2023/5/12

Linux知识点整理

---

1 终端和主机

终端：主机的输入和输出设备

主机：程序和数据的存储及处理

2 行律与驱动程序

驱动程序：不同的硬件需要不同的驱动程序

行律：进行一行内字符的加工（缓冲、回显与编辑）

3 流量控制

主机与终端之间进行流量控制：

- 硬件方式：RS232接口的CTS信号线
- 软件方式：`Ctrl+S`暂停，`Ctrl+Q`恢复

4 一些基础命令

*Linux*对大小写敏感

- 创建新用户（root用户执行）：`useradd`，用户信息存放在`etc/passwd`文件中
- `man`手册，`q`退出，`空格`下一页，`上下箭头`上移下移
    - `man name`
    - `man section name`
    - `man -k regexp`，正则表达式`regexp`
- `date`时间，定制格式`date "+%Y-%m-%d %H:%M:%S Day %j"`（`j`表示是今天的第几天）以及`date "+%s"`
- `cal`打印日历
    - `cal year`
    - `cal month year`
- 计算机`bc`，指定精度`bc -l`（小数点后20位）
- 改密码`passwd`
- 谁在系统中`who`，相关的：
    - `whoami`
    - `who am i`
    - `tty`打印出当前终端的设备文件名
- `uptime`：系统自启动后到现在的运行时间+当前登入系统的用户总数+近期1分钟/5分钟/15分钟内CPU负载
- `top`：列出资源占用考前的进程
    - VIRT：进程逻辑地址空间大小
    - RES：驻留内存数，即占用的物理内存数
    - SHR：与其他进程共享的内存数
    - %CPU：占用CPU的百分比
    - %MEM：占用内存的百分比
    - TIME+：占用CPU的时间
- `ps`：就是peocess state，选项包括：
    - 无选项：只列出当前终端上的进程，包括PID/TTY/TIME/COMAND

    - `e`：列出系统所有进程
    - `f`：full格式
    - `l`：long格式

  `ps`列出的内容包括：

    - 用户UID
    - 进程PID
    - CPU占用指数C（最近一段时间进程占用CPU情况）
    - 启动时间STIME
    - 进程逻辑内存大小SZ
    - 终端名TTY
    - 命令COMMAND
    - 进程在内存的何处休眠WCHAN
    - 累计执行时间TIME（占用CPU时间）
    - 优先级PRI
    - 状态S，sleep/run/zombie

- `free`：了解内存使用情况？？？？？？？？？？？？？？？？？？？？？？
- `vmstat`：？？？？？？？？？？？？？？？？？？？？？？？？？？？？

5 文本处理

- `more/less`：逐屏显示文件，`more shudu.c`，`ls -l |less`，对于`more`：
    - 空格：显示下一屏
    - 回车：上滚一行
    - `q`：退出程序
    - `/pattern`：搜索指定模式的字符串
    - `h`：帮助信息
    - `Ctrl+L`：屏幕刷新

  对于`less`：功能更强

- `cat`：列出文件内容，文本格式打印
    - `cat > try`：从标准输入获取数据，输入到`try`文件中
- `od`：逐字节打印，选项：
    - `-t`：十六进制打印各字节
    - `-c`：逐字符打印，遇到不可打印字符打印编码
- `head -n 10`：显示文件的前`10`行（`head -n -20`：去除文件尾部20行以外都算作head）
- `tail -n 10`：显示文件的尾`20`行（`tail -n +20`：去除文件头部20行以外都算作tail）
  `tail -f`：实时打印文件尾部被追加的内容
- 一个样例：`ls -s | sort | head -n 20`
- `tee`：将从标准输入得到的数据抄送到标准输出显示，同时存入磁盘文件中，`./myap | tee myap.log`
- `wc`：字计数，列出文件中一共有多少行，有多少个单词，多少字符（当指定文件数量大于一时，还给出一个总计）。选项`-l`只列出行计数。`wc sum.c`，`wc -l *.c makefile start.sh`
- `sort`：对文件内容排序（按照字符串比较规则），选项`-n`表示对于数字按照算数值大小排序
- `tr`：翻译字符，`tr string1 string2`，`string1`中出现的字符替换为`string2`中对应字符，例如：
  `cat report | tr '[a-z]' '[A-Z]'`，`cat file1 | tr % '\012'`（百分号改为换行符）
- `uniq`：筛选文件中重复的行，`uniq options`，`uniq options input-file`，`uniq options input-file output-file`，重复的行定义为紧邻的两行内容相同，选项：
    - `-u`：只保留没有重复的行
    - `-d`：只保留有重复的行（但只打印一次）
    - 没有上面两个选项，打印没有重复的行和有重复的行（只打印一次）
    - `-c`：计数同样的行出现几次

6 正则表达式

正则表达式：

- 应用范围：字符串匹配操作和替换操作。举例：Linux中的vi/more/grep/yacc/lex/awk/sed等
- 功能：描述一个字符串的模式
- 注意与文件名通配符区分：
    - 正则表达式规则用于文本处理场合
    - 文件名匹配规则用于文件处理场合
- 正则表达式的元字符：
    - `.`：匹配任意单字符
    - `*`：单字符后接`*`，表示这个单字符零次或多次出现（这个单字符可以是空格！）
    - `[`：`[]`中为集合内容，必须与其中的某一个匹配。`.*\`这三个在中括号内表示自己。`[][]`匹配`][`这两个字符之一
    - `\`
    - `^`：在中括号内开头（不在开头则失去补集意义）表示补集，`[^a-z]`匹配任意非小写字母。`[^][]`表示匹配任意非中括号字符。在中括号外时，仅在首部有意义，否则与其自身匹配
    - `$`：在尾部有特殊意义，否则与其自身匹配
- 正则表达式的扩展？？？？？？？？？？？？？？？？？？？

7 行筛选grep

基本用法：`grep 模式 文件名列表`，举例：

- `grep O_RDWR *.h`
- `ps -ef | grep liang`
- `ls -l / | hrep '^d' | wc -l`
- `grep '[0-9]*' shudu.c`

- `grep '[0-9][0-9]*' shudu.c`

`grep`选项：

- `-F`：按字符串搜索（而不是正则表达式）
- `-G`：基础正则表达式
- `-E`：扩展正则表达式
- `-P`：按PCRE搜索
- `-n`：显示时在每行前面显示行号
- `-v`：显示所有不包含模式的行
- `-i`：字母比较时忽略字母的大小写

8 流编辑sed

基本用法：

- `sed '命令' 文件列表`
- `sed -e '命令1' -e '命令2' -e '命令3' 文件名列表`
- `sed -f 命令文件 文件名列表`

举例：

- `tail -f pppd.log | sed 's/145\.137\.23\26/桥西/g'`
- `cat pm.txt | sed 's/\[[^][]*]//g'`：？？？？？？？？？？？？？？？？

`sed`中的正则表达式替换：可以在模式中添加`\(`和`\)`不影响匹配操作，但可以进行替换操作，例如：`s/\([0-9][0-9]\)-\([0-9][0-9]\)-\([0-9][0-9]*\)/\3.\1.\2/g`这条命令将日期格式“月-日-年”改为“年-月-日”。

9 文本加工awk

基本用法：

- `awk '程序' 文件名列表`
- `awk -f 程序文件名 文件名列表`
- 处理逻辑：对于符合条件的行，执行相应的动作

`awk`中条件描述方法：

- 使用C语言类似的关系算符，如`<`等
- 使用C语言类似的逻辑算符，如`&&`等
- 正则表达式模式匹配：包含该模式的行视为满足条件
- 特殊条件：
    - 不指定任何条件，对每一行执行动作
    - `BEGIN`：开始处理所有文本行之前执行动作
    - `END`：处理完所有文本行之后执行动作

`awk`中动作描述方法：

- 自定义变量
- 加减乘除等算数逻辑运算
- 正则表达式匹配运算符（用作条件判断）
- 流程控制（与C语言类似）：条件判断`if`，循环控制`for`
- `print 变量1, 变量2, ...`
- `printf("格式串", 变量1, 变量2, ,,,)`

`awk`举例：

- `ps -ef | awk '/guest/{ printf("%s ", $2); }'`
- `awk '{printf("%d: %s\n", NR, $0); }' test.c`
- `date | awk '{print $4}'`
- `ls -s | awk '$1 > 2000 { print $2 }'`

10 文件内容比对

`cmp`命令用于逐字节比较两个文件是否完全相同：

- 完全相同时不给出任何提示
- 不同时，打印出第一个不同之处

`md5sum/sha1sum`也可用于文件内容比较，MD5算法根据文件内容生成16字节的hash值，比较hash值是否相同，就可断定两文件内容是否完全相同。

11 文本文件差异

`diff`命令用于求出两文件的差别，基本用法：

- `diff file1 file2`：normal格式输出
- `diff -u file1 file2`：unified格式输出

normal格式：

- `25c25,26`：原文件的第25行更换成新文件的第25-26行
- `61,62a60`：原文件的第61到62行删除后，后面的内容与新文件的第60行之后一致
- `68a67,68`：原文件的第68行增加新文件中的第67到68行

- 输出时，`>`后边的内容是需要在file1中增加的内容，`<`后边的内容是需要从file1中删除的内容

12 vi相关命令

常见命令如下：

- `set number`：每行左边显示行号
- `set tabstop=4`：制表符位置为4格对齐
- `:set`：运行时检查偏好设置
- `i`：在当前字符前插入正文段
- `a`：在当前字符后插入正文段
- `h`：光标左移一列
- `j`：光标下移一行
- `k`：光标上移一行
- `l`：光标右移一列
- `5h`：光标左移5列，类似地，上面四个都可以在前面加上数字
- `Ctrl+b`：向后翻页，可用`PgUp`键代替
- `Ctrl+f`：向前翻页，可用`PgDn`键代替
- 两个翻页指令同样支持数字，例如`6Ctrl+f`（向前翻6页）
- `^`：将光标移至当前行首
- `$`：将光标移至当前行尾
- `w`：移到右一个单词
- `b`：移到左一个单词（这两个命令同样可以接数字）
- `:`可用于移到指定的行，例如`:1`移到文件第一行，`:$`将光标移到文件尾
- 在描述行号时，`.`可以代表当前行号，`$`可以代表最后一行的行号
- `%`：把光标移到一个花括号（或圆括号、方括号）上，按下`%`键，则光标自动定位到与它配对的那一个括号

- `x`：删除当前字符，`5x`删除从当前光标开始的5个字符
- `dd`：删除当前行，`3dd`删除从当前行开始的3行
- `r`：用于字符替换，`ra`将当前光标所指的字符替换为a，`rarbrc`将当前光标开始的三个字符替换为abc
- `u`：取消上一次的编辑操作
- `.`：重复上一次的编辑操作，例如上一次`3dd`删除了3行，则按`.`便会再删除3行
- `zz`和`:wq`：存盘退出
- `:w`：存盘不退出
- `:q!`：不存盘退出
- `:r xyz.c`：读入文件`xyz.c`插入到当前行之下
- `:50,$w file1`：写文件，把第50行至文件尾的内容写到文件file1中（如果`$w!`则为强制覆盖）
- `:10,50d`：删除第10至50行，并拷贝到剪切板
- `:1,.d`：删除文件首至当前行的部分，并拷贝到剪切板
- `:10,50y`：不删除，直接将第10行到50行的内容拷贝到剪切板
- `p`：粘贴剪切板信息
- `:5,10co56`：复制第5到10行到第56行之下
- `:8,34m78`：移动第8到34行到第78行之下
- `J`：两行合并（下面的行合并到当前行）
- `Ctrl+l`：刷新屏幕显示
- `Ctrl+g`：在屏幕最下面一行列出正在编辑的文件的名字、总行数、当前行号、文件是否被修改过等信息
- `/pattern`：正则表达式查找（是循环式搜索），例如`/[0-9][0-9]*`，`n`表示向下查找下一个next，`N`表示向上查找
- `:n1,n2,s/pattern/string/g`：替换命令，例如：
    - `:1,$s/ $//g`消除尾部多余的空格
    - `:1,$s/a\[i]\*b\[j]/x[k]*y[n]/g`将`a[i]*b[j]`替换为`x[k]*y[n]`命令

    - 在模式替换的时候同样可以增加`\(`和`\)`，增强灵活性，例如“将`变量名->number`替换为`变量名->num`”功能由该命令实现：`:1$s/\([a-zA-Z_][a-zA-Z0-9_]*\)->number/\1->num/g`
- `Ctrl+Z`：将当前进程挂起，可以通过`jobs`列出当前被`stopped`的进程有哪些，然后通过`fg %1`将`1`号作业恢复到前台执行

13 中文编码

系统编码方案：

- Windows默认中文编码方案为`GBK`，两个字节表示一个汉字，字节的高位为1，以区别于ASCII码
- 许多Linux默认中文编码方案为`UTF8`，三个字节表示一个汉字，字节的高位为1，以区别于ASCII码

命令`iconv`可以替换中文字符编码：

- `iconv -f gbk -t ytf8`：从`GBK`到`UTF8`
- `iconv -f utf8 -t gbk`：从`UTF8`到`GBK`

14 文件与文件夹

文件名命名规则如下：

- 名字长度允许1-255字符，有些UNIX不支持长文件名（但长度至少为1-14）
- 取名合法字符：除斜线之外的所有字符都是合法命名字符，不可打印字符也可以用做文件名，斜线留作路径分隔符
- 大小写敏感，例如`makefile`和`MakeFile`是不同的（但是在windows下是不敏感的，因此尽量不要利用大小写区分文件，不利于文件迁移）

系统中的目录：

- `/etc`：主要是系统配置文件，例如`passwd`文件、`hosts`文件等
- `/tmp`：临时文件，每个用户都可以在此创建文件，但只能删除自己的文件
- `/var`：系统运行时需要改变的数据，例如系统日志`syslog`等
- `/bin`：系统常用命令，如`ls`、`ln`、`cp`、`cat`等命令
- `/usr/bin`：存放一些常用命令，如`ssh`、`ftp`、`make`、`gcc`、`git`等
- `/sbin,/usr/sbin`：系统管理员专用命令
- `/dev`：设备文件
- `/usr/include`：C语言头文件存放目录
- `/lib,/usr/lib`：存放各种链接库文件，指C语言的链接库文件，以及`terminfo`终端库等。静态链接库文件有`.a`后缀，动态链接库文件有`.so`后缀

15 文件通配符

规则如下：

- `*`：匹配任意长度的文件名字符串，但是`.`（为第一个字符时）和`/`必须显示匹配，比如`*file`匹配`file`文件，但是不匹配`.profile`文件
- `?`：匹配任意单字符
- `[]`：匹配括号内任一字符，也可以用`-`指定一个范围
- `~`（bash特有的）：`~`当前用户的主目录，`~kuan`用户kuan的主目录
- `.`文件与`..`文件：当前目录与上级目录

注意：

- 文件名通配符与正则表达式规则不同，应用场合不同

- 不同种类的shell通配符规则会略有差别
- windows中`*.*`匹配所有文件，而linux中`*.*`要求文件名中必须含有圆点

16 shell与kernel

shell是一个用户进程，如`/bin/bash`，对用户提供命令行界面，可以在此启动其他的应用程序（ap）使用操作系核心提供的功能：包括系统命令和用户编写的程序。shell处理文件名通配符，例如将文件通配符展开，然后才会执行命令。

kernel是操作系统的核心，负责管理系统资源（包括内存、磁盘等），运行在核心态，通过软中断方式对用户态进程提供系统调用接口。

17 文件目录

`ls`命令的常见选项：

- `-F`选项（`Flag`）：
    - 若列出的是目录，就在名字后面加上`/`
    - 若列出的是可执行文件，就在名字后面加上`*`
    - 若列出的是符号链接文件，就在名字后面加上`@`
    - 若列出的普通文件，则名字后面没有任何标记
- `-l`选项：
    - 第一列：文件属性（`-`普通文件，`b`块设备文件，`d`目录文件，`c`字符设备文件，`l`符号链接文件，`p`命名管道命名文件）和文件的访问权限
    - 第二列：文件的link数
    - 第三列和第四列：文件主的名字和组名
    - 第五列：按照文件类型
        - 普通磁盘文件：列出文件大小

        - 目录：列出目录表大小，不是目录下文件长度之和
        - 符号链接文件：列出符号链接文件自身的长度
        - 字符设备和块设备文件：列出主设备号和次设备号
        - 管道文件：列出管道内的数据长度
    - 第六列：文件最后一次被修改的日期和实际按
    - 第七列：文件名，对于符号链接文件，附带出符号链接文件的内容
- `-h`选项：以人类以读的方式打印信息
- `-d`选项：当`ls`的参数是目录时，不像默认情况那样列出目录下的文件，而是列出目录自身的信息
- `-a`选项：列出文件名首字符为圆点的文件（默认情况下不会列出）
- `-A`选项：功能与`-a`相同，但不列出`.`和`..`文件
- `-s`选项：列出文件占用的磁盘空间
- `-i`：列出文件的`i`节点号

18 文件复制与删除

`cp`命令用于拷贝文件，格式如下：

- `cp file1 file2`：`file2`不存在则创建；`file2`存在且是文件则覆盖；`file2`存在且是目录则按下一种形式进行处理
- `cp file1 file2 ... filen dir`：`dir`必须是已经存在的一个目录

`mv`命令用于移动文件（或者给文件或目录改名），格式如下：

- `mv file1 file2`
- `mv file1 file2 ... filen dir`
- `mv dir1 dir2`

`rm`命令用于删除文件，格式如下为`rm file1 file2 ... filen`，选项为：

- `-r`：递归地进行删除，参数为一个目录
- `-i`：每删除一个文件前需要操作员确认
- `-f`：强迫删除（包括只读文件也被删除且没有提示）
- 注意：正在运行的可执行程序文件不能被删除

当文件名中含有`-`时候，我们可以使用`--`来显式地结束选项输入，在`--`之后输入的字符均不被解释为选项。

19 目录相关命令

常用命令如下：

- `pwd`：打印当前工作目录
- `cd`：改变当前工作目录（注意，`cd`是shell的一个内部命令）
- `mkdir`：创建一个目录（选项`-p`自动创建路径中不存在的目录）
- `rmdir`：删除一个目录（要求被删除的目录中除了`.`和`..`以外没有其他文件或目录）
- `cp -r`：递归地复制一个目录，如果`dir2`存在则在`dir2`下新建子目录`dir1`并将内容拷贝到其中；如果`dir2`不存在则创建，然后拷贝`dir1`中内容至其中。选项：
    - `-v`：可以显示在复制目录时正在复制的文件的名字
    - `-u`：增量拷贝，根据文件的时间戳，不拷贝相同的或者过时版本的文件，此时`dir1`和`dir2`位置颠倒不会导致灾难性后果
- `find`：遍历目录树，该命令从指定的查找范围开始，递归地查找子目录，凡是满足条件的文件或目录都执行相关的动作，例如`find ver1.d ver2.d -name '*.c' -print`在当前目录的两个子目录`ver1.d`和`ver2.d`中查找与`*.c`匹配的文件名，并打印出响应文件的路径名。`find`命令关于条件的选项：
    - `-name wildname`：文件名（仅指路径名的最后一部分）需要与`wildname`相匹配

- `-regex pattern`：整个路径名与正则表达式`pattern`匹配
- `-type f/d/l/c/b/p`：类型检查（普通文件/目录/符号链接文件/字符设备文件/块设备文件/管道文件）
- `-size +/-n单位`：指定文件大小大于（`+`）、等于（`空格`）或小于（`-`）某大小，单位有：字符`c`、512字节的块（`b`）、`k`、`M`、`G`，默认为`b`
- `-time +/-ndays`：文件最近修改时间
- `-newer file`：文件最近修改时间比`file`文件还要晚
- `-inum`：指定`i`节点号
- `-user, -nouser`：指定文件主
- `-group, -nogroup`：指定用户组
- `-links`：指定link数
- `-depth`：指定路径访问深度
- `-perm`：指定文件的访问权限
- 支持复合条件，即可以使用`()`、`-o`、`!`等表示多条件的与、或、非

`find`命令关于动作的选项：

- `-print`：打印查找文件的路径名
- `-exec`：对查找到的目标执行某一命令，在`-exec`及随后的分号之间的内容作为一条命令（在这条命令的命令参数中，`{}`代表遍历到的目标文件的路径名
- `-ok`：与`-exec`类似，只是对查找到的木盒条件的目标执行一个命令之前需要经过操作员的确认

`find`命令的例子：

- `find . ! -type d -links +2 -print`：从当前目录开始检索link数大于2的非（注意“非”`!`）目录文件
- `find ~ -size +100k \( -name core -o -name '*.tmp' \) -print`：从当前用户主目录开始查找大小大于100kb且名字为`core`或者满足`*.tmp`描述符的文件
- `find /lib /usr -name 'libc*.so' -exec ls -lh {} \;`：在`/lib`和`/usr`两个目录下查找名字满足`libc*.so`的文件执行`ls -lh 路径名`操作

- `find ~ -size +100k \( -name core -o -name '*.tmp' \) -ok rm {}\;`：执行删除操作之前需要用户确认
- `find src -name \*.c -exec grep -n -- --help {} /dev/null \;`：在目录`src`下查找所有`.c`文件中查找`--help`字符串。`-n`表示显示行号，`/dev/null`代表什么？？？？？？？？？？？

20 打包与压缩

命令为：`tar ctxv[f devicel]] file-list`，选项第一字母指定要执行的操作，`c`表示创建新磁带，`t`表示磁带上的文件列表（若不存在则创建），`x`表示从磁带中抽取指定文件（不存在时抽取所有文件），其他选项：

- `v`：每处理一个文件，就打印出文件名并在该名之前冠以功能字母
- `f`：指定设备文件名
- `z`：采用压缩格式（`gzip`算法）
- `j`：采用压缩格式（`bzip2`算法）

使用案例：

- `tar cvf /dev/rct0 .`：将当前目录树备份到设备`/dev/rct0`中
- `tar tvf /dev/rct0`：查看磁带设备`/dev/rct0`上的文件目录
- `tar xvf /dev/rct0`：将磁带设备`/dev/rct0`上的文件恢复到文件系统中
- `tar cvf my.tar *.[ch] makefile`：指定普通文件代替设备文件，将多个文件或目录树存储成一个文件
- `tar cvf work.tar work`：压缩文件，其中`work`是一个有多个层次的子目录
- `tar cvzf work.tar.gz work`：`gzip`压缩格式
- `tar cvjf work.tar.bz2 work`：`bzip2`压缩格式
- `tar xvf work.tar.gz`：查看归档文件中的文件目录
- `tar xvf work.gz`：从归档文件中恢复目录树

21 命令获取信息

在Linux系统命令和用户程序（ap）运行过程中，有下面几种方式获得信息：

- 配置文件：系统级偏好设置和用户级偏好设置，例如`/etc/profile`和`~/.bash_profile`等
- 环境变量：命令`env`可以打印出当前的环境变量，C语言可以通过库`getenv()`获取环境变量
- 命令行参数：程序启动之前通过命令行参数输入参数
- 交互式键盘输入（在linux中较少使用）：程序运行过程中（暂停）由操作员输入

命令行参数的三种风格：

- 类似`dd`命令的风格，采用`param=value`格式
- 类似`find`和`gcc`命令的风格，以减号打头的一个由多个字符构成的单词用作选项
- 类似`ls`和`grep`命令的风格（如今流行的格式），长选项与短选项交叉，并以`--`标识选项的结束

22 文件系统

文件系统的三个主要概念：

- 根文件系统
- 子文件系统
- 独立的存储系统

常见命令：

- `mkfs /dev/sdb`：新建文件系统
- `mount /dev/sdb /mnt`：安装子文件系统
- `mount`（不带参数）：列出当前已安装好的所有子文件系统
- `umount /dev/sdb`：卸载一个文件系统
- `df`：显示文件系统的空闲空间（`-h`选项可选）

文件系统的结构：

- 整个逻辑设备以块（扇区）为单位进行划分，编号从0开始，每块大小都是512字节（也有可能更大），其中：
    - 0号块：引导块，用于启动系统，只有根文件系统的引导块有效
    - 1号块：也叫管理块，或则超级块，用于存放文件系统的管理信息，如文件系统的大小、`i`节点区的大小、空闲空间的大小、空闲块链表头等信息。执行`mkfs`命令进行初始化时，`df`命令读出部分信息
- `i`节点区：由若干块构成，在`mkfs`命令创建文件系统时确定，每块可以容纳若干个`i`节点，每个`i`节点的大小是固定的（例如64字节）。`i`节点从0开始编号，根据编号可以索引到磁盘跨块。每个文件都对应一个`i`节点，`i`节点中的信息包括：
    - 指向文件存储区数据块的一些索引指针
    - 文件属性，属主、组、权限、link数、大小、时间戳等
- 文件存储区：用于存放文件数据的区域，包括目录表
- 目录表：linux系统允许带有交叉勾连的目录结构，每个目录表也作为一个文件来进行管理，每个目录表由若干个目录项构成，目录项只含两部分信息：文件名，`i`节点号。用`ls`命令列出的目录大小是目录表文件本身的长度。目录表和`i`节点的两级结构的目的主要是为了提高目录索引的检索效率。可以通过`stat`命令来读取`i`节点的信息

23 硬链接与符号链接

硬链接数目就是link数。根据文件目录的存储结构，可以在同一目录或者不同目录中的两个目录项有相同的`i`节点号，每个目录项指定的`文件名-i节点号`映射关系，叫做1个硬链接，link数就是同一`i`节点被目录项引用的次数。

`ln chapt0 intro`：创建源文件为`chapt0`的硬链接文件`intro`，这时如果使用`ls -l chapt0 intro`命令列出两个文件的相关信息，则前面几项必定相同（因为来自同一个`i`节点文件），此时删除其中一个文件只会导致link数减一，不会删除文件（此时`chapt0`文件和`intro`文件的地位完全相同）（硬链接仅限于同一文件系统中的普通文件）。

不允许对目录建立硬链接文件，目录文件的link数一般为直属子目录数+2。

符号链接通过符号链接文件实现，该文件内容仅包括一个路径名。建立符号链接文件之后，删除操作删除的是符号链接文件，而其他所有操作都是直接访问真正的文件。符号链接中：

- 相对路径：是相对于符号链接文件的位置
- 绝对路径：引用绝对路径名，移动符号链接文件之后不会失效

硬链接与符号链接文件的比较：

- 硬链接仅在数据结构层次上实现，并且只适用于文件，不适用于目录，不同文件系统之间也不行
- 符号链接在算法软件层面实现，硬链接能完成的功能都可以通过符号链接来实现，适用于目录、也适用于不同的文件系统，同硬链接相比要占用操作系统内核的一部分开销

24 系统调用

系统调用以C语言函数调用的方式提供，系统调用是应用程序（ap）给操作系统（kernel）进行交互的唯一手段，例如文件的`open`、`write`、`read`、`close`等操作。系统调用特点如下：

- 系统调用以中断方式进行，与函数执行过程不同
- 可以通过库函数方式对系统调用进行封装，例如`printf`是对`write`的封装
- 系统调用和相关的api函数以及库函数的名称是posix标准的一部分，便于在不同的unix系统之间进行迁移

系统调用的返回值：

- 大于等于0：成功
- 等于-1：失败
- 整型变量`errno`为失败原因记录，是个整数，用于标识错误原因，库函数`strerrno`可以将数字形式的错误代码转换成一个可以阅读的字符串
- `printf`函数中的`%m`会被替换成上一次系统调用失败的错误代码对应的消息

25 访问i节点和目录

命令`stat`或`fstat`可以从`i`节点获取文件的状态信息：

- `stat`得到指定路径名文件的`i`节点
- `fstat`得到已打开的文件的`i`节点
- 这两条命令将数据放入调用者提供的`stat`结构体中，`stat`结构体中包含很多信息，详见ppt

注意`i`节点结构体`stat`结构体中有关时间信息的三个属性：

- `a访问`：读、执行等操作（有些系统为了懒惰处理，不更新，但不早于`m`时间）
- `m修改`：文件内容修改，例如写文件
- `c改变`：`i`节点信息变化，例如写文件、修改权限/link数/文件主等

构造自己的目录访问工具是可能的。

26 文件与目录的权限

权限的三个级别：文件主（有且仅有一个）、同组用户、其他用户。

普通文件的权限：读、写、可执行。不可写文件也可能被删除。

两类可执行文件：

- 程序文件：二进制文件，为二进制的CPU指令集合，满足操作系统格式才能被加载执行
- 脚本文件：文本文件，需要指定解释程序（例如`/bin/bash`）

目录的权限主要包括：

- 读权限：若无读权限，那么“目录表”不允许读，例如`ls`的命令的执行会失败
- 写权限：若无写权限，则创建新文件/删除已有文件的操作会失败（因为会改变目录表内容）。修改文件本身不需要修改目录表内容，因此是可行的
- 执行权限（`x`权限）：有执行权限意味着分析路径名过程中可以检索该目录，例如`cat /a/b/c`（`c`为文件）要求当前用户对`a`和`b`目录有`x`权限，对`c`文件有读权限

- `STICKY`权限：早期unix中，具有该权限的可执行文件尽量常驻内存或交换区以提高效率。现代unix中该属性被忽略。如果目录具有该属性，？？？？？？？？？？？

关于权限的常见命令：

- `ls -l`列出当前目录下所有文件和子目录的权限
- `chmod`可用于修改权限，有两种形式：
    - 字母形式：`chmod [ugoa][+-=][rwxst] 文件名列表`：`u`表示文件主权限、`g`表示同组用户权限、`o`表示其他用户权限、`a`表示上述所有人的权限
    - 数字形式：`chmod xxx 文件名列表`：`xxx`为三个八进制数字
- `umask`命令：控制文件和目录在创建时的初始权限，注意`umask`是进程属性的一部分。`umask 022`将`umask`值设为八进制的`022`，对应二进制为`000 010 010`，这时创建新文件将会取消新文件的同组用户`w`权限和其他用户的`w`权限

当前的三级权限体系有不足之处，主要体现在：系统中的一个用户，要么对文件的全部内容具有访问权限、要么完全不能访问该文件，这种缺点在修改密码等敏感信息时缺陷很大。为了解决该问题，有下面的思路：

- 用户liu有`list.txt`文件
- 用户liu希望用户liang只能读取行首为liang的数据
- 用户liu编写程序`query.c`实现对`list.txt`文件的遍历，并比较给定的名字与行首是否符合，若符合则输出信息
- 用户liu将`list.txt`文件权限设为`-rw-------`，将`query.c`文件编译后的文件`query`权限设为`-rw---x--x`
- 用户liang在试图执行`query`可执行文件的时候，会发现无法访问`list.txt`文件导致执行失败，无法进行查询
- 用户liu给`query`可执行文件增加`s`权限，此时`query`文件权限为`-rws--x--x`
- 这时，用户liang再次执行`query`文件的时候，便可以访问到`list.txt`文件了，可以进行查询操作了！

原理解释：一般情况下，进程的UID和有效UID相同，通过`open`打开文件的时候，系统会根据进程的有效UID与文件所有者UID进行比较并进行合法性验证。如果可执行文件具有SUID权限，那么在执行这个可执行文件的时候，进程的实际UID是当前用户，而进程的有效UID则变成了可执行文件的文件主！

27 shell基础知识

shell的主要用途是批处理，其执行效率比算法语言低。shell是面向命令处理的语言，提供的流程控制结构通过对一些内部命令的解释实现。shell提供了灵活的机制（策略与机制相分离），shell许多灵活的功能，通过shell替换来实现（例如流程控制所需的条件判断、四则运算，都由shell之外的命令完成）。

有三种方法启动交互式bash：

- 注册shell
- 键入bash命令
- 脚本解释器

自动执行的一批命令（用户偏好）：当bash作为注册shell被启动时，自动执行用户主目录下的`.bash_profile`文件中的命令（在`~`文件夹或`$HOME`文件夹下）；当bash作为注册shell退出时，自动执行`$.bash_logout`文件。类似`umask`之类的命令，应该写在`.profile`文件中。

自动执行的一批命令（系统级）：当bash作为注册bash被启动时，自动执行`/etc/profile`文件中命令；当bash作为交互式shell启动时，自动执行`/etc/bash.bashrc`；当bash作为注册shell退出时，自动执行`/etc/bash.bash.logout`。

脚本文件的执行可以通过新创建子进程来实现，几种命令如下：

- `bash<lsdir`：这种格式无法携带命令参数
- `bash -x lsidr`：可以携带命令参数
- `bash lsdir /usr/lib/gcc`：可以携带命令参数

这三种方法均启动程序`/bin/bash`并生成新进程。

如果想在当前shell进程中执行脚本：`. lsidr /usr/lib/gcc`或者`source lsdir /usr/lib/gcc`。

shell支持查看历史表，通过内部命令`history`（文件`$HOME/.bash_history`）来查看历史表，常用机制：

- 人机交互时可以使用上下箭头
- `!!`引用上一命令，`!str`引用以`str`开头的最近使用过的命令

可以给现有命令起别名，例如（如有需要，应把`alias`命令放入`./bashrc`文件）：

- `alias dir="ls -flad"`
- `alias n="netstat -p tcp -s | head -10"`

可以通过`alias`命令查看别名表，并且可以使用内部命令`unalias`命令取消别名（例如`unalias n`便是取消上面关于`n`的别名）。

可以使用TAB键补全。

常见的输入重定向：

- `<filename`：从文件`filename`中获取`stdin (fd=0)`，例如`sort < telno.txt`
- `<<word`：从shell脚本文件获取数据直到再次遇到定界符`word`，例如：

```
cat << TOAST
* NOW : `date`
* My Home DIrectory is $HOME
TOAST
# 定界符做界定内容的加工处理(等同双引号处理):
# 变量替换 命令替换
# 不执行文件名生成
```

- `<<<word`：从命令行获取信息作为标准输入，例如`base64 <<< beiyou`以及`base64 <<< 'bei you'`

程序的输出一般包括：

- 标准输出：`stdout`，`fd=1`
- 标准错误输出：`stderr`，`fd=2`

其中`stdout`重定向的常见方式包括：

- `>filename`：重定向到文件，若文件存在则先清空（覆盖方式）
- `>>filename`：将`stdout`追加到文件尾

而`stderr`重定向的常见方式包括：

- `2>filename`：重定向到文件，可以据此分离`stdout`和`strerr`的重定向
- `2>&1`：将文件句柄`2`（`fd=2`）重定向到文件描述符`1`指向的文件

除此之外，还允许对其他文件句柄输入或输出进行重定向，例如：

- `./mpap`
- `5 < a.txt`
- `6 > b.dat`

同时重定向`stdout`和`stderr`例如：`./stda 1>try.out 2>$1`（`1`可选，注意顺序）

管道是指迁移命令的输出作为下一命令的输入，例如：

- `ls -l | grep '^d'`
- `cc try.c -o try 2>$1 | more`

管道实际上可以认为将`stdout`和`stderr`都重定向到了一个PIPE文件中？？？

28 变量的赋值及使用

bash可以存储变量，存储的内容是字符串（注意：对于数字串来说，存储的是字符串而不是表示数值的二进制串），并且变量名必须遵守：

- 第一个字符必须是字母
- 其余字符可以是字母、数字、下划线

变量的赋值注意等号两侧不能有空格，并且如果有空格需要用双引号；变量的引用需要在前面冠以`$`符号，并且引用未定义变量则变量值位空字符串。

与这部分相关的shell内部开关包括：

- `set -u`：当引用一个未定义的变量时，产生一个错误
- `set +u`：当引用一个未定义的变量时，认为是一个空串（默认情况）
- `set -x`：执行命令前打印出shell替换后的命令及参数，为区别于正常的shell输出，前面冠以`+`符号
- `set +x`：取消上述设置

对于`echo`命令，注意以下两个案例的区别：

- `echo beijing china`
- `echo "beijing china"`

内部命令`read`是变量取值的另外一种方法，从标准输入输入一行内容赋值给变量。例如：读取用户的输入，并使用输入的信息：先输入`read name`，然后在命令行（`stdin`）上键入内容即。

29 环境变量

创建的shell变量都默认为全局变量（即只在当前shell中可以访问到），可以通过内部命令`export`将局部变量转换为环境变量，例如`export proto`，这条命令的意义在于：shell启动的子进程只能继承环境变量，而不继承局部变量（当然，此后子进程对环境变量的修改，不影响父进程中同名变量）。

登录系统后系统自动创建一些环境变量影响应用程序的运行，例如：

- `HOME`：用户主目录的路径名
- `PATH`：命令查找路径
- `TERM`：终端类型

相关命令包括：

- `set`：列出当前所有变量及其值以及函数定义（包括环境变量就局部变量、函数定义），例如`set | grep ^fname=`
- 外部命令`/bin/env`：列出环境变量及其值

环境变量可以供shell脚本和C程序的引用，例如：

- `cat Connect to $proto Networks`
- `cat report.c`（`report.c`文件中通过命令`char *proto = getenv("proto");`实现对环境变量`proto`的引用）

下面的案例演示了环境变量的使用（来自互联网）：

```
[net]$ a=22         #定义一个全局变量
[net]$ echo $a      #在当前Shell中输出a，成功
22
[net]$ bash         #进入Shell子进程
[net]$ echo $a      #在子进程中输出a，失败

[net]$ exit         #退出Shell子进程，返回上一级Shell
exit
[net]$ export a     #将a导出为环境变量
[net]$ bash         #重新进入Shell子进程
[net]$ echo $a      #在子进程中再次输出a，成功
22
[net]$ exit         #退出Shell子进程
exit
[net]$ exit         #退出父进程，结束整个Shell会话

# export a 这种形式是在定义变量 a 以后再将它导出为环境变量
# 如果想在定义的同时导出为环境变量，可以写作export a=22
```

30 shell的替换

shell一定是先进行替换工作、再执行命令，包括：

- 文件名生成：遵循文件通配符规则，按照字典序排列，例如`ls *.c`实际执行`ls a.c x.c`；如果没有匹配的文件则不展开，即仍然是`ls *.c`

- 变量替换：例如`ls $HOME`以及`echo "My home is $HOME, Terminal is $TERM"`
- 命令替换：反撇号内的命令先执行
- `$()`格式的替换：其中的`stdout`替换这部分内容，例如`./arg $(date)`实际上执行`./arg Sun Dec 4 14:54:38 Beijing 2018`

shell常见内部变量包括：

- `$0`：脚本文件本身的名字
- `$1`和`$2`等：1号命令行参数、2号命令行参数，等
- `$#`：命令行参数的个数
- `"$*"`：等同于`"$1 $2 $3 $4 ..."`
- `"$@"`：等同于`"$1" "$2" "$3" "$4" ...`

shell命令`shift`用于位置参数的移位操作，支持形如`shift 3`这种形式的命令，具体含义详见ppt。

31 shell的元字符

shell的元字符包括：

- 空格，制表符：命令行参数的分隔符
- 回车：执行键入的命令
- `>`、`<`、`|`：重定向与管道（还有`||`）
- `;`：用于一行内输入多个命令（还有`;;`）
- `&`：后台运行（还有`&&`）
- `$`：引用shell变量
- 反撇号：用于命令替换
- `*`、`[]`、`?`：文件通配符（`echo "*"`与`echo *`不同，前者？？？后者？？？）

- `\`：取消后继字符的特殊含义（转义）
- `()`：用于定义shell函数或在子shell中执行一组命令

重要知识：

- `\`用作转义，取消其后元字符的特殊作用（如果`\`加在非元字符后，那么就和没有一样）
    - 例如：`echo Windows Dir is C:\\Windows\\WORK.DIR`
- 单引号对其中任何字符不作任何特殊解释，全盘输出（直到再次遇到单引号）
- 双引号中只进行变量替换（`$`决定）和命令替换（反撇号决定）
    - 例如：`echo "My home dir is $HOME"`
- 重要案例：
    - `echo 'Don'\''t remove Peter'\''s Windows dir "C:\PETER"!'`
    - `echo "反撇号whoami反撇号's \$HOME is \"$HOME\""`
    - `year=反撇号expr \反撇号date '+%Y'\反撇号 - 10反撇号`（这条命令是多级嵌套的典例）
        - 先执行反撇号中的内容，为`expr 反撇号date '+%Y'反撇号 - 10`
        - 再执行这其中反撇号中的内容，为`date '+%Y'`
        - 得到当前年份为`2023`
        - 带入`expr 反撇号date '+%Y'反撇号 - 10`得到`expr 2023 - 10`
        - 得到十年前年份为`2013`
        - 带入`year=反撇号expr \反撇号date '+%Y'\反撇号 - 10反撇号`得到`year=2013`
    - 下面三者在实现效果上是一样的（在`*.conf`文件中找行尾是被单引号括起来的IP地址`192.168.x.x`的行，`grep`得到的第一个参数字符串应该为正则表达式`'192\.168\.[0-9]*'$`）：
        - `grep \''192\.168\.[0-9]*'\''$' *.conf`
        - `grep "'192\\.168\\.[0-9]*'\$" *.conf`
        - `grep \'192\\.168\\.\[0-9.]\*\'\$ *.conf`

- 给出程序名字，中止系统中正在运行的进程：

```
PIDS=`ps -e | awk '/[0-9]:[0-9][0-9]'$1'$/
{printf("%d ", $1);}'`
echo $PIDS
kill $PIDS
# 或者
kill `ps -e | awk '/[0-9]:[0-9][0-9]'$1'$/
{printf("%d ", $1);}'`
```

- 上面是否丢了一个空格？应当求证！！！

32 shell编程

shell中的逻辑判断基本内容如下：

- 每条命令执行结束之后有返回码，可以根据返回码判断命令是否执行成功（如`main`函数的`return 0`）
- shell内部命令`$?`表示上衣命令的返回码（用管道连接在一起的若干命令，进行条件判断时以最后一个命令执行的返回码为准）
- 可以用`&&`或`||`连结两个命令，注意短路计算原则
- 命令`true`（在`/bin/true`中）返回码总为`0`（同理有`false`命令）

`test`命令常用在shell条件判断中，有许多种用法：

- 文件特性检测，例如`test -r /etc/motd && echo readable`检测可读文件，其他选项有：普通文件`-f`、目录文件`-d`、可写文件`-w`、可执行文件`-x`、非空文件`-s`（为真条件是文件`size > 0`）
- 整数的比较，例如`test 反撇号ls | wc -l反撇号 -ge 100 && echo "Two many files"`，选项有：`-eq`、`-ne`、`-gt`、`-ge`、`-lt`、`-le`共六个

`[`命令也常用在shell条件判断中（要求最后一个参数必须是`]`，例如：

- 字符串比较，例如`[ "$a" = "" ] && echo empty string`（注意中括号两侧的空格不能少，否则`[`命令的参数检测不到）。检测不等号用`!=`
- 复合逻辑条件，选项包括：非`!`、或`-o`、与`-a`，用法例如：`[ ! -d $cmd -a -x $cmd ] && $cmd`，检测`$cmd`不是目录文件并且可以执行，则执行

在shell中，常见的命令组合方法如下所示：

- `{}`方式：在当前shell中执行一组命令
- `()`方式：在子shell中执行一组命令

```
# {}方式
pwd
DIR=/usr/bin
[ -d $DIR ] && {
    cd $DIR
    echo "Current Directory is `pwd`"
    echo "`ls | wc -l `files"
}
pwd

# ()方式
pwd
DIR=/usr/bin
[ -d $DIR ] $$ (
    cd $DIR
    echo "Current Directory is `pwd`"
    echo "`ls | wc -l `files"
)
pwd
```

注意这两种方式执行之区别：`{}`方式在当前shell中执行一组命令，那么当前shell的目录将会被切换，两个`pwd`显示的当前路径不同；而`()`中两次`pwd`命令显示的当前路径相同。除此之外，把多行合并为一行的时候，`{}`方式和`()`方式语法不同：

- `{}`方式：`[ -f core ] && { echo "rm core";rm core;}`（圆括号是shell元字符，而左花括号不是--必须作为一个特殊内部命令处理，所以必须有空格）
- `()`方式：`[ -f core] && (echo "rm core";rm core)`

在shell编程中，条件分支`if`基本语法如下：

```
if condition
    then list
elif confition
    then list
else
    list
fi
# 其中if/then/elif/else/fi为关键字(内部命令)
```

多条件分支`case`基本语法如下：

```
case word in
    pattern) list1;;
    pattern) list2;;
    ...
esac
```

注意：

- `word`与`pattern`匹配使用shell的文件名匹配规则
- `;;`是一个整体，不可修改
- 可以使用竖线表示多个模式：`pattern1|pattern2) list;;`
- `word`与多个`pattern`匹配时，执行遇到的第一个命令表

shell不支持除字符串以外的数据类型，也就是说不支持加减乘除等算术运算和关于字符串的正则表达式运算，这些功能通过`/usr/bin/expr`来实现，`expr`命令：

- 支持算属于水暖、关系运算、逻辑运算和正则表达式运算，符号包括：
  - `()`、`+`、`-`、`*`、`/`、`%`、`>`、`>=`、`<`、`<=`、`=`（单等号！）、`!=`、或运算`|`、与运算`&`、正则表达式运算`:`
- 注意空格的使用，例如：

```
x=`expr $a '*' '(' $b + $c')'`
y=`expr '(' $a + 4 '<' $b ')' '&' '(' $c != 8')'`
```

- 正则表达式运算`:`举例：

```
expr 123 : "[0-9]*"              # 结果为3
expr A123 : "[0-9]*"             # 结果为0
expr "$unit" : ".*"              # 返回变量unit的长度
expr `pwd` : ".*/\([^/]*\)$"     # 截取路径名的最后一个分量
```

一般来说是打印匹配长度（不匹配则为0），但如果和`\(`和`\)`使用则打印括号内能匹配的部分（否则为空字符串）

内部命令`eval`将程序中的输入或加工出来的数据作为程序来执行，包括解释和编译两个步骤，案例如下：

```
a=100
b=200
read line        # 在程序运行时输入字符串 result=`expr $a + $b + 100` (注意反撇号!)
eval "$line"     # 经过实际测试 似乎没有双引号也可以？PPT上加上双引号的目的是什么？
                 # 好像是为了处理输入为空的情况？需要测试!
echo $result     # 得到结果1300
```

shell编程中的循环包括`while`循环和`for`循环，语法为：

```
while test -r lockfile # 也可以用 while [ -r lockfile ] 代替
    do list
done

while test -r lockfile; do
    sleep 5
done

while test -r lockfile; do sleep 5; done

# 上面实际上就是将换行符换成了分号
# 每删除一个换行符就需要一个分号 测试发现分号后的空格可有可无

for name in word1 word2 ...
    do list
done

for name # 相当于 for name in $1 $2 ...
    do list
done
```

在循环结构中，可以使用`break`、`continue`、`exit`等命令，其中`break 2`命令可以退出两层循环，`exit 1`命令指定了进程结束后的返回码为`1`。

shell中的函数基本语法为`name() {list;}`，函数定义完成之后，该函数名作为一个自定义内部命令执行，后面可以调用，调用时函数名后可以附加0个到多个参数，并在函数体内部以`$1`、`$2`或`$*`、`$@`等方式进行引用。函数体内部使用命令`return`规定返回码，`0`表示成功。

33 进程基础知识

进程和程序的区别。

进程的四个组成部分：

- 指令段Text
- 用户数据段
- 用户栈段
- 系统数据段

进程虚拟地址空间的布局，详见PPT。

进程的虚实地址转换，结合两级页表，VPN1表示页面目录的偏移量、VPN2表示页表项的偏移量，VPO表示到物理页面的虚拟页面的字偏移量。最终结果是，VPN1和VPN2转换成了PPN，而VPO直接变成了PPO（不变！）。

进程的基本状态：

- 进程在创建之后，主要有运行状态和睡眠状态（也叫阻塞状态、等待状态、挂起状态，等）
- 内核总是在分时处理运行状态的进程，而不顾哪些处于睡眠状态的进程
- 睡眠状态的进程，在满足条件后转化为运行状态

进程的调度：

- 优先级：内核将可运行进程按优先级进行调度（高优先级优先）
- 进程的睡眠状态的进程一旦被叫醒后，被赋予高优先级，以保证其响应速度

- 用户程序用`nice()`系统调用可以有限地调整进程的优先级

命令`ps`可以用来查看进程信息：

- `-e`选项：列出系统中所有的进程
- `-f`选项：以`full`格式列出每一个进程
- `-l`选项：以`long`格式列出每一个进程
- 列出的属性包括：
    - `UID`：用户ID（注册名）
    - `PID`：进程ID
    - `C`：CPU占用指数，不同系统算法不同
    - `PPID`：父进程的`PID`
    - `STIME`：启动时间
    - `SZ`：进程逻辑内存大小
    - `TTY`：终端的名字
    - `COMMAND`：命令名
    - `WCHAN`：进程睡眠通道，进程阻塞在何处
    - `TIME`：累计执行时间（占用CPU的时间）
    - `PRI`：优先级
    - `S`：状态，sleep/run/zombie

进程的执行时间包括：

- `real`时间：从进程开始到完成所花费的总时间，包括进程执行时实际使用的CPU时间、进程耗费在IO上的时间、其他进程所耗费的时间等
- `user`时间：进程执行用户态代码所耗费的CPU时间，该时间仅指进程执行时实际使用的CPU时间，而不包括其他进程所使用的时间和本进程阻塞的时间

- `sys`时间：进程在内核态运行所耗费的时间，即执行内核系统调用所耗费的CPU时间

与时间有关的函数包括：

- `time()`：获取当前时间坐标，坐标原点是1970年1月1日0点
- `gettimeofday()`：获取当前时间坐标，坐标原点是1970年1月1日0点，可以精确到微秒（10^{-6}）
- `mktime`：将年月日时分秒转换为坐标值
- `ctime()`和`asctime()`、`localtime()`：坐标值和年月日时分秒的转换
- `strtfime`：定制表示日期和时间的字符串

在多任务中，“忙等待”是不可取的！

34 进程控制

`fork`系统调用是创建新进程的唯一方式，通过在内核上创建新的PCB来赋值父进程的环境（包括PCB和资源），原先的进程叫父进程，新创建的进程为子进程，其中：

- 完全复制的有：新进程的指令、用户数据段、堆栈段
- 部分复制的有：系统数据段

`fork`调用的返回值：

- 父进程得到的`fork`返回值大于零（是子进程的`PID`）
- 子进程得到的`fork`返回值为零
- 失败时返回`-1`

命令行参数和环境参数是位于进程堆栈底部的初始化数据，有三种方法可以访问环境参数：

- C库定义的外部变量`environ`
- `main`函数的第三个参数
- `getenv`库函数调用

```
main() {
    extern char **environ;
    char **p;
    p = environ;
    while (*p) printf("[%s]\n", *p++);
}

main(int argc, char **argv, char **env) {
    char **p;
    p = env;
    while (*p) printf("[%s]\n", *p++);
}

main() {
    char *p;
    p = getenv("HOME");
    if (p) printf("[%s]\n");
}
```

`exec`系统调用可以用一个指定的程序文件重新初始化一个进程，可指定新的命令行参数和环境参数。注意，`exec`不创建新进程，只是将当前进程重新初始化了指令段和用户数据段、堆栈段以及CPU的PC指针。`exec`系统调用需要一到两个下面的选项：

- `-l`：指定命令行参数的方式为表方式
- `-v`：指定命令行参数的方式为指针数组方式
- `-e`：需要指定`envp`来初始化进程

- `-p`：使用环境变量`PATH`查找可执行文件

`exec`系统调用有六种格式，详见PPT。

`wait`系统调用：

- 功能：等待进程的子进程终止
- 如果已经有子进程终止，则立即返回

库函数`system`可以在C程序中运行一个linux命令，例如：

```
int main(void)
{
    char fname[256], cmd[256], buf[256];
    FILE *f;
    sprintf(fname, "/tmp/eth-status-%d.txt", getpid());
    sprintf(cmd, "ifconfig -a > %s", fname);
    printf("Execute \"%s\"\n", cmd);
    system(cmd);
    f = fopen(fname, "r");
    while (fgets(buf, sizeof buf, f))
    printf("%s", buf);
    fclose(f);
    printf("Remove file \"%s\"\n", fname);
    unlink(fname);
}
```

35 重定向、管道、信号

磁盘文件目录分两级，包括：文件名、`i`节点。系统中活动文件目录AFD分为三级：

- 文件描述符表FDT：每个进程一张，在PCB的user结构中，记录打开的文件
- 系统文件包SFT：整个核心一张，记录诸如读写要求、引用次数、读写位置等信息
- 活动`i`节点表：整个核心一张，`inode`结构

`fork`创建的子进程继承父进程的文件描述符表FDT，并且对于父进程在`fork`之前打开的文件，父子进程有相同的文件偏移。（详见PPT，比较重要，AFD三级中只有一级被复制给了子进程，另外两级是共享的）

如果文件设置了`close-on-exec`标志，那么在执行`exec()`时系统会自动关闭这些文件。

重定向可以将已赋值的文件描述符重新赋值，通过`int dup2(int fd1, int fd2);`函数可以将文件描述符`fd1`复制到`fd2`上（`fd2`可以是空闲的文件描述符，如果是已打开的文件则关闭）。

管道