第3章
文件系统管理

主要内容

主要内容

3.1 文件名和文件通配符

文件命名和目录结构

文件和目录的命名规则

- **名字长度**
  - 一般允许1－255字符
  - 有些UNIX不支持长文件名，但至少长度为1-14
- **取名的合法字符**
  - 除斜线外的所有字符都是命名的合法字符
  - 不可打印字符也可以做文件名(除了字节0）
  - 斜线（/）留做路径名分割符 正斜线.
- **大小写字母有区别**
  - **Makefile与makefile是两个不同的文件**
  - **尽量不要依靠字母的大小写区分不同文件名，文件名带来的说明性差，不便于Windows/Linux之间的迁移**

系统配置信息

/etc目录

- 供系统维护管理用的命令和配置文件
  - 文件格式为文本文件
  - 功能类似Windows的注册表信息
- passwd，hosts文件
- *.conf文件
- ssh,xinet.d,apt,network….等目录
- 系统启动阶段系统初始化和启动各服务的脚本rc*.d
- **profile/bash.bashrc系统级bash等shell的偏好设置**
- **自定义的需要自启动的服务脚本rc.local**

文件和目录的命名规则

- /tmp
  - 临时文件，每个用户都可以在这里临时创建文件，但只能删除自己的文件，不可以删除其他用户创建的文件
- /var
  - 系统运行时要改变的数据
  - 系统日志syslog等

可运行程序和设备文件

（类似Windows的Program Files目录和Windows\system32）

- /bin
  - 系统常用命令，如ls，ln，cp，cat等
- /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，如终端设备，磁带机，打印机等
  - 尽量不要依靠字母的大小写区分不同文件名，文件名带来的说明性差，不便于Windows/Linux之间的迁移

头文件和库文件

- /usr/include（usr=Unix System Resource）
  - C语言头文件存放目录
- /lib,/usr/lib
  - 存放各种库文件，指C语言的链接库文件，以及terminfo终端库等等
  - 静态链接库文件有.a后缀(archive，存档)
  - 动态链接库文件后缀是.so（shared objects：共享的目标代码，多个.o文件的集成)
  - Linux广泛使用动态链接库，静态链接库逐渐过时

    使用动态链接库的好处

动态链接

- 动态链接与静态链接
  - 命令ldd
  - gcc hello.c –o hello　　(7.2KB)
  - gcc –static hello.c –o hello　(712KB)
- 应用程序可编程控制的动态链接库的加载、卸载和调用

文件通配符规则

文件通配符规则（1）

- **星号　***
  - 匹配任意长度的文件名字符串（包括空字符串）
  - 点字符（.）作为文件名或路径名分量的第一个字符时，必须**显式**匹配
  - 斜线（/）也必须**显式**匹配
  - 例：*file匹配file，makefile，不匹配.profile文件
    try*c 匹配try1.c try.c try.basic

try1.c
try.c
try.basic

文件通配符规则（2）

- **问号 ？**
  - 匹配任一单字符
- **方括号 [ ]**
  - 匹配括号内任一字符，也可以用减号指定一个范围
  - 例： [A-Z]* *.[ch] [Mm]akefile
- **波浪线 ~**

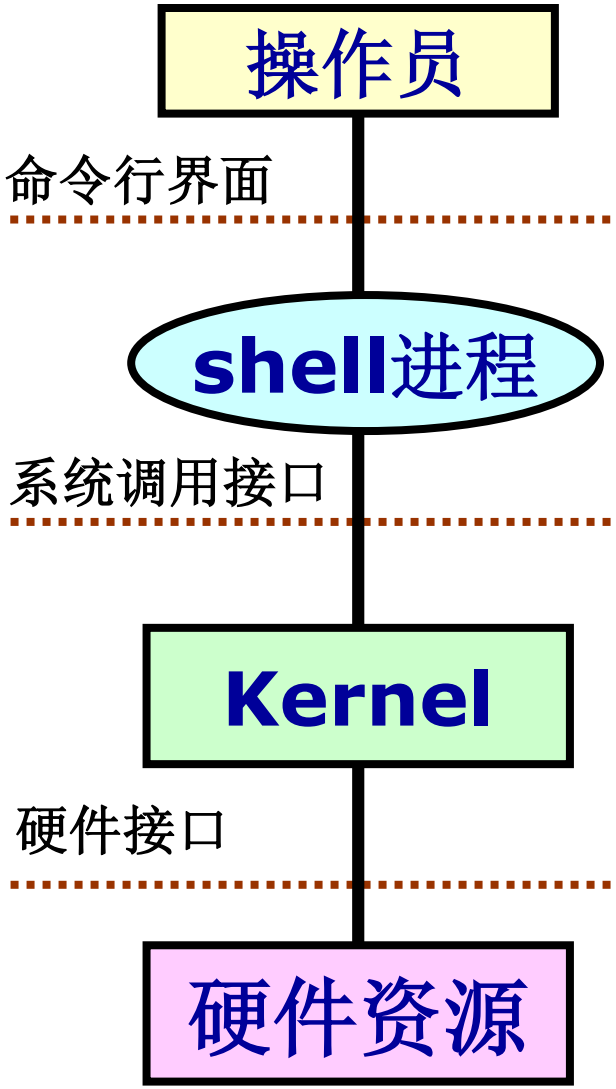
  （Bash特有的）
  - ~ **当前用户的主目录**(home)
  - ~kuan **用户**kuan**的主目录**(home)

注意

- **关于点文件.与点点..文件**
  - 当前目录与上级目录
  - 把.文件解释为当前目录，不是通配符处理程序完成的，来源于目录的存储结构
- **关于匹配规则**
  - 字符（.）作为文件名或路径名分量的第一个字符时，必须**显式**匹配
  - 文件名通配符规则与正则表达式的规则不同，应用场合不同
  - 不同种类shell通配符规则会略有些差别
  - Windows中*.* 匹配所有文件，Linux中*.*要求文件名中必须含有圆点，否则不匹配，如：*.*与makefile不匹配
  - 可以使用类似*/*.[ch]通配符， */*/*.conf
  - 文件通配符适用所有命令

文件通配符处理过程

shell与kernel



shell

- shell是一个用户态进程，如/bin/bash
- 对用户提供命令行界面
- **启动**其他应用程序（ap）使用操作系统核心提供的功能:包括系统命令和用户编写的程序

kernel：操作系统核心

- 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

程序获取命令行参数的方式

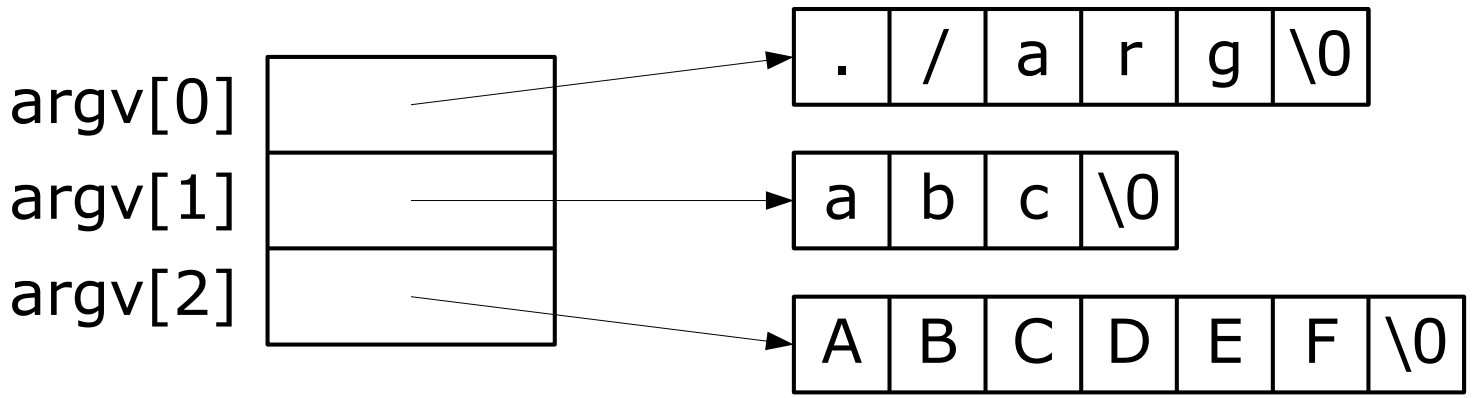
从main函数的两个参数，可获得命令行参数的内容

演示程序arg.c

```
void main(int argc, char *argv[])
{
    int i;
    for (i = 0; i < argc; i++)
        printf("%d:[%s]\n", i, argv[i]);
}
```

编译，链接：gcc arg.c -o arg

运行 ./arg abc ABCDEF



shell文件名通配符处理

- **文件名通配符的处理由shell完成，分以下三步**
  - 在shell提示符下，从键盘输入命令，被shell接受
  - shell对所键入内容作若干加工处理，其中含有对文件通配符的展开工作(文件名生成)，生成结果命令
  - 执行前面生成的结果命令

文件名通配符举例(1)

- 设当前目录下只有try.c，zap.c，arc.c三文件
  - 键入内容　cat *.c
  - 实际执行　cat arc.c try.c zap.c (按字典序)
  - 对命令cat来说,指定了3个文件
- grep a*.c try.c与grep 'a*.c' try.c的区别
  - 设当前目录下有a1.c和a2.c
  - 前者实际执行grep a1.c a2.c try.c
    - 在a2.c和try.c中查找正则表达式a1.c
  - 后者在try.c文件中查找正则表达式a*.c

文件名通配符举例(2)

键入命令时的简化输入

手工键入　　**vi m*e**

实际执行　　**vi makefile**

手工键入　　**cd *sna***

实际执行　　**cd configure-IBM-sna-network.d**

验证文件通配符处理方式

- **执行**

  ./arg *

  ./arg /usr/include/*

  ./arg */* /usr/*

- **执行结果与同样arg.c在Windows下运行的比较**
  - UNIX由shell完成对文件通配符的展开
  - Windows由命令自身来解释文件通配符

3.2 文件和目录的管理

列出文件目录

ls:文件名列表

- **基本功能**

  在同一命令行中可以指定0~n个实参

  - 不给出实参时，列出当前目录下所有文件和目录
  - 实参为文件时，列出文件项
  - 实参为目录时，列出目录下的所有文件项

- **ls命令有几十个选项**

  - 控制列表格式，有选择的为每个项目列出某些属性

- **选项-F (Flag)**

  - 若列出的是目录，就在名字后面缀以斜线 /
  - 若列出的是可执行文件，就在名字后面缀以星号 *
  - 若列出的是符号连接文件，就在名字后面缀以符号@
  - 若列出的是普通文件，则名字面后无任何标记

ls选项-F举例

- 命令`ls -F`的执行结果举例

```
bin/            pmd@
core            tmp/
dev/            unix@
etc/            usr/
lost+found/     var/
mnt/            zap*
```

ls选项-l: 长格式列表(1)

- 例：ls -l arg
  - -rwxr-x--x 1 liang stud 519 Jul 5 15:02 arg
- 第1列：文件属性
  - 第1字符为文件类型
    - – 普通文件
    - b 块设备文件(Block)
    - d 目录文件(Dir)
    - c 字符设备文件(Char)
    - l 符号连接文件(Link)
    - p 命名管道文件(Pipe)
  - 文件的访问权限(rwx读权限，写权限，可执行权限)
    - 2-4字符：文件所有者对文件的访问权限
    - 5-7字符：同组用户对文件的访问权限
    - 8-10字符:其它用户对文件的访问权限
- 第2列：文件link数，涉及到此文件的目录项数

ls选项-l: 长格式列表(2)

```
-rwxr-x--x 1 liang stud 519 Jul 5 15:02  arg
```

- 第3列，第4列：文件主的名字和组名
- 第5列
  - 普通磁盘文件：列出文件大小(字节数)
  - 目录：列出目录表大小，不是目录下文件长度和
  - 符号连接文件：列出符号连接文件自身的长度
  - 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6列：文件最后一次被修改的日期和时间
- 第7列：文件名
  - 对于符号连接文件，附带列出符号连接文件的内容

ls -l举例

```
drwxr-xr-x   3 bin  bin    3584 Jul 11 11:55 bin
-rw-------   1 root root 164470 Oct  2 11:43 core
drwxr-xr-x  11 bin  bin    7168 Oct 18 09:55 dev
drwxrwxr-x  27 bin  auth   7680 Oct 18 09:55 etc
drwxr-xr-x   2 root root   1024 Jul 11 07:24 lost+found
drwxrwxrwx   2 root bin     512 Jul 28  1998 mnt
drwxrwxrwt   2 sys  sys    4096 Oct 18 10:48 tmp
lrwxrwxrwx   1 root sys      11 Jul 11 07:31 unix -> /stand/unix
drwxrwxr-x  25 root auth    512 Oct  2 17:18 usr
drwxr-xr-x   6 root sys     512 Jul 11 07:43 var
crw-r--r--   1 bin  ter   0,  9 Oct 18 09:56 /dev/tty10
prw-r--r--   1 root sys    2642 Oct 18 11:07 /tmp/pipe
```

ls的其他选项

- 选项-h（human-readable），许多其他命令中也有类似选项
  - 以便于人阅读的方式打印数值 (例如：1K 234M 2G)
- -d (directory)

当ls的参数是目录时，不象默认的情况那样列出目录下的文件，而是列出目录自身的信息

ls与ls *的区别

ls –l /etc与ls –ld /etc的区别

ls的其他选项

- -a (all)

  列出文件名首字符为圆点的文件(默认情况下这些文件不列出，经常会用来保存用户的偏好设置信息或保存某些软件的状态信息)。

- -A (功能与-a相同，除了不列出. 和..)

  例如：

  ls ~

  ls -a ~

  ls –ad ~/.*

- -s (size)列出文件占用的磁盘空间
- -i (i-node)列出文件的i节点号

文件的复制与删除

cp: 拷贝文件

命令的两种格式和功能

cp *file1 file2*

cp *file1 file2 ... filen* ***dir***

其中*file1*，……，*filen*为文件名，*dir*为已有目录名

第二种格式中：*dir*必须已经存在并且是一个目录

第一种格式中：***file2*不存在，则创建；*file2*存在且是文件，则覆盖；*file2*存在且是目录，则按格式二处理**

如果cp的最后一个参数是一个已存在的目录，…

例：cp a.c a.bak

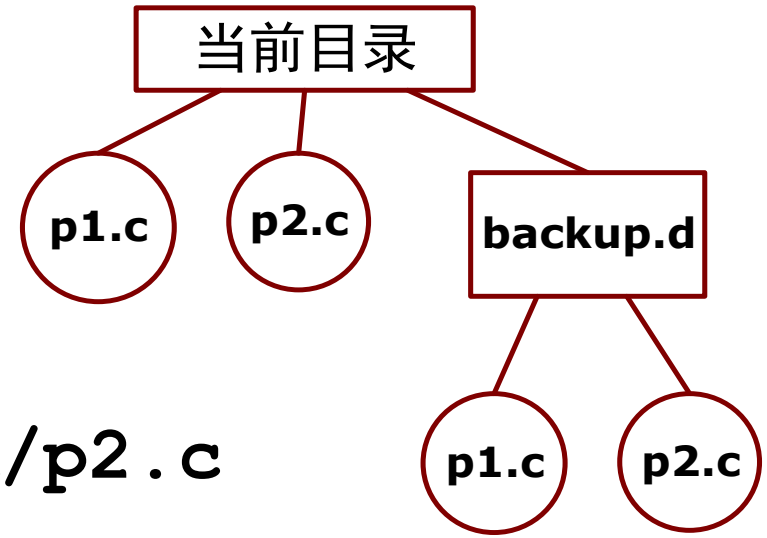
cp a.c b.c backup.d

例: cp *.c backup.d

- 与**Windows**命令COPY *.C BAK.D执行结果相同，过程不同
- UNIX中实际执行cp a1.c a2.c b1.c b2.c backup.d

cp:拷贝文件举例

```
cp backup.d/p*.c
```



实际执行：

```
cp backup.d/p1.c backup.d/p2.c
```

结果：

backup.d目录下文件p1.c将覆盖掉p2.c

（与Windows的copy backup.d\p*.c执行结果不同，Windows的命令会把这两个备份文件拷贝回当前目录）

将这两个文件拷贝回当前目录下的正确用法：

```
cp backup.d/p*.c .
```

mv: 移动文件

格式

mv *file1 file2*

mv *file1 file2 ... filen dir*

mv *dir1 dir2*

如果mv的最后一个参数是一个**已存在的目录**，…

功能

使用mv命令可以将文件和目录改名

可以将文件和子目录从一个目录移动到另一个目录

mv *dir1 dir2* 两种执行情况（同文件系统，不同文件系统）

rm: 删除文件

- **命令格式**

  rm *file1 file2 ... filen*

- **例**

  rm core a.out
  rm *.o *.tmp
  rm * .bak

- **选项**

  -r 递归地(Recursive)删除实参表中的目录，也就是删除一整棵目录树。

  -i 每删除一个文件前需要操作员确认(Inform)

  -f 强迫删除(Force)。只读文件也被删除并且无提示，无操作权限的文件强制删除也不能删掉

- **其它问题**

  - 正在运行的可执行程序文件不能被删除

显式区分命令选项和处理对象

问题

设当前目录下只有a，b，c三个文件

`rm -i`	只提供选项，未指定任何文件，命令格式错
`ls>-i`	生成文件-i(符合文件的命名规则)
`rm -i`	不能删除文件-i
`rm *`	⇒ `rm -i a b c`
`cat *`	⇒ `cat -i a b c`
`ls *`	⇒ `ls -i a b c`

解决方法

用--显式地表示命令行参数列表中选项的结束，分隔选项和处理对象，也适用于其他命令

`rm -i -- -i`

目录管理

路径名

- 文件 . 与 . .
  - 在目录表中确实有两个文件
  - 这两个目录项由系统自动创建和删除
- 主目录(Home Directory)

  每个用户都有自己独立的主目录，用env命令查环境变量HOME的值
- 绝对路径名与相对路径名

  路径分量分隔符用斜线/，而不是反斜线\

  例如 /**home**/stud/liu　　test/case1/conf
- 当前工作目录
  - 当前工作目录是**进程属性**的一部分，每进程一个

打印/改变当前目录

- **pwd命令：打印当前工作目录**

  print working directory

- **cd命令：改变当前工作目录(Change Directory)**
  - cd /usr/include
  - cd /　斜线前必须有空格
  - cd ..
  - cd　　命令无实参
    - 在**Windows**中，打印当前工作目录
    - 在UNIX中，回到用户的主目录(Home Directory)
- **注意**
  - cd是shell的一个内部命令

创建/删除目录

创建目录mkdir

例：mkdir sun/work.d

mkdir除创建目录外，系统自动建立文件.与..

- 选项-p，自动创建路径中不存在的目录。例如：

mkdir database/2019/09/04/log

删除目录rmdir

例：rmdir sun/work.d

要求被删除的目录除.与..外无其它文件或目录

- 其他命令：rm -r sun/work.d

cp: 增量拷贝

- **选项-u，增量拷贝(update)，便于备份目录**
  - 根据文件的时戳，不拷贝相同的或者过时的版本的文件，以提高速度
  - *dir1*和*dir2*不慎颠倒位置，不会出现灾难性后果
  - **Windows**中类似功能的命令XCOPY，选项/D可以用来实现增量拷贝(Date)

cp: 复制目录(举例)

- **复制目录**

  将目录work.d复制为bak.d

  若bak.d不存在，执行：cp -r work.d bak.d
  若bak.d已存在，执行上述命令会把work.d拷贝成bak.d的子目录
  cp -r work.d/* bak.d

- **增量拷贝**

  将work.d中的内容增量拷贝到备份目录bak.d中

  cp -ruv work.d/* bak.d

- **命令touch**

  将文件的最后一次修改时间设置为当前时间，但不修改文件内容。
  例如：touch *.[ch]

rsync:数据备份工具（增量拷贝）

- **rsync命令，完成**远程同步，可以镜像整个目录树
- **问题**
  - 网络中两个主机A和B，都有同一个二进制的大型数据文件**big.dat**，文件在A上做了改变成为了**新版本的big.dat**，需要让B也得到**新文件**
- **优化的流程，增量文件传输，效率高**
  - rsync用一精巧的算法，将文件分块，在两主机间传播数据块的hash值，据此推出两版本文件之间区别，使得网络只传输文件的增、删、改部分
- **Windows下的也有类似工具**

目录遍历的命令

find:遍历目录树

功能

find命令从指定的**查找范围**开始，递归地查找子目录，凡满足**条件**的文件或目录，执行规定的**动作**

举例

- find verl.d ver2.d -name '*.c' -print

范围：当前目录的子目录ver1.d 和ver2.d

条件：与名字*.c匹配。注:*.c应当用引号括起

动作：把查找到的文件的路径名打印出来

find命令的特点

命令的特点

- 功能强，选项较多
- 递归式查找，提供了一种遍历目录树的手段
- **find命令提供的灵活性：**在“动作”中可以指定任何命令（也包括用户自己编写的处理程序），使得find成为**一个任意处理命令可以借用来进行目录遍历**的壳（类似awk对文本的逐行扫描，find对目录森林中的文件和目录逐个扫描）

find关于条件的选项(1)

- **-name** *wildcard*

  文件名与*wildcard*匹配，注意：

  - ✓ 必需的引号
  - ✓ 这里的“文件名”**仅指路径名的最后一部分**
  - ✓ 对通配符的解释由find完成

- **-regex** *pattern*

  整个路径名与正则表达式*pattern*匹配

find关于条件的选项(2)

- **-type**

  f:普通文件　　**d**:目录　　**l**:符号连接文件

  c:字符设备文件 **b**:块设备文件　**p**:管道文件

- **-size ±*n*单位**

  指定文件大小（大于+，等于，小于-）

  单位有c(字符), b(块，512字节), k(1024),M(兆),G（吉），默认为b

- **-mtime ±*ndays*　文件最近修改时间**

- **-newer *file*　文件最近修改时间比*file*的还晚**

find关于条件的选项(3)

➢ 其它条件选项

（文件属性字段可用来对遍历到的目标进行筛选，查阅find手册）

指定i节点号-inum

可指定文件主-user，-nouser

可指定用户组-group，-nogroup

指定link数-links

指定路径深度-depth

指定文件的访问权限-perm

➢ 复合条件

可以用() –**o** **!**等表示多条件的“与”，“或”，“非”

find关于动作的选项

- **-print**

  打印查找的文件的路径名

- **-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，只是对符合条件的目标执行命令前需要经过操作员确认

目录遍历的应用

find使用举例(1)

- **find . -type d -print**

  从当前目录开始查找，寻找所有目录，打印路径名

  结果：按层次列出当前的目录结构

- **find / -name 'stud*' -type d -print**

  指定了两个条件：名字与stud*匹配，类型为目录

  两个条件逻辑“与”，必须同时符合这两个条件

- **find / -type f -mtime -10 -print**

  从根目录开始检索最近10天之内曾经修改过的普通磁盘文件（目录不算）

find使用举例(2)

- **find . ! -type d -links +2 -print**

  从当前目录开始检索link数大于2的非目录文件

  条件“非”用!

  注意：!号与-type之间必须保留一空格

find使用举例(3)

find ~ -size +100k \(-name core -o -name '*.tmp' \) -print

从主目录开始寻找大于100KB的名叫core或有.tmp后缀

- 使用了**两条件“或” (-o)及组合**(圆括号)
- 不要遗漏了所必需的引号，反斜线和空格，尤其是圆括号前和圆括号后。
  圆括号是shell的特殊字符
- 其他写法

find / -size +100k '(' -name core -o -name *.tmp ')' -print

find / -size +100k \(-name core –o -name *.tmp ')' -print

find使用举例(4)

find /lib /usr -name 'libc*.so' -exec **ls -lh {} \;**

- -exec及随后的分号之间的内容作为一条命令执行
- shell中分号有特殊含义，前面加反斜线\
- {}代表遍历时所查到的符合条件的路径名。注意，两花括号间无空格，之后的空格不可省略

- -ok选项在执行指定的命令前等待用户确认

find ~ -size +100k \(-name core -o -name '*.tmp' \) **-ok rm {} \;**

find使用举例(5)

利用find的递归式遍历目录的功能在文件中搜寻字符串

find src -name *.c **-exec grep -n -- --help {} /dev/null \;**

在目录src中所有.c文件中查找--help字符串

grep的-n选项，--选项，/dev/null文件的作用

其他类似做法：将满足条件的文件转码或者对文件进行其他分析处理等

批量处理文件

问题

find src -name *.c **-exec grep -n -- --help {} /dev/null \;**

借助find的“壳”功能去遍历目录，对遍历到的每个符合条件的文件执行grep命令。

缺点：效率低，因为每个命中的对象都需要执行grep命令：创建一个进程，完成任务后进程消亡，然后再创建，再消亡，……。

grep命令也提供了-r选项，可以递归地搜索子目录下的文件，例如：

grep –nr -- --help *.c 这样无法检索，因为那些子目录名字不能被*.c匹配

grep –nr -- --help * 这样可以检索，但是却检索了太多非C源程序的不感兴趣的文件，输出结果被无用的信息淹没。应该使用find精确筛选的功能。

利用find与xargs的组合

find src -name *.c **-exec grep -n -- --help {} /dev/null \;**

如果能把

find src -name *.c –print

生成的文件名列表追加在下列命令**后面就可以了**

grep -n -- --help *filelist*

命令xargs可以用来完成这个工作：

find src -name *.c –print | **xargs grep -n -- --help**

xargs命令把标准输入追加到参数表后面，也就是上述grep…的后面，再作为一个命令来执行。这样利用find精确筛选，利用grep批量处理文件，提高效率。

xargs将标准输入组织成命令执行

- 将标准输入构造为命令的命令行参数
- 如果标准输入数据量大，xargs指定的处理程序会启动多个进程运行，每个进程处理一批，会是几千个参数（命令行参数占满128K字节）
- 可以使用xargs的-n选项指定每批处理多少个
- xargs经常与find配合使用，也可以与其它命令组合使用

xargs:构造参数列表并运行命令

- 解决shell文件名生成时，因为文件太多，缓冲区空间受限而文件名展开失败的问题

  rm -f *.dat 文件名*.dat展开失败，可以使用下面的命令

  ls | grep **".dat$"** | xargs rm -f

find命中目录名因删除目录导致目录遍历过程遇到麻烦

find . -name CVS –exec rm –rf {} \;

改为：

find . -name CVS -print | xargs rm -rf

打包与压缩

tar:文件归档(1)

功能

tar命令最早为顺序访问的磁带机设备而设计的(Tape ARchive，磁带归档)，用于保留和恢复磁带上的文件

命令语法 tar ctxv[f *device*] *file-list*

选项第一字母指定要执行的操作，是必需的

c: Create创建新磁带。从头开始写，以前存于磁带上的数据会被覆盖掉

t: Table列表。磁带上的文件名列表，当不指定文件名时，将列出所有的文件

x: eXtract抽取。从磁带中抽取指定的文件，不指定文件名则抽取所有文件

tar:文件归档(2)

除功能字母外的其它选项

- **v**：Verbose冗长。每处理一个文件，就打印出文件的文件名，并在该名前冠以功能字母
- **f**：File。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)

tar的使用：磁带机操作

- **tar cvf /dev/rct0 .**

  将当前目录树备份到设备/dev/rct0中，圆点目录是当前目录

- **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的使用：文件打包

- **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。这是UNIX世界早期最常用的文件和目录打包工具
  - 这一命令**危险的误操作**是：

    tar cvf *.[ch] makefile

    漏掉了功能字母**f**必需的“设备文件名”，按照shell对文件名的展开规则，会**覆盖**掉现存的排位第一的文件

    tar cvf **a1.c** a2.c ab.h makefile

tar的使用：目录打包

设work是一个有多个层次的子目录

tar cvf work.tar work

tar cvzf work.tar.gz work （gzip压缩算法，对C源程序体积为原来的20%)

tar cvjf work.tar.bz2 work（bzip2压缩**算法，对C源程序17%，执行时间三倍**）

查看归档文件中的文件目录：

tar xvf work.tar.gz

从归档文件中恢复目录树：

tar xvf work.tar.gz

注意：文件名后缀.tar,.tar.gz,.tar.bz2仅仅是惯例，不是系统级强制要求

文件压缩和解压缩

gzip/gunzip （执行速度快）

bzip2/bunzip2 （占用较多的CPU时间，压缩效率更高）

注意：有的文件，如JPG文件，MP3文件等，不应该压缩，压缩只会浪费CPU的计算能力

3.3 命令获取信息的方法

命令获取信息的方法

命令运行时获取信息的方法

- **Linux系统命令和用户程序(ap)**
  - 从操作系统看，在地位上相同，都属于用户态程序
  - 运行时需要获取的信息包括配置信息、处理方式（选项参数）、被处理的对象
- **配置信息等硬编码是不可取的**

  硬编码需要编程时就确定服务器的地址，

  程序运行时就无法改变，太不灵活
- **运行时获取信息的常见方式**

  易变性从小到大为
  - 配置文件
  - 环境变量
  - 命令行参数
  - 交互式键盘输入

```
#define SERVER  "180.249.151.131"
main()
{
      …
}
```

配置文件

- 存储配置信息或者偏好配置信息
- 分为系统级偏好设置和用户级偏好设置，例如bash的/etc/profile 和~/.bash_profile
- 提供了灵活性（同一个程序文件因用户不同读取的配置文件不同而表现不同），变更这些信息不很方便，一般不需要变化的配置信息或选项信息存入配置文件，持久化存储

环境变量

命令env可以打印出当前的环境变量。

- 与“环境”相关的配置或选项信息，信息量不大。这些选择在一段时间内反复使用同一个命令或者不同命令时保持不变。例如：LANG(语言), HOME(主目录)，PATH(可执行文件的查找路径), CLASSPATH(类库), CVSROOT
- 虽然运行的程序（可执行文件）是完全相同的一个文件，程序通过获得环境变量感知环境的不同，控制自己的行为。
- 环境变量值的获取与设置：C语言有库函数getenv()，用户设置环境变量的方法也很简单
- 性能问题：比读取配置文件需要的系统开支要小

使用环境变量

```c
#include <stdio.h>
#include <stdlib.h>

#define DEFAULT_SERVER  "180.249.151.131"

int main(void)
{
    char *server;

    server = getenv("SERVER");
    if (server == NULL)
        server = DEFAULT_SERVER;

    printf("Server [%s]\n", server);

}
```

命令行参数和交互式输入

命令行参数

- 程序启动之前指定：通过命令行参数，操作员输入命令时提供处理选项和操作对象
- 每个命令都不同，命令运行完之后，对后续命令无影响
- 命令行参数的三种风格

交互式键盘输入：这种方式在Linux命令中极少使用

程序启动之后通过计算机与操作员之间的人机交互获取信息，C语言scanf()，fgets()函数

命令行参数的三种风格

类似dd命令的风格

特点：命令行参数采用param=value的风格

```
dd  if=sysdisk.img of=/dev/sdb
```

用dd命令将硬盘映像拷贝到硬盘：if指定输入文件，of指定输出文件

```
dd if=/dev/urandom of=test.dat bs=1024
count=512
```

用dd命令，生成512KB测试数据文件test.dat。命令行参数中：if，of，bs，count分别指定输入文件，输出文件，块大小(block size)，以及块计数

类似find和gcc的风格

特点：以减号打头的一个由多个字符构成的单词用作选项

```
find src  -name  '*.c'  -type  f  -exec dos2unix --keepdate {} \;
```

将所有扩展名.c的普通文件由Windows文本格式转为Linux格式

```
gcc -O0 -Wall -g -masm=intel  -Wa,-ahl -c shudu.c
```

编译C语言源程序文件mytest.c并生成C程序与汇编代码对比的列表信息

类似ls和grep的风格

类似ls和grep的风格：现今流行的格式

特点：长选项与短选项，有的选项同时有两种格式，也有的选项仅有长格式或仅有短格式

例如(ls的w选项指定排版时屏幕宽度)

ls --classify --all --size --human-readable --width=80 /home/jiang 长选项

ls -Fashw80 /home/jiang 多个选项挤在一起

ls -F -a -s -h -w 80 /home/jiang 多个选项分开

ls -F -w80 /home/jiang -has 可把选项放后面

用独立的命令行参数--显式地标识选项结束，选项的处理统一由C语言标准动态链接库libc.so中库函数getopt_long()完成

3.4 文件系统

文件系统的创建与安装

根文件系统与子文件系统

根文件系统(root filesystem)

- 根文件系统是整个文件系统的基础，不能“脱卸(umount)”

子文件系统

- 子文件系统，包括硬盘，软盘，CD-ROM，USB盘，网络文件系统NFS
- 以根文件系统中某一子目录的身份出现(不似Windows逻辑盘）

独立的存储结构

- 根文件系统和子文件系统都有其自己独立的文件系统存储结构，甚至文件系统的格式也不同

文件系统的创建和安装

- **创建文件系统mkfs（磁盘格式化）**

  mkfs /dev/sdb

  块设备文件/dev/sdb上创建文件系统(make filesystem)

- **安装子文件系统mount**

  mount /dev/sdb /mnt

  /mnt可以是任一个事先建好的空目录名

  此后，操作子目录/mnt就是对子文件系统的访问

  从文件或目录名看不出和其它根文件系统的对象有什么区别

  - **不带参数的mount命令**

  列出当前已安装的所有的子文件系统

- **umount命令：卸载**已安装的子文件系统：

  umount /dev/sdb

df: 文件系统空闲空间

-h human-readable

```
$ df
Filesystem     1K-blocks     Used Available Use% Mounted on
/dev/vda1       20510332 12396412   7065396  64% /
/dev/vdb1       30962684  3764236  25625636  13% /opt
$ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        20G   12G  6.8G  64% /
/dev/vdb1        30G  3.6G   25G  13% /opt
```

文件系统的存储结构

文件系统存储结构

把整个逻辑设备以块(扇区) 为单位为划分，编号为0，1，2，...。

（每块512字节或其他更大2^n字节大小）

引导块	专用块	i 节 点 区	文 件 存 储 区

- **引导块(0号块)：** 启动系统，只有根文件系统引导块有效
- **专用块(1号块)：也叫管理块，或者超级块**
  - 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，空闲空间大小，空闲块链表的头等等
  - mkfs命令时初始化，df命令读出部分信息，df -i和df

i节点区和文件存储区

- **i节点区：** i节点(index node，简记为i-node)
  - 由若干块构成，在**mkfs命令创建文件系统时**确定
  - 每块可容**若干**个i节点，i节点大小固定（比如64字节）
  - i节点从**0**开始编号，根据编号可以索引到磁盘块
  - 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（**index**）指针（组成文件的逻辑块与硬盘的物理块之间的映射）
    - 文件类型，属主，组，权限，link数，大小，时戳

    （i节点内不含文件名）
- **文件存储区**
  - 用于存放文件数据的区域，包括目录表

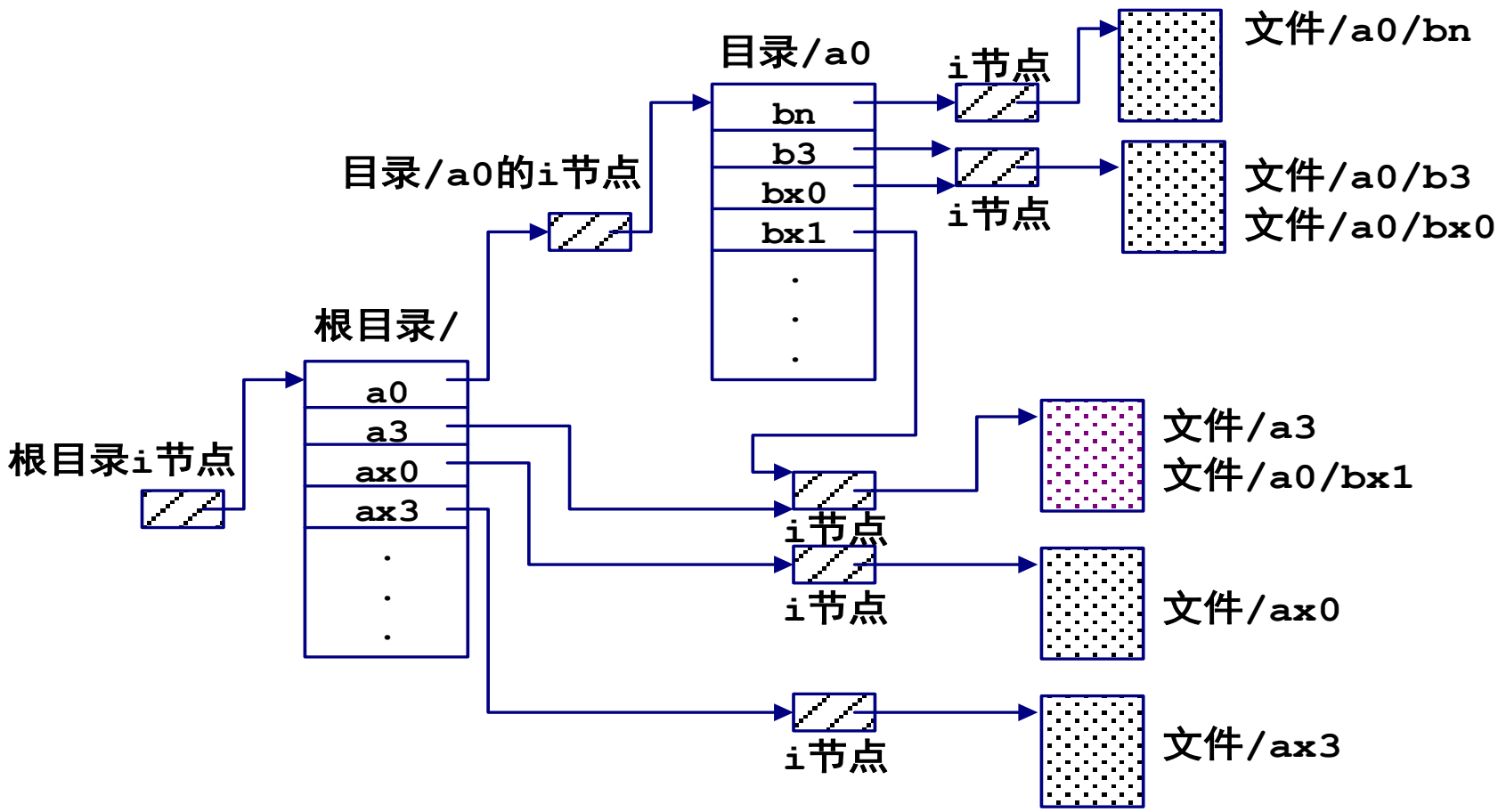
目录的存储结构

Linux目录结构是树形带交叉勾连的目录结构

目录表

- 每个目录表也作为一个文件来管理，存于“文件存储区”中，有其自己的i节点和数据存储块
- 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

目录和i节点的存储结构



目录表和i节点两级结构(1)

- **主要目的：** 分开存放的主要目的是为了提高目录检索效率
- **假定的环境**
  - 存储文件名使用定长14字节，索引信息需要64字节，每磁盘块512字节
  - 当前目录下有100个文件，需要访问的文件的文件名mydata.bin存放在目录表的最末尾处
- **方案一：一级结构**
  - 文件名和索引信息存放在一起，放在目录表中
  - 每个目录项78字节，每块可容纳512÷78=6个目录项
  - 读入目录直到第17块才找到mydata.bin以及索引信息，根据索引信息访问文件存储区的数据块

目录表和i节点两级结构(2)

方案二：两级结构

- 文件名和索引信息分开，索引信息存放在i节点中，目录表中仅记录文件名和i节点的2字节编号
- 每个目录项16字节，每磁盘块含512÷16=32个目录项
- 读入4块就检索到名字mydata.bin的i节点号
- 根据i节点号访问对应的磁盘块，读入i节点中的索引信息
- 总共磁盘操作5块，就可以根据名字找到文件的索引信息

两种方案的比较

- 后者需要更少的磁盘访问次数
- 文件系统采用这样的存储结构，完全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

命令stat：读取i节点信息

stat jane.txt

```
  File: 'jane.txt'
  Size: 280025          Blocks: 560        IO Block: 4096   regular file
Device: fd01h/64769d    Inode: 85804       Links: 1
Access: (0644/-rw-r--r--)  Uid: ( 1001/    jiang)   Gid: ( 1000/      gw)
Access: 2018-11-14 18:33:23.000000000 +0800
Modify: 2018-11-14 18:33:39.000000000 +0800
Change: 2018-11-18 01:02:30.145173450 +0800
```

硬链接

硬连接

- 目录表由目录项构成，目录项是一个“文件名-i节点号”对
- 根据文件系统的存储结构，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个硬连接
- 硬连接数目(link数)：同一i节点被目录项引用的次数

ln: 普通文件的硬连接

```
$ ln chapt0 intro
$ ls -l chapt0  intro
-rw-rw-rw- 2  kc kermit 17935 Dec 12 18:07 chapt0
-rw-rw-rw- 2  kc kermit 17935 Dec 12 18:07 intro
```

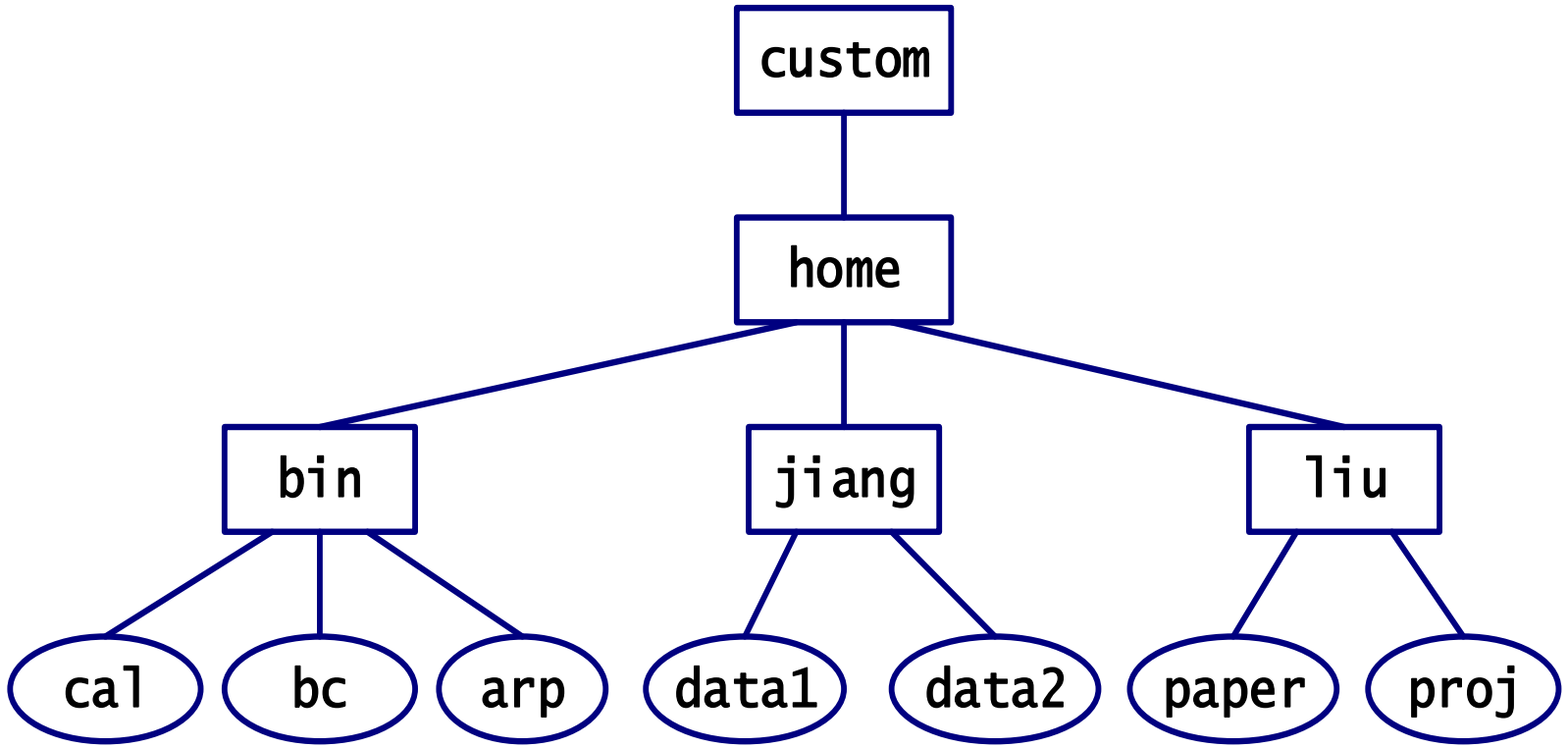
（前面的几项必相同）

```
$ ls -i chapt0  intro
    13210  chapt0
    13210  intro
```

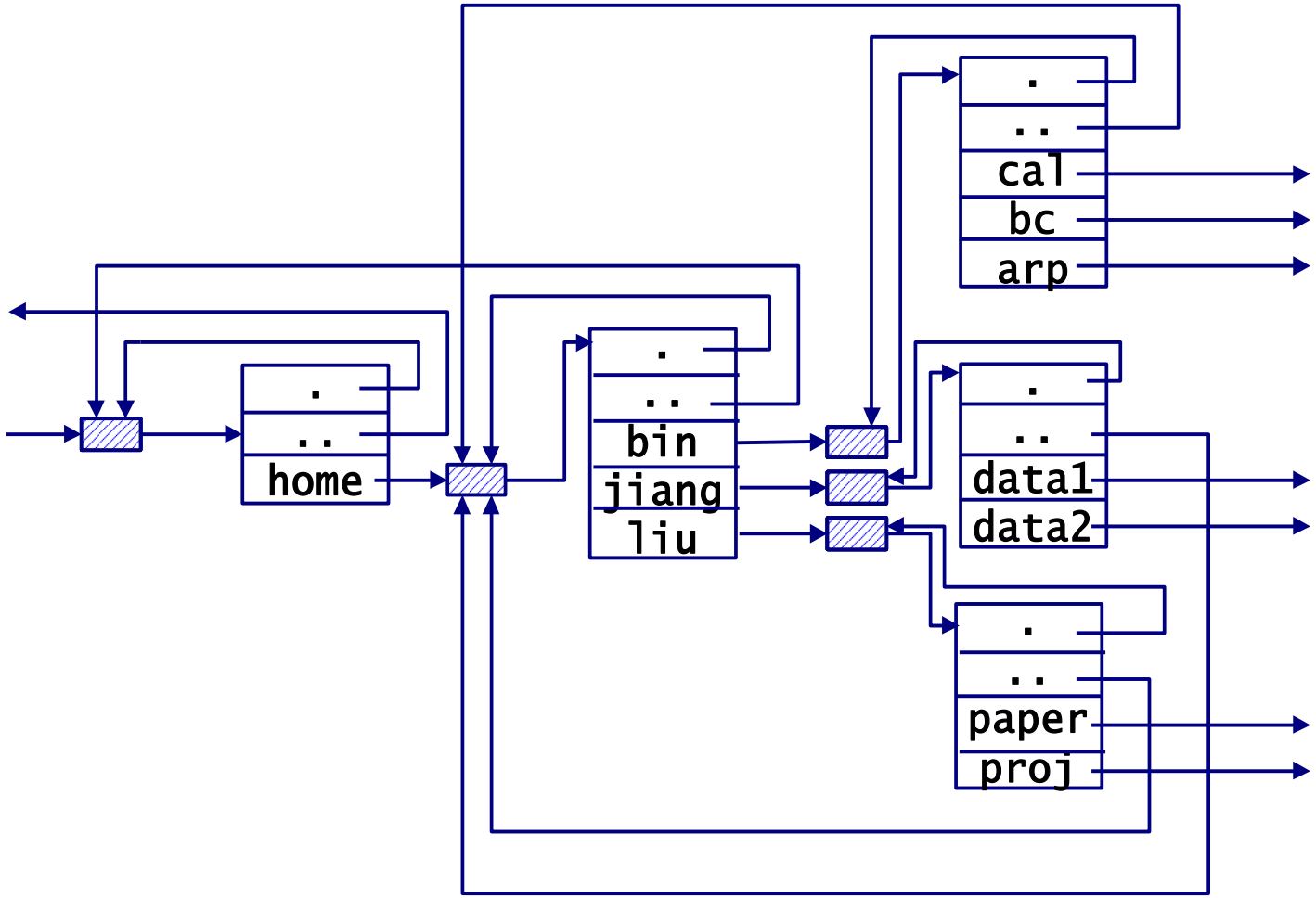
- chapt0与intro同时存在时，地位完全平等
- 删chapt0文件，则intro仍存在但link数减1
- 硬连接，只限于同一文件系统中的普通文件

目录表的硬连接

- 不允许对目录用ln命令建立硬连接
- 一般来说,目录的link数=直属子目录数+2



目录表的硬连接示意图



点和点原、

符号连接

符号连接

符号连接也叫软连接

- 用特殊文件“符号连接文件”来实现
- 文件中仅包括了一个路径名
- 命令`ln -s`和`ls -l`

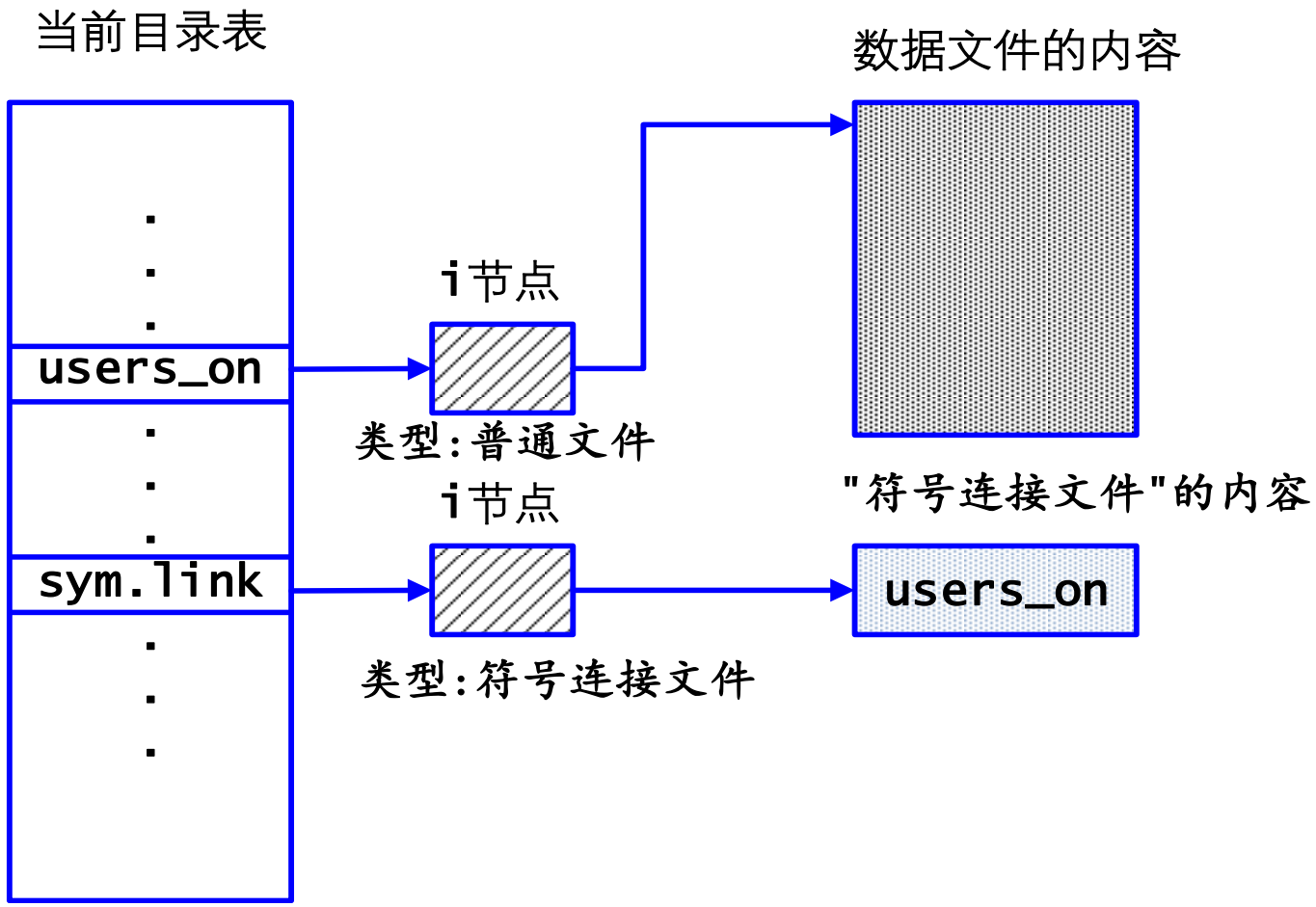
```
ln -s users_on sym.link

ls -l sym.link

lrwxrwxrwx 1 guest other 8 Jul 26 16:57 sym.link->users_on
```

  - 类型为l，大小为8字节，文件中只存放users_on字符串
  - 文件的最后一次写时间以后不再变化
  - 一旦建立了符号连接，删除操作删除的是符号连接文件，其它所有操作都将访问符号连接所引用的文件

符号连接的实现


当前目录表
.
.
.
users_on
.
.
.
sym.link
.
.
.
i节点
类型:普通文件
i节点
类型:符号连接文件
数据文件的内容
"符号连接文件"的内容
users_on

符号连接中的相对路径

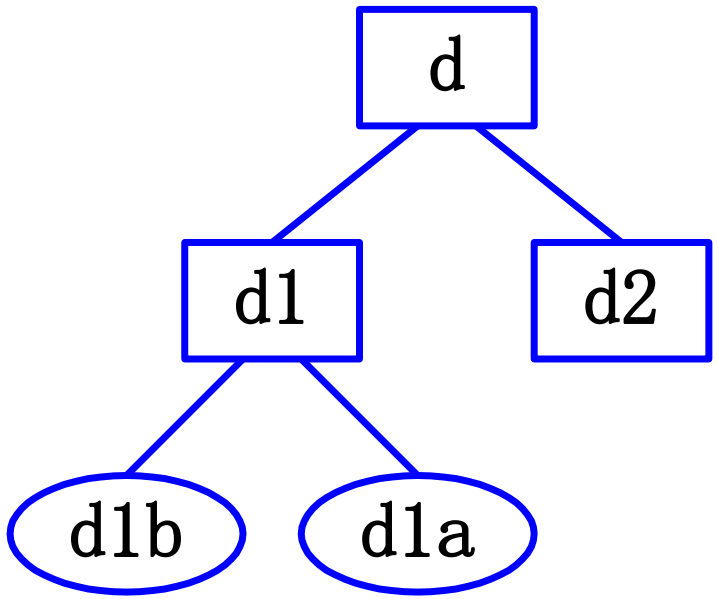
- 若符号连接含绝对路径名，引用绝对路径名
- 若符号连接含相对路径名，是相对于符号链接文件的位置(不是相对于调用进程的当前工作目录)

设当前目录（bash进程的当前目录）为d

ln -s d1/d1b d1/dx

在d1下新建文件dx

访问d1/dx实际访问d1/d1/d1b



硬连接与符号连接的比较

硬连接

- 在数据结构层次上实现
- 只适用于文件，不适用于目录
- 不同文件系统之间也不行
- 硬连接能够完成的功能软连接可以做到

符号连接

- 在算法软件上实现
- 硬连接能够完成的功能软连接都可以做到
- 适用于目录，也适用于不同的文件系统
- 同硬连接相比要占用操作系统内核的一部分开销
- 循环式符号连接，以及处理方法（解析路径时设置符号链接解析计数器）

系统调用

系统调用(System call)

- 系统调用以C语言函数调用的方式提供
- 操作系统内核提供的编程界面

  应用程序(ap)和操作系统(kernel)进行交互的**唯一手段**

  例如：文件操作的open，read，write，close
- 种类
  - 早期UNIX有50多个，后来扩充到120个，Linux有300个左右

系统调用(System call)

系统调用与库函数在执行方式上的区别

例如：获取进程ID的getpid()与字符串拷贝函数strcpy()

CPU的INT指令（软中断）与CALL指令（子程序调用）

库函数对系统调用的封装（API）

目的：执行效率更高或者调用界面更方便。例如：

库函数printf对系统调用write的封装

库函数malloc/free对系统调用sbrk的封装

可移植性

系统调用和相关API函数以及库函数的名称、参数排列顺序、参数类型，返回值的类型，以及实现的功能，都属于类似POSIX标准规范的内容，便于不同Unix系统之间的移植

系统调用函数的返回值

- **返回值：**返回一个整数值
  - 返回值大于或等于零：成功
  - 返回值小于0：如-1，表示失败
- **整型变量errno（或可像整形变量那样可读取的宏定义）**
  - C标准库定义了errno，系统调用失败后自动填写错误代码，记录失败原因

  #include <errno.h>之后，就可以直接使用errno
  - errno.h头文件定义了许多有E前缀的宏，例如
    - EACCESS，EIO，ENOMEM，EINTR
    - 相关系统调用的手册页中有出错说明

  在man手册页ERRORS节介绍出错原因，如man recv

库函数strerror与printf的格式符%m

strerror

char *strerror(int *errno*);

errno是个整数，便于程序识别错误原因，不便于操作员理解失败原因。

库函数strerror将数字形式的错误代码转换成一个可阅读的字符串

printf的%m

printf类函数格式字符串中的%m会被替换成上次系统调用失败的错误代码对应的消息（message）

errno, strerror, %m

```c
#include <fcntl.h>
#include <stdio.h>
#include <errno.h>
#include <string.h>

int main(char argc, char *argv[])
{
    int fd;

    fd = open(argv[1], O_WRONLY);
    if (fd == -1) {
        printf("ERROR %d: %m\n", errno);
        printf("ERROR [%s]\n", strerror(errno));
    }
    . . . . . .
}
```

访问i节点和目录

系统调用stat/fstat

从i节点获得文件的状态信息

- stat得到指定路径名的文件的i节点
- fstat得到已打开文件的i节点

stat和fstat将数据放入调用者提供的stat结构中

放入调用者提供的 stat 结构 buf 中.

```
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

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

结构体stat

```
struct stat {
    dev_t      st_dev;          /* 存储该文件的块设备的设备号ID */
    ino_t      st_ino;          /* inode号 */
    mode_t     st_mode;         /* 访问权限及文件类型 */
    nlink_t    st_nlink;        /* link数 */
    uid_t      st_uid;          /* 文件主ID */
    gid_t      st_gid;          /* 组ID */
    dev_t      st_rdev;         /* device ID (if special file) */
    off_t      st_size;         /* 文件大小（字节数）*/
    blksize_t st_blksize;       /* blocksize for filesystem I/O */
    blkcnt_t   st_blocks;       /* 分配的512字节尺寸块个数 */
    struct timespec st_atim;    /* access时间 */
    struct timespec st_mtim;    /* modification时间 */
    struct timespec st_ctim;    /* change时间 */
};
```

结构体stat

- st_dev：存储该文件的块设备的设备号，包括主设备号与次设备号

例如：stat命令显示文件Device: 821h/2081d

十六进制0821，主设备号8（高字节），次设备号33(低字节)， /dev/sdc1

ls -l /dev | grep '^b.* 8, *33'

brw-rw---- 1 root disk 8, 33 Nov 18 10:40 sdc1

- st_mode域：16比特

文件的基本存取权限和SUID/SGID权限(11比特)以及文件的类型(若干比特)

文件类型判st_mode & S_IFMT

- S_IFREG 普通磁盘文件
- S_IFDIR 目录文件
- S_IFCHR 字符设备文件
- S_IFIFO 管道文件
- S_IFLNK 符号连接文件

结构体stat

st_size与st_blocks

程序可以通过st_size获取文件大小。

一般情况:st_size ≤ st_blocks * 512

稀疏文件:st_size > st_blocks * 512

st_ctim，st_atim，st_mtim域

Linux中存储这三个时间的精度为纳秒

“**a**访问”：读，执行（有些系统为了效率做懒惰处理，不更新，但不早于m时间）

“**m**修改”：**文件内容修改。写文件**

“**c**改变”：**i节点信息变化。**写文件，修改权限/link数/文件主等（m变，c也变）

访问目录

早期UNIX象普通磁盘文件那样open()打开目录read()读取

现在的系统不再这样操作，而是直接使用封装好的库函数

目录访问的一组库函数

```
#include <dirent.h>
DIR *opendir(char *dirname);
struct dirent *readdir(DIR *dir);
int closedir(DIR *dir);
```

- opendir打开目录得到句柄（NULL表示失败）
- readdir获取一个目录项
  - 返回值指针指向的dirent结构体（返回NULL表示已经读到目录尾）
  - dirent结构体：记录i节点号和文件名(d_ino和d_name成员)
- 访问结束：用closedir关闭不再使用的目录句柄。

目录访问程序

```
int main(int argc, char *argv[])
{
    DIR *dir;
    struct dirent *entry;
    if (argc != 2) {
        fprintf(stderr, "Usage : %s <dirname>\n", argv[0]);
        exit(1);
    }
    if ((dir = opendir(argv[1])) == NULL) {
        printf("Open directory \"%s\": %s (ERROR %d)\n",
            argv[1], strerror(errno), errno);
        exit(1);
    }
    while ((entry = readdir(dir)) != NULL)
        printf("%d %s\n", entry->d_ino, entry->d_name);
    closedir(dir);
}
```

编程构造自己的工具

- **目录访问：可以穷举一个目录中的所有项目，可得文件名及节点号**
- **stat调用：可以根据文件名获得文件i节点中的状态信息**

由此，可以编程构造诸如ls, rm, find等类似的工具，并理解系统命令是如何实现的

作业二：遍历目录

编程作业：遍历文件目录

编程实现程序list.c，列表普通磁盘文件，包括文件名和文件大小。

使用vi编辑文件，熟悉工具vi。

使用Linux的系统调用和库函数。

体会Shell文件通配符的处理方式以及命令对选项的处理方式。

（对选项的处理，自行编程逐个分析命令行参数。不考虑多选项挤在一个命令行参数内的情况）

处理对象和选项

- **与ls命令类似，处理对象可以有0到多个**
  - 0个：列出当前目录下所有文件
  - 普通文件：列出文件
  - 目录：列出目录下所有文件
- **实现自定义选项r,a,l,h,m以及--**
  - r 递归方式列出子目录（每项要含路径，类似find的-print输出风格，需要**设计递归程序**）
  - a 列出文件名第一个字符为圆点的普通文件（默认情况下不列出文件名首字符为圆点的文件）
  - l 后跟一整数，限定文件大小的最小值（字节）
  - h 后跟一整数，限定文件大小的最大值（字节）
  - m 后跟一整数n，限定文件的最近修改时间必须在n天内
  - -- 显式地终止命令选项分析

编辑，编译，运行

- 编辑，编译

  vi list.c

  make list 或者 gcc list.c –o list

- 运行举例

  ./list –l 100 –h 5000 /bin /etc　　列出大小在100~5000之间的文件

  ./list –a -r -l 50000 –m 2　　递归式列出当前目录树下大小超50KB且2天内修改过的文件（包括文件名首字符为圆点的文件）

  ./list -- -l

  ./list *

- 延伸学习

  用于处理命令选项的库函数getopt_long，用这个函数重新设计选项处理部分，设计长短格式选项。体会这个库函数功能的设计思想

参考函数

```
#include <stdio.h>
#include <dirent.h>
#include <sys/stat.h>

/* 获取路径名path对应的i节点中的属性 */
struct stat st;

ret = stat(path, &st);

/* 判断i节点属性为目录 */
if (S_ISDIR(st.st_mode)) {
    . . .
}
```

3.5 文件和目录的权限

文件的权限

文件的权限

用于控制进程对系统中文件和目录的访问

- **权限的三个级别**
  - 文件主，同组用户，其他用户
  - 每个文件有唯一的属主
- **普通文件的权限**
  - 读、写、可执行
  - 不可写文件也可**能会被**删除

两类可执行文件

- **程序文件（可执行文件）**
  - 二进制的CPU指令集合，满足操作系统规定的格式才可以被加载运行
- **脚本文件：文本文件**
  - 默认的**解释**程序为/bin/sh
  - 可以在文件的第一行自行指定解释程序（必须是第一行，#！必须是这个文件首先出现的两个字符），例如：

    #! /bin/bash
    #! /usr/bin/bc
  - 解释程序也可以是用户自己编写的应用程序
  - **脚本程序运行时，实际上是由解释程序创建了一个进程**

目录的权限

目录的读写权限

- 读权限
  - 若无读权限，那么“目录表”文件不许读，ls会失败
- 写权限
  - 若无写权限，那么“目录表”文件不许写
  - 创建文件，删除文件，文件改名会修改目录文件
  - 修改文件不需要修改目录文件，需要修改i节点
  - 目录无写权限不是指目录下所有文件禁止写

目录的x权限

- 执行权限
  - 有执行权限意味着**分析路径名**过程中可检索该目录
  - cat /a/b/c
  - 要求/a，a/b三目录有x权限，c文件有读权限；否则，命令执行失败
  - cd ../st8要求当前目录，..和st8必须有x权限

STICKY权限（粘着位）

- **STICKY文件**
  - 早期Unix，有sticky属性可执行文件尽量常驻内存或交换区以提高效率
  - 现代Linux对访问过的文件自动缓冲在内存，文件sticky属性被忽略
- **STICKY目录**
  - **问题：对于公共目录，用户user1和user2没有写权限，就不可以在这个目录下创建新文件；若有写权限，用户user1的文件就算是“只读文件”也可以被user2删除**
  - **STICKY属性用于解决这个问题：目录有写权限并且带STICKY属性，此目录下的文件仅文件主可以删除，其他用户删除操作会失败**
  - 例如：/tmp目录，**ls -ld /tmp，输出的第一列**最后字符为**t**

drwxrwxrw**t** 13 root root 20480 Mar 22 06:11 /tmp

权限验证的顺序

每个文件都有文件主和组的属性（文件节点中）
每个进程也有进程主和组的属性（进程PCB中）
都是整数，uid和gid的编号与名字对应关系见/etc下passwd和group文件

- 文件主与进程主相同
  - 使用文件主权限，不再查组和其他用户的权限
- 文件主与进程主不同,但文件主与进程主同组
  - 只使用组权限，不使用关于其他用户的权限
- 文件主与进程主不同,文件主与进程主又不同组
  - 使用文件关于其他用户的权限。

注意：超级用户root不受权限的限制

例： 权限`---r--rw-`，文件主不可读但同组用户可读，即使文件主是该组用户之一也不行

权限相关命令

确定文件的权限

- 使用ls命令

  有关选项-l和-d

  例:ls -l 可以查当前目录下所有文件和子目录的权限

```
drwxrwx---   3 bin backup 512 Jul 11 07:31 sysadmin
drwxr-xr-x   2 bin bin    512 Jul 11 07:21 tabset
drwxr-xr-x   3 bin bin    512 Jul 11 11:55 tcl
-r--r--r--   1 bin bin   7820 Jul 11 11:53 tclXmain.o
drwxr-xr-x  10 bin bin    512 Jul 11 07:32 tcprt
drwxr-xr-x  43 bin bin   1024 Jul 11 07:24 terminfo
drwxr-xr-x   3 bin bin    512 Jul 28  1998 terminfo
-rw-r--r--   1 bin bin   6997 Jul 11 06:56 timconv
```

  ls -ld . 列出当前目录自身的权限

```
drwxr-xr-x 51 bin bin   4608 Aug 15 16:51 .
```

chmod：修改权限（字母形式）

- 字母形式

```
chmod [ugoa][+-=][rwxst] 文件名表
u--user   文件主的权限
g--group  同组用户的权限
o--other  其他用户权限
a--all    所有上述三级权限
(t--Sticky, s--SUID)
例: chmod u+rw *
    chmod go-rwx *.[ch]
    chmod a+x batch
    chmod u=rx try2
```

chmod：修改权限（数字形式）

- 数字形式(八进制数字)

例：chmod　644　xyz1 xyz2

八进制：	6	4	4
二进制：	110	100	100
权限：	rw-	r--	r--

注意： 只允许文件主和超级用户修改文件权限

umask命令

- **功能：决定文件/目录的初始权限**
  - 用vi新建文件
  - 用输出重定向创建文件
  - 创建新目录
- **umask是进程属性的一部分**
  - umask是shell内部命令
  - umask是进程属性的一部分
- **命令**
  - umask　　打印当前的umask值
  - umask 022　将umask值设置为八进制的022

进程umask属性的作用

掩码值的含义

例：掩码值（八进制）： 022

二进制： 000 010 010

取消新文件/目录的组w权限和其他用户w权限

umask 077

禁止组权限和其他用户权限（对应比特1处的权限被屏蔽掉）

自动执行批处理文件

将umask命令放到shell自动执行批处理文件中

- bash的$HOME/.bash_profile

系统调用umask

- **功能**
  - 修改进程自身的umask属性值
- **初创文件的权限**
  - **受open的规定值和进程自身属性umask值影响**
  - 已存在的文件的权限，不受open/umask的影响

    例:当umask为077时，用C程序

    ```
    fd=open(filename,O_CREAT|O_WRONLY,0666);
    ```

    open的权限为0666，屏蔽掉077后实际为0600
- **系统调用umask**

  ```
  int umask(int mask);
  ```

  - *mask*为指定的新umask值，返回值为原先的umask值
  - 读出进程umask属性而不改变它，需调umask两次

设定文件和目录的权限

演示：文件的读写权限

文件的写权限

```
$ who am i
jiang      pts/2        Jun 06 08:34
$ who > mydata
$ ls -l mydata
-rw-r--r--   1 jiang    usr             58 Jun 06 09:04 mydata
$ chmod u-w mydata
$ who >> mydata (只读文件不许写)
mydata: The file access permissions do not allow the specified action.
$ rm mydata (只读文件可以被删除)
rm: Remove mydata? y
$ ls -l mydata
ls: 0653-341 The file mydata does not exist.
```

文件的读权限

```
$ who > mydata
$ chmod u-rw mydata
$ cat mydata (无法读取不允许读的文件中内容)
cat: 0652-050 Cannot open mydata.
$ chmod 644 mydata
```

演示：目录的w权限

目录写权限

$ chmod u-w . (当前目录不许写)
$ who > mydata2 (不能创建新文件)
mydata2: The file access permissions do not allow the specified action.
$ who >> mydata (但是可以修改已有的文件,追加一部分数据)
$ rm mydata (不能删除文件)
rm: 0653-609 Cannot remove mydata.
The file access permissions do not allow the specified action.
$ cp /etc/passwd mydata (可以覆盖旧文件)
$ cp /etc/passwd mydata2 (不能创建新文件))
cp: mydata2: The file access permissions do not allow the specified action.
$ mv mydata MyData （文件不许改名)
mv: 0653-401 Cannot rename mydata to MyData:
The file access permissions do not allow the specified action.
$ mkdir Test (不可创建子目录)
mkdir: 0653-357 Cannot access directory ..
.: The file access permissions do not allow the specified action.

演示：目录的r权限

目录读权限

$ pwd
/usr/jiang
$ chmod u-r .
$ ls (不可读的目录无法列表出其中文件)
ls: .: The file access permissions do not allow the specified action.
$ chmod 000 . (取消当前目录所有权限)
$ ls
ls: 0653-345 .: Permission denied.
$ chmod 755 . (试图恢复当前目录权限失败，因为试图访问当前目录下的.文件)
chmod: .: The file access permissions do not allow the specified action.
$ chmod 755 /usr/jiang (这种访问不需要当前目录权限，可恢复当前目录权限)

演示：目录的x权限

子目录没有读写权限，但是保留了x权限

```
$ chmod u=x ttt
$ cat ttt/ccp.c
main(int argc, char **argv)
{    ...
}
$ rm ttt/arg.c (子目录没有写权限，不能删除其中的文件)
rm: 0653-609 Cannot remove ttt/arg.c.
The file access permissions do not allow the specified action.
$ ls ttt  (子目录没有读权限，不能列出其中的文件)
ls: ttt: The file access permissions do not allow the specified action.
```

子目录有读写权限,但没有x权限

```
$ chmod u=rw ttt
$ ls ttt
BUGS.report  arg.c       ccp.c       chap.h       mydata
arg          auth.c      chap.c      disk.img
$ cat ttt/arg.c
cat: 0652-050 Cannot open ttt/arg.c.
(试图设置其他用户的文件或目录的权限)
$ chmod 777 /
chmod: /: Operation not permitted.
```

SUID权限

三级权限存在的问题

问题

- 系统中任一个用户，要么对文件的全部内容具有访问权，要么不可访问文件。有的情况下，很不方便。
- 用户修改口令:文件/etc/passwd和/etc/shadow
- 用户liu的文件list.txt：希望用户liang，只能读取行首为#的行和与他有关的行。

```
#=============================================================
# 登录名    工作证号    姓名      月份      工资      奖金      补助      扣除      总额
#-------------------------------------------------------------
    tian      2076     田晓星     03       4782      4500      200       175      8307
    liang     2074     梁振宇     03       4560      4400      180        90      8050
    sun       3087     孙东旭     03       4804      4218      106       213      7915
    tian      2076     田晓星     04       4832      4450      230       245      8267
    liang     2074     梁振宇     04       4660      4450      230        70      8270
    sun       3087     孙东旭     04       4700      4310      283       270      8023
#=============================================================
# 注：燃气费自本年度开始不再从工资中扣除。
```

用户liu的程序query.c

```
int main(void)
 {
    FILE *f;
    char line[512], name[64], *username;

    if ((f = fopen("list.txt", "r")) == NULL) {
        printf("*** ERROR: Open file \"list.txt\": %m\n");
        exit(1);
    }

    username = getlogin();
    while (fgets(line, sizeof(line), f)) {
        if (line[0] == '#')
            printf("%s", line);
        else if (sscanf(line, "%s", name) > 0 &&
             strcmp(name, username) == 0)
            printf("%s", line);
    }
}
```

简单的三级权限

- **用户liu**

  源程序经过编译后，生成可执行文件query。为了保密，用户liu将文件list.txt的权限设为rw-------，文件query的权限为rwx--x--x

```
$ chmod 600 list.txt
$ chmod 711 query
$ ls -l list.txt query
-rw-------  1 liu  leader 722   Dec 10 23:04 list.txt
-rwx--x--x   1 liu leader 56134 Dec 10 23:07 query
```

- **用户liang执行命令query**

```
$ query
*** ERROR: Open file "list.txt" : Permission denied
$ cat list.txt
cannot open list.txt: Permission denied
```

SUID权限

- 文件query的文件主liu给文件query增加SUID权限

```
$ chmod u+s query
$ ls -l query
-rws--x--x   1 liu   leader   56134 Dec 10 23:07 query
```

- 用户liang通过query命令查询只许liu可读的文件list.txt

```
$ ls -l list.txt query
-rw-------   1 liu   leader     722 Dec 10 23:04 list.txt
-rws--x--x   1 liu   leader   56134 Dec 10 23:07 query
$ query
#=============================================================
# 登录名  工作证号  姓名     月份    工资    奖金    补助    扣除    总额
#-------------------------------------------------------------
   liang    2074    梁振宇    03     4560    4400     180      90    8050
   liang    2074    梁振宇    04     4660    4450     230      70    8270
#=============================================================
# 注：燃气费自本年度开始不再从工资中扣除。
$ cat list.txt
cannot open list.txt: Permission denied
```

进程实际UID/有效UID

- 一般情况下，进程的实际UID和有效UID相等。
- 打开文件open()时，系统根据进程的有效UID，与文件所有者UID之间的关系和文件的权限进行访问合法性验证
- 可执行程序**具有**SUID权限，进程的实际UID和有效UID不再相等。实际UID是当前用户，而有效UID为可执行文件的文件主。
- SUID使得用户可以通过文件主提供的程序，以文件主的权限访问文件，但这种访问依赖于文件主提供的程序，进行有限的访问。