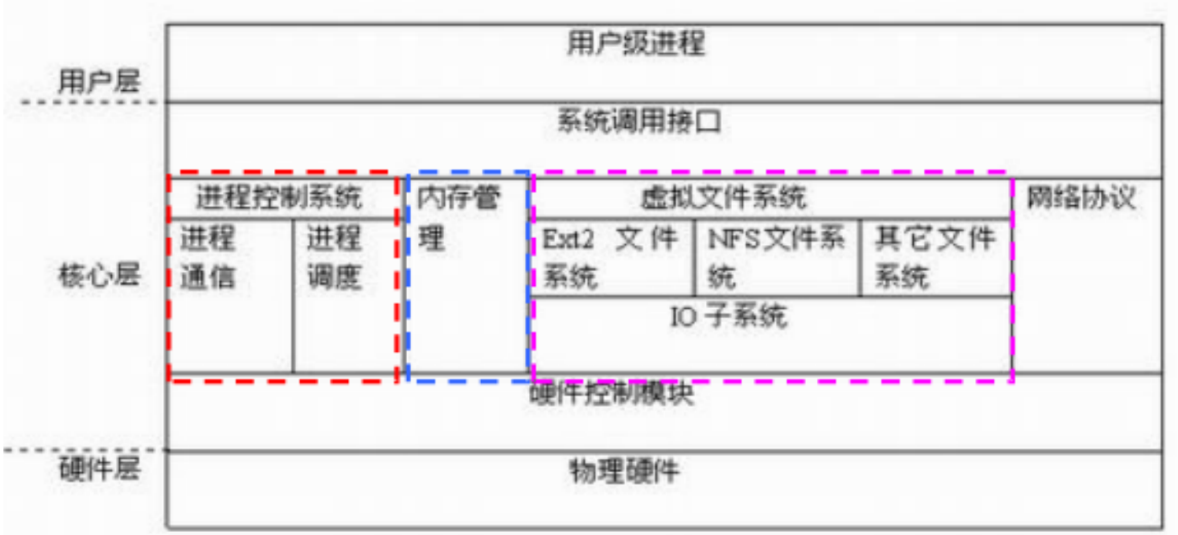
内核的定义：内核就是操作系统的核心组成部分。

单内核与微内核：单内核是大进程，分为若干模块，模块间通讯通过调用函数；微内核是独立进程，通过消息传递通讯。

Linux内核的能力：内存管理、文件系统、进程管理、多线程、抢占式、多处理。

Linux内核的优点：单内核、开源、兼容性好

二、内核层次结构



三、驱动和模块

驱动

1. 许多常见驱动的源代码集成在内核源码里
2. 也有第三方开发的驱动，可以单独编译成模块.ko
3. 编译需要内核头文件的支持

模块

模块依赖

1. 一个**模块A引用另一个模块B**所导出的符号，我们就说**模块B被模块A引用**
2. **如果要装载模块A，必须先要装载模块B。**这种模块间的相互关系就叫做**模块依赖**
3. moddep：显示内核模块的依赖关系
4. lsmod：列出已经加载的内核模块
5. modinfo：显示内核模块的信息

模块通讯

为了丰富系统的功能，所以模块之间常常进行通信。其之间可以共享变量，数据结构，也可以调用对方提供的功能函数

四、内核和应用程序的区别

1. 不可以使用C库来开发驱动程序
2. 没有内存保护机制
3. 小内核栈
4. 并发上的考虑

ch5G 华为给我增智慧

1.2019年底，EulerOS开源，成为OpenEuler。

2.是专用系统吗？不，是通用服务器操作系统。

3.支持多种处理器架构。

4.指令集是ARMv8-64

5.对鲲鹏处理器做了优化

6.第一个PPT 第一题选C 第二题ABCD 第二个PPT 选F

7.OpenEuler使用多队列调度策略

8.知道有numa aware这个东西存在，能够减少开销

9.isulad是一种容器，支持gRPC/RESTFUL两种通信方式

10.有智能优化引擎A-Tune，智能决策、自动调优

11.有内存管理预测机制

12.有内存共享机制

13.使用精简指令集

14.没有神经网络加速引擎

15.如果问四点优势：多核调度、软硬件协同、isulad、atune。正好一条一分。