第4章 UNIX的文件和目录

4.1 文件和目录的层次结构

UNIX系统通过目录管理文件，文件系统组织成树状结构，目录中可以含有多个文件，也可以含有子目录。UNIX系统中，路径名分割符用正斜线/。表4-1所示为一些常见的目录和文件。与系统有关的一些主要目录的取名和在层次结构中的位置，几乎在所有UNIX系统中都相同。

表4-1 UNIX常见的目录和文件

路　　径	说　　明
/unix	程序文件，UNIX 内核
/etc	供系统维护管理用的命令和配置文件。例如： 文件/etc/passwd:存放的是用户相关的配置信息； 文件/etc/issue:登录前在 login 之上的提示信息； 文件/etc/motd:存放登录成功后显示给用户的信息。 文件系统管理的程序有 fsck，mount，shutdown 等。 许多系统维护的命令，在不同的 UNIX 系统之间区别很大
/tmp，/usr/tmp	存放临时文件
/bin	系统常用命令，如 ls，ln，cp，cat 等
/dev	存放设备文件，如终端设备文件，磁带机，打印机等
/usr/include	C 语言头文件存放目录
/usr/bin	存放一些常用命令，如 ftp，make 等
/lib,/usr/lib	存放各种库文件，包括 C 语言的链接库文件，动态链接库，还包括与终端类型相关的 terminfo 终端库，等等。静态链接库文件有.a 后缀，动态链接库文件的后缀不是.DLL，而是.so。UNIX 很早就广泛地使用动态链接库，静态链接库逐渐过时。.a 取名于 archive（存档），.so 取名于 shared objects（共享对象）
/usr/spool	存放与用户有关的一些临时性文件,如：打印队列,已收到但未读的邮件等

4.2 文件和目录的命名

（1）名字长度。现在的UNIX都支持长文件名，文件名长度的最大值都在200以上，早期的UNIX至少可以支持长度为14个字符的文件名。

（2）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，空格、星号甚至不可打印字符也可以做文件名。一个字节的取值0~255之中，47是斜线的ASCII码，不可作为文件名，ASCII码0用作C语言的字符串结束标志，其余的254种取值都可以作为文件名。

（3）大小写字母有区别。例如：makefile，Makefile，MAKEFILE是三个不同的文件名。

4.3 shell的文件名通配符

4.3.1 规则

- UNIX的文件名通配符是由shell程序解释的。
- 对几乎所有的shell来说，表4-2列出的有关文件名通配符的规则几乎都一致。

表4-2 常用的shell文件名通配符

符号	含　义
*	匹配任意长度（包括空字符串）。如：try*c 匹配 try1.c，try.c，try.basic。例外的情况是当文件通配符的第一个字符为*时，*不匹配以句点（.）开头的文件。例：*file 匹配文件 file，tempfile，makefile，但不匹配.profile 文件
?	匹配任一单字符。例如：p?.c 可以匹配 p1.c，p2.c，pa.c
[]	匹配括号内任一字符，也可以用减号指定一个范围。例如：[A-Z]*匹配所有名字以大写字母开头的文件，*.[ch]匹配所有含.c 或者.h 后缀的文件，[Mm]akefile 匹配 Makefile 或者 makefile

4.3.2 与DOS文件名通配符的区别

- UNIX的文件名通配符要无二义性。
- 在UNIX中，文件名通配符允许用于任何命令，而DOS中只能用于dir/del/copy等有限的几个命令中。
- 关于文件扩展名。DOS中*.*匹配所有文件，UNIX中*.*要求文件名中必须含有句点，否则不匹配。
- 匹配子目录中的文件。在UNIX中可以使用*/*.[ch]通配符，匹配当前目录下所有一级子目录中文件名后缀为.c和.h的文件，这在DOS中不允许。

4.3.3 文件名通配符的处理过程

UNIX处理文件名通配符的过程分三步：

（1）在shell提示符下，从键盘输入命令，输入的命令被shell所接受。

（2）shell对所键入的内容作若干种加工处理,其中含有对文件名通配符的扩展工作，生成结果命令。

（3）调用操作系统的系统调用，创建新的进程执行命令，并把参数传递给新进程，执行生成的结果命令。

shell在第二步中，含有文件名生成工作，把用空格分开的每一段作为一个“单词”，扫描每个词，从中寻找 * ？ []。

- 如果其中之一出现，则该词被识别为一个文件名通配符，用与文件名通配符相匹配的文件名表取代该词。
- 可以匹配多个名字时，按字母序排列多个名字。
- 如果没有找到与文件名通配符相匹配的文件名，在**B-shell**中不改变该词，在**C-shell**中产生错误。

【例4-1】 体验shell对文件名通配符的展开处理。

（1）设当前目录下只有try.c，zap.c，arc.c三文件，在shell提示符下，键入命令：

cat *.c

- shell根据当前目录下的所有文件的文件名集合，将*.c扩展为arc.c try.c zap.c，扩展后的多个文件名按照字典序排列。
- 这样，cat *.c被加工成了cat arc.c try.c zap.c，实际执行加工之后的命令。
- 从cat命令的角度来说，都是指定了三个文件作为处理对象，cat程序在执行的时候，已经看不到*.c，它看到的是三个文件名。

（2）设当前目录下有四个文件0131.rpt，0130.rpt，wang.mail，lee.mail。在两个mail文件中查找数字串的命令为：

grep '[0-9][0-9]*' *.mail

（3）使用文件名通配符，可以简化一些命令的输入，尤其是那些较长的名字。

如：vi m*e 替换成 vi makefile

cd *work.d替换成cd configure_network.d。

(cd后面只能有一个参数，此时只能匹配一个名字)

4.3.4 验证文件名匹配的结果

【例4-2】 从程序员的角度理解shell对通配符的处理。

编写一个很简短的C语言程序文件arg.c。

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

编译、链接以生成可执行文件。使用命令：

cc arg.c -o arg

或者使用命令：

make arg

在当前目录下生成名为**arg**的可执行文件，没有**Windows**系统中那样的**.EXE**扩展名。执行这个程序的命令，如：

./arg abc ABCDEF

- 在UNIX系统中，如果直接键入arg abc ABCDEF，默认情况下，shell会只在系统规定的目录中搜索指定文件arg，搜索不到也不会到当前目录下搜索。
- 需要用命令./arg显式地指定程序文件存储的路径为点目录(./)，shell就会到./目录下搜寻执行文件arg。
- 用户可以通过设置shell环境变量PATH将当前目录增加到系统的搜索路径中去。一般默认情况下不搜索当前目录。

针对输入的./arg abc ABCDEF，argv数组的布局如图4-1所示。在C语言中，从主函数main的两个参数，可以获得命令行参数的内容。

- 第一个参数argc是命令行参数的个数
- 第二个参数argv是一个指向数组的指针，数组的每个元素是个指针，指向一个字符串，argv[0]字符串是命令自身，其余的是命令行的参数。

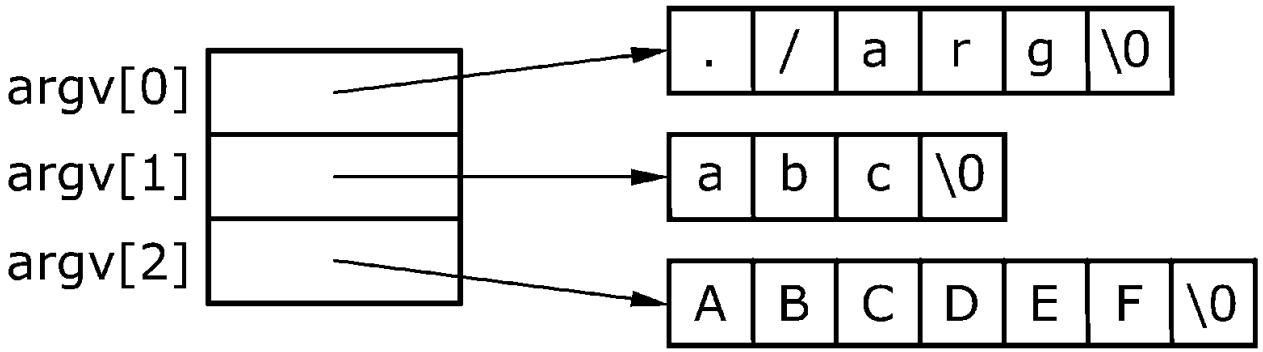


图4-1 main函数的argv参数的存储结构

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

$ cc arg.c -o arg
$ ./arg abc ABCDEF
0: [./arg]
1: [abc]
2: [ABCDEF]
```

$ ./arg *.[ch]

0: [./arg]

1: [arg.c]

2: [auth.c]

3: [ccp.c]

4: [chap.c]

…

$ ./arg '*.[ch]'

0: [./arg]

1: [*.[ch]]

将执行结果与同样的arg.c在DOS下执行结果相比较，在DOS系统中，.\arg　*.c执行时，不会进行文件名展开，执行结果为：

0: [.\arg]

1: [*.c]

4.4 文件管理

4.4.1 ls:文件名列表

列出目录中的文件名。ls之后可以跟0个到多个名字。

- 不给出任何名字时，列出当前目录下所有文件和子目录。
- 名字为文件时，列出文件名。
- 名字为目录时，不列出目录名，而是列出目录下的所有文件和子目录。
- 在同一命令行中可以指定多个名字，这点可以配合shell文件名通配符工作。

例如：

$ ls

（ls未指定名字，列出当前目录下的所有文件和子目录名）

```
arg       bak.d     document   pipe1      xsh2.c
arg.c     config.ap  inc        xsh2
```

$ ls arg.c

（为ls指定一个实参，是普通文件）

```
arg.c
```

$ ls /

（为ls指定一个实参，是目录，列出根目录下的所有文件和目录）

```
TT_DB            ftphome1         lpp            tftpboot
WebSM.pref       ftphome2         mbox           tmp
audit            ftphome3         mnt            u
bin              home             nsmail         unix
bosinst.data     image.data       opt            usr
cdrom            inst.log         proc           var
dead.letter      lab              sbin           websm.log
dev              lib              smit.log       websm.script
etc              lost+found       smit.script    wsmmon.dat
```

$ ls arg*

（为ls指定多个实参，均为普通文件）

arg arg.c

$ ls /usr/include/*

（为ls指定多个实参，既有文件也有目录）

/usr/include/60x_regs.h /usr/include/lvm.h
/usr/include/NLchar.h /usr/include/lvmrec.h
/usr/include/NLctype.h /usr/include/macros.h
/usr/include/NLregexp.h /usr/include/malloc.h
/usr/include/NLxio.h /usr/include/math.h
/usr/include/a.out.h /usr/include/mbstr.h
/usr/include/acl.h /usr/include/memory.h
/usr/include/aio.h /usr/include/mesg.h
⋮

/usr/include/Motif2.1:
Dt Mrm Xm uil

/usr/include/Mrm:
MrmAppl.h MrmDecls.h MrmPublic.h MrmWidget.h MrmosI.h

ls有几十个选项，控制每个文件的列表格式，以及列表的范围包括哪些文件。

（1）-a：列出所有（all）项，包括以句点打头的文件，默认情况下，名字以句点打头的文件不被列出。

```
$ ls -a
.        .cshrc    arg       bak.d     document  pipe1     xsh2.c
..       .profile  arg.c     config.ap inc       xsh2
```

（2）-R：递归地列出碰到的子目录（Recursion）。如：ls -R / 将列出系统中所有文件，这一命令执行时间会很长，列出的内容也很多。

```
$ ls -R .
arg       bak.d     document   pipe1     xsh2.c
arg.c     config.ap  inc        xsh2
./bak.d:
1s.c      cld.c     cld2.c     client.c  clock.c

./document:
manual.pdf  paper.pdf
```

（3）-F：标记（Flag）每个文件。

- 若列出的是目录，就在名字后面缀以/；
- 若列出的是可执行文件，就在名字后面缀以*；
- 若列出的是符号连接文件，就在名字后面缀以@；
- 若列出的是管道文件，则名字后面缀以|；
- 若列出的是普通文件，则名字后面无任何标记。

ls命令允许同时指定多个选项，ls -aF命令就是同时使用两个选项a和F。

```
$ ls -F
arg*       bak.d/     document/   pipe1|     xsh2.c
arg.c      config.ap* inc@        xsh2*
$ ls -aF
./         .profile   bak.d/      inc@       xsh2.c
../        arg*       config.ap*  pipe1|
.cshrc     arg.c      document/   xsh2*
```

（4）-i：列出文件的i节点号。例如：

```
$ ls -i
184323 arg      184393 config.ap 184326 pipe1
184321 arg.c    206873 document  184391 xsh2
184327 bak.d    184325 inc       184392 xsh2.c
```

（5）-d：若实参是目录，则只列其名字（不列内容）。

例如：设当前目录结构如图4-2所示。

```
$ ls *
abc      abc.rpt

abc.dir:
f1  f2
$ ls -d abc*
abc      abc.dir  abc.rpt
```

无参数的ls命令，与ls *执行结果并不同，与ls -d *功能相同。

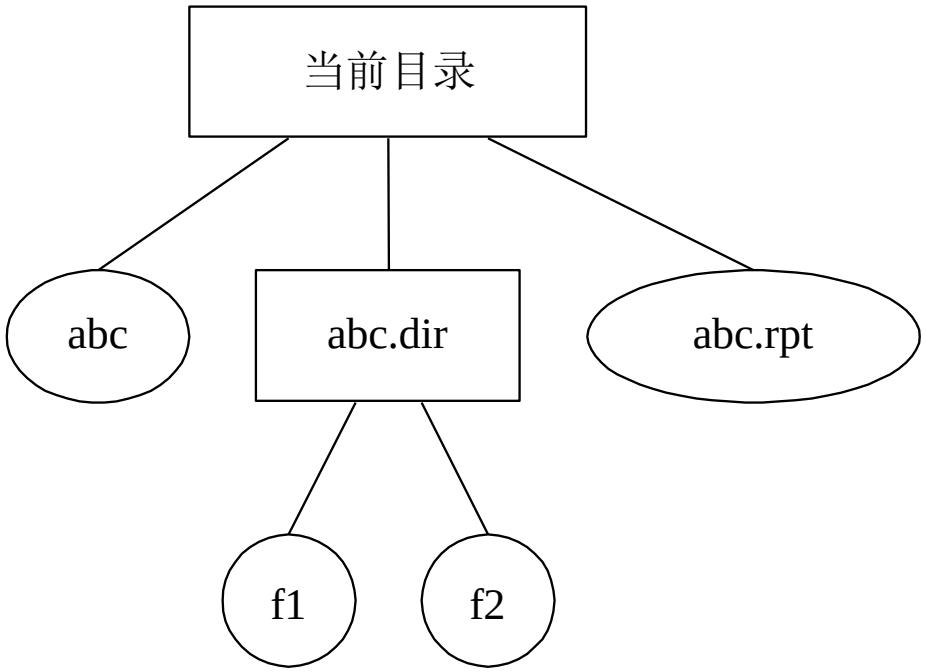


图4-2 文件和目录结构举例，展示ls的-d选项的功能

（6）-l：长格式（long）列表。例如：

$ ls -l

```
-rwxr-xr-x    1 jiang   usr    4423   Aug26  13:31  arg
-rw-r--r--    1 jiang   usr     116   Aug26  13:31  arg.c
drwxr-x---    2 jiang   usr    1536   Aug26  14:00  bak.d
-rwxr-xr-x    1 jiang   usr     128   Aug26  13:38  config.ap
drwxr-x---    2 jiang   usr     512   Aug26  13:59  document
lrwxrwxrwx    1 jiang   usr      12   Aug26  13:33  inc -> /usr/include
prw-r--r--    1 jiang   usr       0   Aug26  13:33  pipe1
-rwxr-xr-x    1 jiang   usr    8661   Aug26  13:36  xsh2
-rw-r--r--    1 jiang   usr    1076   Aug26  13:36  xsh2.c
```

- 第1列为文件属性。

第1列的第1字符描述文件类型。常见的文件类型有：- 普通文件，d目录文件，l 符号连接文件，b块设备文件，c 字符设备文件，p（pipe）命名管道文件。

第1列的第2~10个字符，描述文件的访问权限。其中第2~4个字符描述文件所有者对文件的访问权限；第5~7个字符描述文件所有者的同组用户对文件的访问权限；第8~10个字符描述其他用户对文件的访问权限。权限描述方式为rwx，分别对应读权限,写权限,可执行权限,相应位置显示的是字母，表明有相应权限，显示减号，表明没有相应的权限。

- 第2列是文件的link数，涉及此文件的目录项数。

- 第3列，第4列分别是文件主的名字和组名。
- 第5列是文件的大小。对于普通磁盘文件，列出文件内容大小；对于目录，列出目录表大小（而不是目录下各文件长度之和）；对于符号连接文件，列出符号连接文件自身的长度；对于字符设备和块设备文件，列出主设备号和次设备号；对于管道文件，列出当前管道内的数据长度。
- 第6列是文件最后一次被修改的日期和时间。
- 第7列是文件名。对于符号连接文件，还附带列出符号连接文件的内容。

再如：

ls -l 以长格式列出当前目录下所有文件

ls -l ·列出当前目录下所有文件，可查知各文件权限

ls -ld ·列出当前目录自身，可查知当前目录自身的权限

ls -l * 列出当前目录下所有文件（但不包含目录）和一级子目录中所有文件名

ls -Flad rpt* 可以在同一命令中指定多个选项

ls -l | grep '^d' | wc -l 统计当前目录有多少子目录

4.4.2 cp:复制文件

复制文件的命令格式如下。

格式1: cp *file1* *file2*

格式2: cp *file1* *file2* ... *filen* *dir*

其中，*file1*，*file2*，...，*filen*为文件名（*n*>0），*dir*为已有目录的名字。

- 命令的第一种格式把文件*file1*复制到*file2*。若*file2*存在，则覆盖，否则，创建*file2*。
- cp命令的第二种格式，将*file1*~*filen*一个或多个文件复制到目录*dir*下。当*n*为1时，第二种格式的命令看起来和第一种格式相同，但是完成的操作不同。

例如：

cp *.c backup.d

其中 **backup.d** 为一个子目录,符合第二种格式。

cp fa.c backup.d命令也符合上述的格式2，把文件**fa.c**复制到现有的一个目录**backup.d**中。

【例4-3】 设文件目录结构如图4-3所示，将backup.d下的两个文件p1.c和p2.c复制到当前目录。

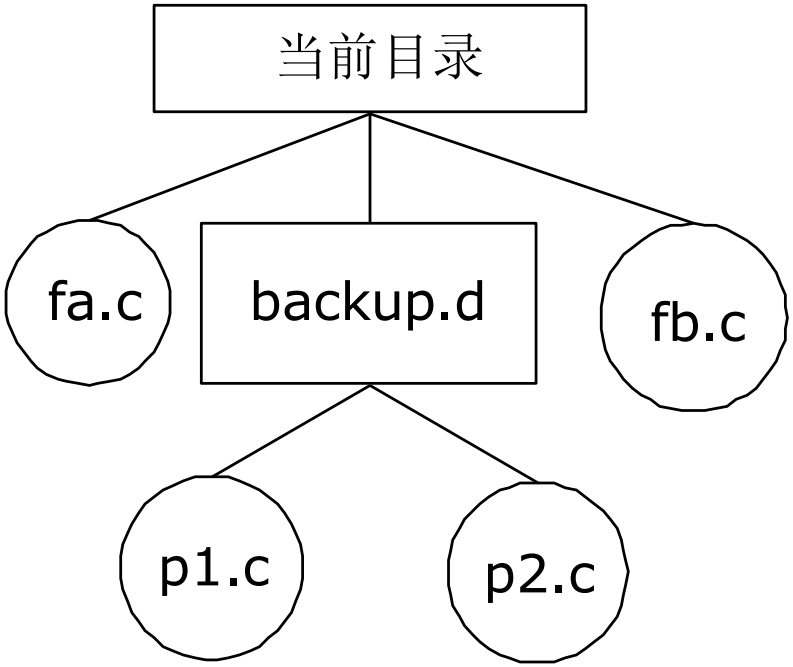


图4-3 文件和目录结构举例，展示cp命令的处理过程

- 在UNIX中执行cp backup.d/p?c命令，经过shell完成文件名生成之后，实际执行：

  cp backup.d/pl.c backup.d/p2.c

  执行结果是p1.c和p2.c没有复制到当前目录,而是文件p1.c覆盖掉文件p2.c。

- 如果运气好的话，backup.d目录下还有p3.c，展开后cp有三个参数，既不符合格式1，也不符合格式2，命令无效，拒绝做任何操作。

- 将上述两文件复制到当前目录的方法，使用cp命令的格式2，最后一个参数句点是当前目录:

  cp backup.d/p?.c .

  再如：

  cp /usr/include/*.h .

4.4.3 mv:移动文件

格式1：mv *existing-file-or-dir new-name*

格式2：mv *file1 file2 ... filen directory*

- mv命令的第一种格式可以将文件和目录改名，或移动到另一个目录，并使用新名字。
- 第二种格式，*directory*是一个已存在的目录，可以把一个或多个已有的文件或目录移动到目录*directory*中，并保持原先的名字。其中的*file1~filen*可以是已有文件，也可以是已有的目录。

在同一个文件系统内的mv操作比先复制成新文件再删除旧文件的方法更高效，因为mv操作只需要修改目录表，不需要访问文件内容。

4.4.4 rm:删除文件

格式：**rm *file1 file2 ... filen***

例如：**rm core a.out**

rm *.o *.tmp

（1）选项-r：递归删除（recursively）。允许***file1~filen***是已有的文件名或者目录名。当它是一个目录时，递归地删除子目录中的所有文件和目录。经常使用这一命令删除一棵已有的目录树。

（2）选项-i：交互方式（interactive）。每次删除前，经过操作员确认。

（3）选项-f：强迫删除（force）。只读文件也可以被删除。

【例4-4】 rm命令选项的功能。

（1）rm -r backap.d

删除当前目录下的整个子目录backup.d。

（2）rm -rf xx*

清除所有以xx打头的文件。程序员应当为一段时间内临时使用的一些文件的取名符合某一特定规律。

（3）rm -i *.test

有选择地删除若干文件。

（4）rm * .bak

误操作，星号之后多出了一个空格，那么，经shell文件名展开之后，rm会忠实地删除所有文件，并且可能会通知用户，企图删除的文件.bak不存在。

【例4-5】处理以减号（-）打头的文件。

设当前目录下只有a，b，c这3个文件。

- 执行命令rm -i：rm会将-i理解为命令选项，没有指定文件名，不能删除任何文件。

执行命令who > -i，将生成文件-i，因为这个名字符合文件命令规则。

- rm –i： 不能删除文件-i。
- 如果使用命令rm *： 文件名通配符展开后的命令变成rm -i a b c，那么，rm将提示删除文件a，b或c，惟独不提示删除文件-i。
- 如果使用命令ls -l *：文件名通配符展开后的命令变成ls -l -i a b c，那么，ls命令以长格式列出a，b，c这3个文件并附带其i节点号，但不列出-i文件。

- **UNIX的许多命令，都允许使用选项，选项是可选择的，并不是每次使用该命令时都需要所有选项。按照惯例，选项都以减号开头，**
- **用一个独立的命令行参数--显式地标志命令行选项的结束，从--参数之后的任何命令行参数，都不再解释为选项。这样命令就可以识别以“-”开头的文件名。**

命令 1、 rm -- -i 删除文件-i。

2、rm ./-i删除当前目录下的文件-i。

4.4.5 find:查找文件

- find命令在一个指定的范围内查找符合条件的文件或者目录，然后执行相应的动作。
- find从指定的路径开始,递归地查找其下属的所有子目录,凡满足条件的文件或目录,执行规定的动作。
- find功能很强，描述“条件”和“动作”选项较多。

例如：

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

对于在规定范围内找到的符合条件的文件，执行的动作是把查找到的文件的路径名打印出来。

1. 关于“条件”的选项

（1）-name文件名的匹配，允许使用文件名通配符*、?和[]，应当把这个文件名通配符描述串传递到find程序中，因此，应当用引号括起来，以免被shell展开。

（2）-type 类型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，如：-type d。

（3）-size ±n[c]文件大小，正号表示大于，负号表示小于，无符号表示等于，其他与数量有关的选项，也采用这样的方式。例如：

-size +100　　表示长度大于100块

-size -100　　长度小于100块

-size +100000c 表示长度大于100000字节

-size -100000c 表示长度小于100000字节

（4）-mtime ±n　文件最近修改（modify）时间，单位为天。例如：

-mtime -10，表示10天之内曾经修改过

（5）-atime ±n　文件最近访问（access）时间，单位为天。文件“访问”指读取了文件的内容，或者文件作为一个程序被执行。仅仅写文件，文件的“访问”时间不变。

- **find还有许多其他的条件选项，可以指定文件主（-user），组（-group），文件的link数（-links），i节点号（-inum）等。**
- **find中可以指定多个条件，罗列出的多个条件默认为多条件的“与”。**
- **find可以用!和-o表示条件“非”和两条件的“或”，可以使用括号表示更复杂的复合条件。**

2．关于“动作”的选项

（1）-print　打印查找到的符合条件的文件的路径名。

（2）-exec，-ok　对查找到的符合条件的文件执行某个命令。

find命令由于可以用-exec和-ok选项引入其他的命令，允许对搜索到的文件执行所需要的操作。多命令的这种组合，使得find命令可以配合其他命令提供灵活又强大的功能。

【例4-6】几个使用find命令的例子。

（1）find . -type d -print

从当前目录开始查找，仅查找目录，找到后，打印路径名。这种方法可以按层次列出当前的目录结构。

（2）find / -name 'stud*' -type d -print

指定了两个条件：名字与stud*匹配，类型为目录。这是两个条件的“逻辑与”，同时符合这两个条件的项目，打印路径名。

（3）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。

（4）find . -atime +30 -mtime +30 -print

从当前目录开始检索最近30天之内既没有读过，也没有写过，而且也没有被当作命令执行过的文件。这种方法可以筛选出一个时间周期内不活跃的文件。

（5）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。

条件的“非”用！。注意，!号与-type之间必须保留一空格。

（6）find / -size +100000c \(-name core -o -name '*.tmp' \) -print

从根目录开始检索那些文件尺寸大于100KB，并且文件名叫core或者文件名有.tmp后缀的文件。在这个命令中，使用了两条件的“或”（-o）及组合（括号）。注意，不要遗漏了所必需的引号、反斜线和空格，尤其是括号前和括号后。上述命令也可以写作下面的形式。

find / -size +100000c '(' -name core -o -name *.tmp ')' -print

find / -size +100000c \(-name core –o -name *.tmp ')' –print

（7）find / -name make -print -exec ls -l {} \;

在-exec及随后的分号之间的内容作为一条命令，上述命令以长格式列出文件。

- 由于在shell中分号有特殊含义，因此，在此前面加\以取消shell对分号的特殊解释，使得find命令可以见到分号。
- {}代表所查到的符合条件的路径名。
- -ok选项和-exec选项类似，只是在执行指定的命令前等待用户确认。再如：

```
find . -name '*[ch]' -exec cat {} \;
find / -size +100000c \( -name core -o name '*.tmp' \) -ok rm {} \;
```

（8）在/usr/include目录下属的所有子目录中的C语言头文件里检索字符串AF_INET6

- **grep无法递归式地搜索一个目录中的所有文件以寻找一个字符串；**
- **find命令的“递归式”穷尽搜索一个子目录，-exec选项又提供了处理的灵活性，使得在find框架下，组合其他的命令。**
- **使用find命令和grep命令的组合，可以达到这样的目的。**

```
find /usr/include -name '*.h' -exec grep AF_INET6 {} \;
```

（9）欲将当前目录下所有文件复制到目录../bak中，可以执行cp * ../bak命令。shell在执行这一命令时，会将*扩展为当前目录下的所有文件名，但是，如果当前目录下文件数目太多，shell在进行文件名展开的时候，展开后的名字太多，有的系统中就会执行失败。这时，可以使用命令：

find . -type f -maxdepth 1 -exec cp {} ../bak \;

选项-maxdepth 1将find的搜索深度限制为最多1层，如果当前目录有子目录，就不再检索子目录。

4.5 目录管理

4.5.1 路径名

- 绝对路径名与相对路径名
- 当前工作目录

  UNIX中，每个进程都有一个当前工作目录。
- 文件.与..
- 主目录

  UNIX是一个多用户系统，每个用户都对应一个主目录，主目录的设置可以从/etc/passwd文件中看到，或用env命令查环境变量HOME的值。

4.5.2 pwd:打印当前工作目录

pwd命令打印当前工作目录的名字（Print Working Directory）。

例：pwd

/root

root 用户当前所在目录是它的主目录 /root

pwd

/home/demo

当前登陆 Linux 系统的是用户 demo，当前所在目录为 demo 的主目录 /home/demo

4.5.3 cd:改变当前工作目录

cd命令用于修改当前工作目录的路径（Change Directory）。例如: cd /usr/include

cd / 进入根目录, 斜线/前必须要有空格。

cd .. 返回上级目录。

cd命令之后未给出任何参数，默认回到用户的主目录。 (cd ~ 进入用户主目录)

- cd命令是shell的一个内部命令。
- 用于修改“shell进程”的进程属性中的当前工作目录。在C语言程序中可以用系统调用chdir函数修改进程的当前目录。

4.5.4 mkdir:创建目录

例如：**mkdir tmp**

mkdir -p sun/work1.d

若**sun**目录原本不存在，则建立一个。

- 若不加 **-p** 参数，且原本**sun**目录不存在，则产生错误。
- **mkdir**除创建目录外，还在所创建的目录中自动建立文件.与..。

4.5.5 rmdir:删除目录

例如：**rmdir sun/workl.d**

rmdir tmp

- 要求被删除的目录除.与..外没有其他文件或目录。否则，**rmdir**命令会失败(删除空目录)。
- 删除一个含有文件或者子目录的目录树，可以直接使用命令**rm**，例如：

rm -r sun/workl.d

参数-r 删除目录及其下所有文件

4.5.6 cp:复制目录

- 选项-r，用于递归（Recursively）地复制一个目录。命令格式为:

  cp -r *dir1 dir2*

（1）若*dir2*不存在，则新建子目录，并将*dir1*下所有文件复制进来。

（2）若*dir2*已存在，则将所有文件复制到目录*dir2*。

- 选项-v，冗长（verbose）方式，执行时列出所复制的文件名。
- 选项是-u，用于增量复制（update），在源文件的修改时间较目的文件更新时，或是名称相互对应的目的文件并不存在，才复制文件。

【例4-7】 目录的复制与增量复制。

（1）复制目录work.d。

cp -r work.d bak.d

（2）增量复制。

设bak.d是work.d的备份目录，将work.d中的内容增量复制到备份目录中。

cp -ruv work.d bak.d

touch命令：可以将文件的最后一次修改时间设置为当前时间，但是不修改文件内容；或创建文件。

例如：

touch *.[ch]

4.6 文件的归档与压缩处理

4.6.1 tar:文件归档

文件归档程序tar最早是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。

命令用法：

tar [ctxu][v][f *device*] *file-list*

选项的第一个字母指定要执行的操作（功能字母）是必须的。

c: 创建（**create**）新磁带。从磁带的头上开始写，以前存于磁带上的数据会被覆盖掉。

t: 列表（**table**）。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，否则，将列出指定的文件。

x: 抽取（**extract**）。从磁带中抽取指定的文件。当不指定文件名时，抽取所有文件。如果磁带上有几个名字相同的文件时，则最后一个文件将覆盖所有较早的同名文件。

u: 更新（**update**）。把文件追加到磁带的尾部，这个文件的某个版本也可能曾经存放到磁带上。为了兼顾磁带设备的顺序访问特点，新版本文件追加到磁带尾部，旧版本文件仍保留在磁带中。

除功能字母外，还可附加其他字符以选用所想要的功能。

v: 冗长（**verbose**）。**tar**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。若用**t**功能，则以长格式列出磁带中的文件名。

f: 指定设备文件名（**file**）。

【例4-8】使用tar命令的例子。

（1）tar cvf /dev/rct0 .

将从当前目录开始的整个目录树，备份到设备/dev/rct0中，句点目录是当前目录。

（2）tar xvf /dev/rct0

将磁带设备/dev/rct0上的数据恢复到文件系统中。

（3）tar tvf /dev/rct0

打印磁带设备/dev/rct0上的文件目录。

（4）tar uf /dev/rct0 test.c

磁带备份完成之后，文件test.c又发生了变化，将更新过的test.c追加到磁带尾部。

（5）tar cv *

将当前目录下所有文件备份到默认的设备中。

（6）tar命令可以指定一个普通文件代替设备文件，可将多个文件或一个目录树存储成一个文件。

tar cvf my.tar *.[ch] makefile

将多个文件存成单一的**my.tar**文件。

tar cvf work1.tar work1

其中，**work1**是一个复杂的子目录，有多个目录层次。结果，打包成一个文件**work1.tar**。

（7）tar xvf work1.tar

从归档文件中恢复数据。

4.6.2 compress:文件压缩

采用LZW算法对文件压缩，普通文本文件可压缩掉50%~80%，这是一种字典压缩算法。压缩算法对于那些文件中的数据很有规律的内容压缩效率很高，有许多字段是空白的一些数据库文件压缩后文件体积甚至可以减少90%以上。

例如：

compress chapt5　压缩，生成新文件**chapt5.Z**

zcat chapt5.Z　　读取压缩格式的文件

uncompress chapt5.Z 解压缩，还原文件**chapt5**

4.6.3 应用

文件归档tar与压缩处理compress在不同的UNIX系统之间有通用性。因此,可以使用这些方法在不同UNIX中交换程序或数据。

【例4-9】 使用tar命令和compress命令通过网络复制一个目录树。

在主机A:

tar cvf xapi.tar xapi

将整个xapi目录树，存到一个文件xapi.tar中。

compress xapi.tar

压缩生成文件xapi.tar.Z。

将文件xapi.tar.Z通过网络传到主机B（用ftp或者E-mail）。

在主机B上执行下面的操作：

uncompress xapi.tar.Z

tar xvf xapi.tar

用这种方法可以把整个一棵目录树复制到另一台UNIX主机上。

4.7 文件系统的存储结构

4.7.1 基本文件系统与子文件系统

- UNIX的整个文件系统分成基本文件系统（也叫根文件系统root file system）和子文件系统。
- 基本文件系统是整个文件系统的基础，不能“脱卸（umount）”。子文件系统以基本文件系统中某一子目录的身份出现，包括用作子文件系统的硬盘、软盘、USB盘、CD-ROM网络文件系统NFS等。
- 根文件系统和子文件系统都有自己独立的一套存储结构和目录结构，甚至文件系统的格式都不一样。

【例4-10】文件系统的创建和安装。

- 创建文件系统的命令**mkfs**

  mkfs /dev/fd0135ds18

  块设备文件**/dev/fd0135ds18**指3.5英寸容量1.44MB的A盘。

- 安装一个子文件系统的命令是**mount**

  例如：**mount /dev/fd0135ds18 /mnt**

  mount的第二个参数**/mnt**可以是任一个事先建好的空目录名，允许处于根文件系统的任何目录中。

- 不带参数的**mount**命令列出当前所有的子文件系统。

下面的命令都操作软盘设备。

cp /usr/include/*.h /mnt

rm /mnt/stdio.h

mkdir /mnt/jiang

cp /usr/jiang/*.[ch] /mnt/jiang

vi /mnt/jiang/fn.c

ls /mnt

- 拆除一个已安装的子文件系统**umount**命令。下面的命令使得软盘A不再与/mnt目录有联系。

  umount /dev/fd0135ds18

- **UNIX**的设备文件分字符设备文件和块设备文件，只有块设备文件，才可以使用**mkfs**命令创建文件系统和使用**mount**命令安装到根文件系统中。

【例4-11】网络文件系统NFS的安装。

建立一个网络文件系统。设有两台主机，主机C和主机S，两主机上均已安装好了NFS软件包，主机C期望共享主机S上的文件目录/usr/jiang。在主机S的文件/etc/exports中添加行/usr/jiang。然后在主机C上,执行下面的命令：

mount -f NFS 203.123.54.189:/usr/jiang /xbg

203.123.54.189是主机S的地址，/xbg是事先在主机C上已建好的空目录。这样，主机C上访问的文件/xbg/makefile，实际上访问的是主机S上的文件/usr/jiang/makefile。

【例4-12】 Linux系统中引用Windows格式的磁盘分区的例子。

在Linux系统中，事先建立空文件目录/a，/c，/d，/u，/cdrom。分区1是FAT32格式的C盘，分区4是FAT32格式的扩展D盘。

```
mount /dev/fd0 /a
mount /dev/hdc1 /c
mount /dev/hdc4 /d
mount /dev/cdrom /cdrom
mount /dev/sda1 /u
```

Linux可以自动识别并支持FAT32格式的Windows文件系统。设备/dev/cdrom是CD-ROM盘，设备/dev/sda1是USB接口Flash盘,设备/dev/fd0是软驱A，设备/dev/hdc1是硬盘分区1上的C盘，设备/dev/hdc4是硬盘分区4上的D盘。

4.7.2 文件系统的结构

- 对块设备的特定编号的块的读取或者写入请求，需要映射到对于设备有意义的参数，即：这个块对应的磁盘上的磁道号、柱面号、扇区号，并且启动相应的控制器完成读写操作，这些，都是块设备驱动程序的任务。
- 文件系统使用块设备，要求所有不同种类的设备提供相同的界面。
  - 文件系统的设计者不再关注硬件驱动器，物理介质等方面的差异。
  - 设备提供商，只需要提供设备驱动程序就可以用文件系统管理它的设备上的信息。

- **UNIX的每个逻辑块设备，如：硬盘的一个分区，一张软盘，USB接口的Flash盘，光盘等，都对应一个块设备文件，如：/dev/hdc4，/dev/sda1，/dev/cdrom等，在每个逻辑设备上构造一个独立的子文件系统。**
- **系统在这个设备上创建UNIX格式的子文件系统的时候，把整个逻辑设备以“块”为单位划分，编号为0，1，2，...。**
- **磁盘设备读写的最小单位是“扇区”。典型的，一个扇区会有512B，逻辑设备的“块”一般是一个扇区，或者2^n个扇区，这样，一块可能会是512B，1024B，4096B，……。**
- **文件系统使用设备时，空间分配或释放的最小单位是“块”。例如，块大小为4096B的系统中，存放大小4097B的信息，需要两块。**

一个文件系统由下述几部分构成，参见图4-4。

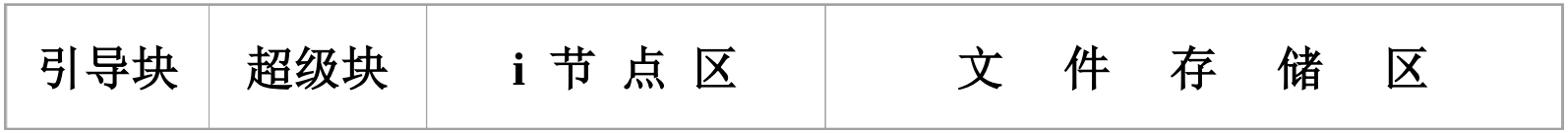


图4-4 文件系统的布局

（1）引导块（Boot block）：0号块。用于启动系统，存放引导程序，它含有的程序代码用于系统启动时引导执行操作系统的内核。

（2）超级块（Super block）：1号块, 也叫管理块。存放与整个文件系统的管理有关的信息。

（3）i节点区：i节点（index node），简记为i-node。i节点区由若干块构成，专用于存放i节点。

- 系统中的每个文件都对应一个i节点。每块可容多个i节点，每个i节点有固定大小。i节点中包含“索引”信息和文件属性信息。注意：i节点内不含有文件的文件名。
- 在使用命令mkfs创建文件系统时，根据整个块设备的大小，i节点区大小由系统管理员自行指定，或者采用默认的大小。
- i节点编号从1开始。1，2，3，...，。

（4）文件存储区：用于存放文件中数据的区域，除了普通磁盘文件之外，还包括目录表。一个存储设备的文件存储区占整个存储空间的绝大部分。

4.7.3 目录结构

- 每个目录表在UNIX中也被组织得跟一个普通文件一样，存于“文件存储区”中，有其自己的i节点。
- 用ls命令列出的目录的大小是目录表本身的长度，而不是目录下所有文件的长度总和。
- 目录表的基本组成单位是“目录项”，每个目录项由一个“文件名-i节点号”对构成。

在图4-5的例子中，一个文件可以有多个名字。只要在不同的目录项中填写“文件名-i节点号”的时候将i节点号填写的相同就可以了。

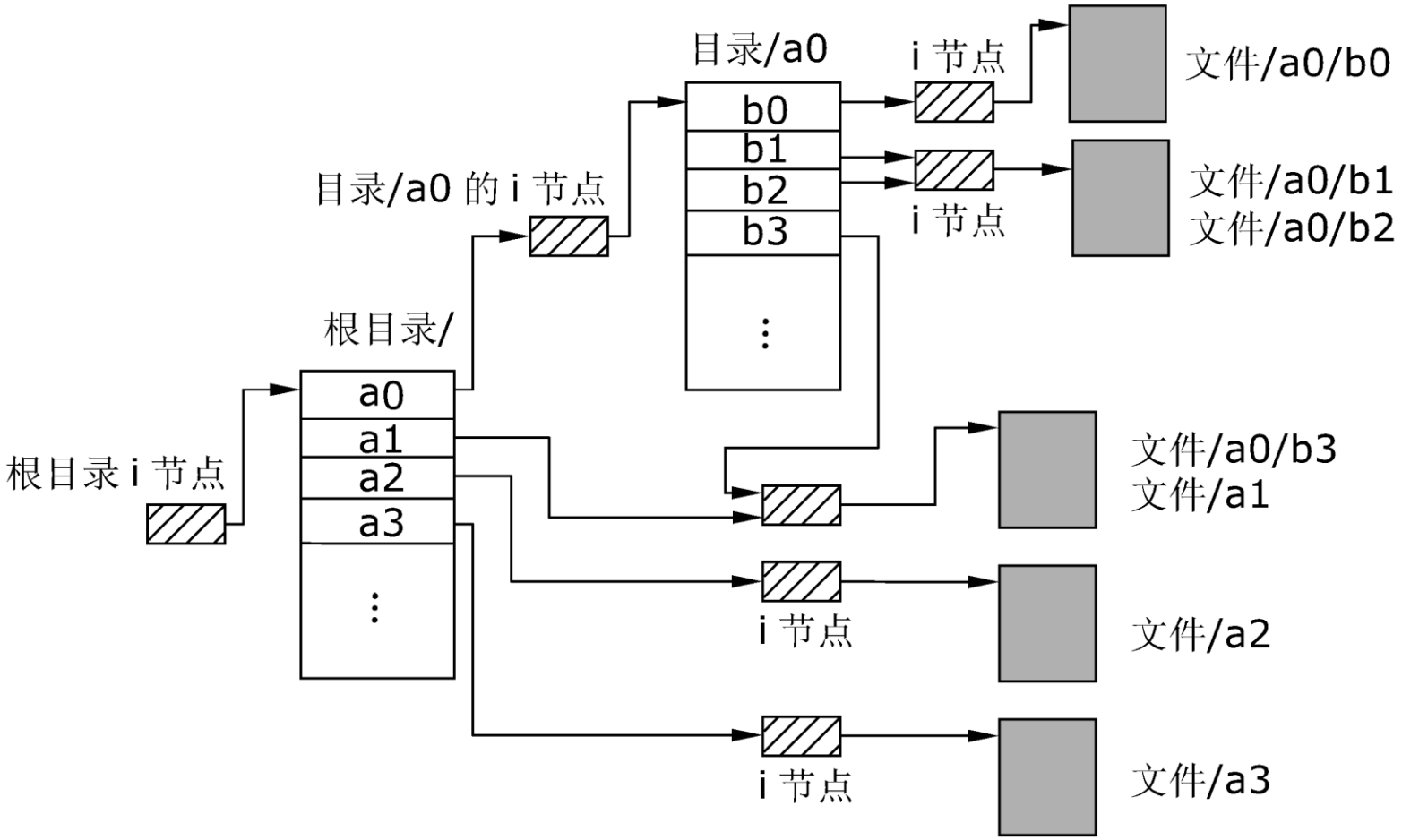


图4-5 文件系统目录表的构造

- 命令ls的-i选项，可以列出文件的i节点号。如果两个文件的i节点号相等，它们就引用同一个文件。
- 在ls -l命令时，每个文件列出的项目之一是“文件的link数”，就是同一个i节点被引用的次数，可以判断出文件被多少个目录项所引用。。
- link数被记录在i节点中，便于文件的删除操作。
- 根据i节点号，find命令的-inum选项可以查找指定i节点号的文件。

4.7.4 命令df与du

1. df:显示可利用的磁盘空间

- df命令用于显示文件系统的空闲空间（disk free space）。df可以列出每个子文件系统的设备文件名，mount安装的路径，文件存储区和i节点区的总长度，空闲空间和百分比。
- 不同的系统df的显示不完全相同，常用选项有-v和-i。-v选项使得系统显示更多的信息，-i显示与i节点有关的信息。

【例4-13】使用df命令的例子。

```
$ df
Filesystem    512-blocks      Free %Used    Mounted on
/dev/hd4        2359296   2167496    9%     /
/dev/hd2       15007744   8842688   42%     /usr
/dev/hd11     128188416 107533208   17%     /home/malaba
$ df -i
Filesystem   Iused    Ifree %Iused Mounted on
/dev/hd4      2104   587720     1% /
/dev/hd2     75444  1800524     5% /usr
/dev/hd11    11441 16012111     1% /home/malaba
```

df在列出文件系统的文件存储区时，含有总块数和空闲块数，根据系统的不同，一块可能是512B，或者其他数目。

2. du:显示磁盘使用信息

du命令（disk usage）显示包括所有下属子目录在内的某一目录树中文件使用的块数总和，显示单位仍然是“块”。

【例4-14】使用du命令的例子。

```
$ du  /etc
8       /etc/rc.d/rc2.d
8       /etc/rc.d/rc3.d
8       /etc/rc.d/rc4.d
8       /etc/rc.d/rc5.d
8       /etc/rc.d/rc6.d
8       /etc/rc.d/rc7.d
8       /etc/rc.d/rc8.d
8       /etc/rc.d/rc9.d
32      /etc/rc.d/samples
```

```
112     /etc/rc.d
272     /etc/methods
24      /etc/locks
8       /etc/snmpinterfaces
8       /etc/rpm
16      /etc/vatools/vac
16      /etc/vatools/vacpp
48      /etc/vatools
112     /etc/vacpp/C
112     /etc/vacpp/en_US
232     /etc/vacpp
3408    /etc
```

对/etc下每一个下属子目录，都列出其下属文件所占用磁盘空间的块数，最后列出一个总合计。

4.8 硬连接与符号连接

4.8.1 硬连接

1. 硬连接的定义

存于文件存储区的每一个文件都有一个i节点。目录表由目录项构成，目录项就是一个“文件名-i节点号”对。每个目录项指定的文件名-i节点号的映射关系，叫做硬连接。

硬连接数目（link数）：同一i节点被目录项引用的次数。

2. 命令ln

ln命令（link）可以用来创建一个硬连接。

【例4-15】 普通数据文件的硬连接。

```
$ ls -l chapt0
-rw-r--r--   1 kc   kermit        54332 Jun 01 12:28 chapt0
$ ln chapt0 intro
$ ls -l intro chapt0
-rw-r--r--   2 kc   kermit        54332 Jun 01 12:28 chapt0
-rw-r--r--   2 kc   kermit        54332 Jun 01 12:28 intro
```

$ ls -i intro chapt0

88077 chapt0

88077 intro

$ vi intro （修改文件中的内容并存盘）

$ ls -li intro chapt0

88077 -rw-r--r-- 2 kc kermit 54140 Jun 01 12:30 chapt0

88077 -rw-r--r-- 2 kc kermit 54140 Jun 01 12:30 intro

$ rm chapt0

$ ls -l intro

-rw-r--r-- 1 kc kermit 54140 Jun 01 12:30 intro

【例4-16】 程序文件的硬连接。

用ln可以为程序文件建立硬连接。

```
$ ln arg arg1
$ ls -li arg*
88090 -rwxr-xr-x   2 kc   kermit        11628 Jun 01 09:27 arg
88090 -rwxr-xr-x   2 kc   kermit        11628 Jun 01 09:27 arg1
87302 -rwxr--r--   1 kc   kermit          164 Jun 01 09:20 arg.c
$ ./arg abc ABCDEF
0: [./arg]
1: [abc]
2: [ABCDEF]
$ ./arg1 abc ABCDEF
0: [./arg1]
1: [abc]
2: [ABCDEF]
```

```
$ pwd
/usr/bin
$ ls -li grep [ef]grep
30997 -r-xr-xr-x   3 bin      bin         20434 Feb 11 2002  egrep
30997 -r-xr-xr-x   3 bin      bin         20434 Feb 11 2002  fgrep
30997 -r-xr-xr-x   3 bin      bin         20434 Feb 11 2002  grep
$ ls -li sh [kpt]sh
31015 -r-xr-xr-x   4 bin      bin        241436 Oct 24 2001  ksh
31015 -r-xr-xr-x   4 bin      bin        241436 Oct 24 2001  psh
31015 -r-xr-xr-x   4 bin      bin        241436 Oct 24 2001  sh
31015 -r-xr-xr-x   4 bin      bin        241436 Oct 24 2001  tsh
$ ls -li vi
30991 -r-xr-xr-x   5 bin      bin        232114 Sep 24 2001  vi
$ find . -inum 30991 -exec ls -li {} \;
30991 -r-xr-xr-x   5 bin      bin        232114 Sep 24 2001  ./ex
30991 -r-xr-xr-x   5 bin      bin        232114 Sep 24 2001  ./edit
30991 -r-xr-xr-x   5 bin      bin        232114 Sep 24 2001  ./vedit
30991 -r-xr-xr-x   5 bin      bin        232114 Sep 24 2001  ./vi
30991 -r-xr-xr-x   5 bin      bin        232114 Sep 24 2001  ./view
```

- 用ln建立硬连接时，只限于文件，两个文件名的路径名也必须处于同一文件系统中。
- 不允许对目录用ln命令建立硬连接，从文件系统的存储结构上看，建立这种硬连接很容易实现，但是，如果对目录允许这样的操作，就会破坏目录的树型结构，使得目录结构变成网状结构。因此，对磁盘目录的硬连接操作被禁止。

3. 目录表的硬连接

- 尽管不允许用户对目录使用ln命令建立硬连接，但是，创建目录的操作，操作系统内核会自动在所创建的目录中创建.和..文件，使得目录的硬连接数目也不再是 1。图4-6示例出一个目录结构和这个目录结构对应的硬连接情况。

- 一般来说，对于一个普通目录，它的link数等于直属子目录数加2。

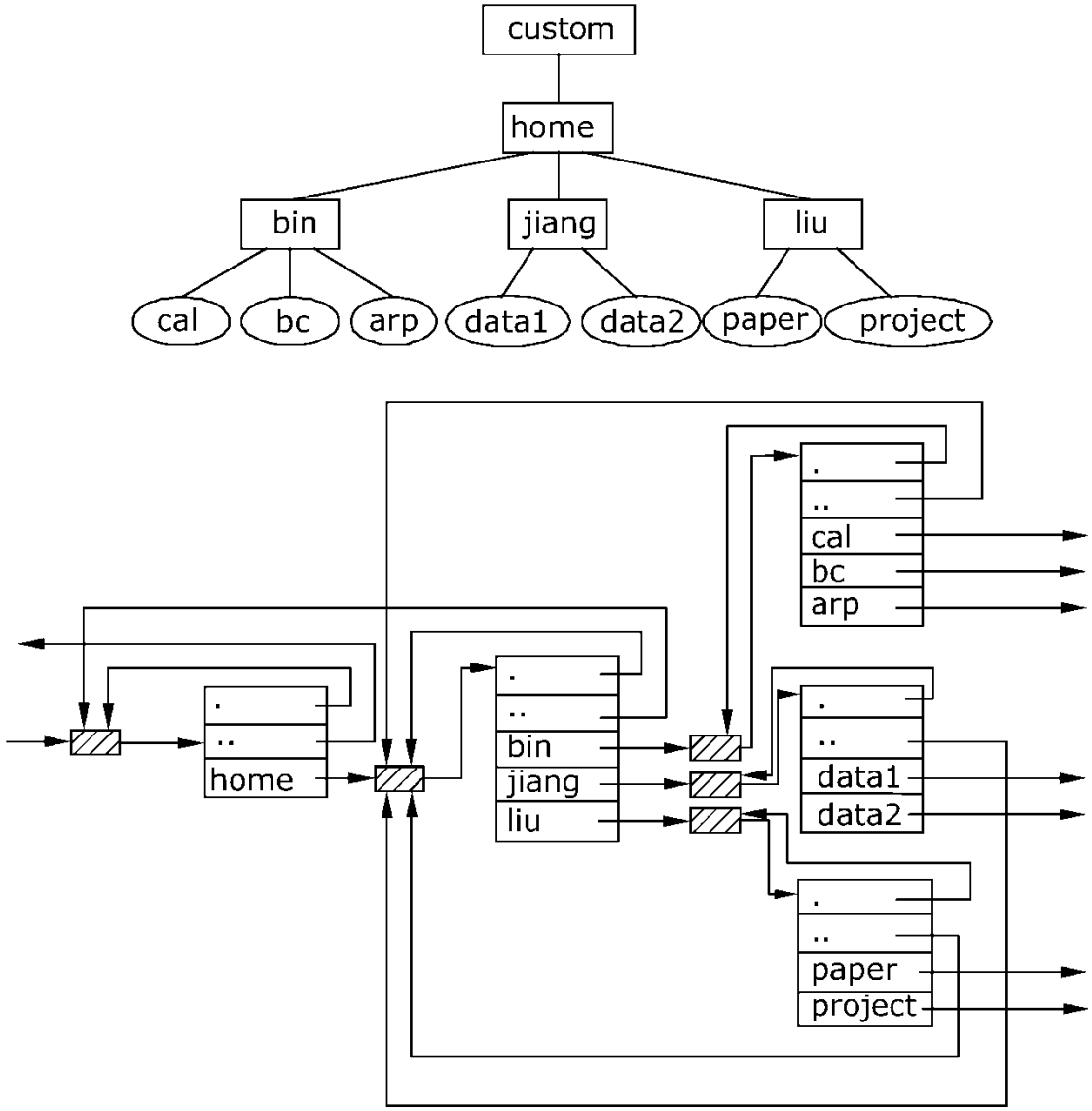


图4-6 目录文件的硬连接

4.8.2 符号连接

- 符号连接也叫软连接，最早在BSD UNIX中实现。
- 在UNIX系统中，用一个特殊文件“符号连接文件”来实现符号连接。在“符号连接文件”中仅包括了一个描述路径名的字符串。
- 创建符号连接的命令是使用带-s选项（Symbol link）的ln命令。类似建立硬连接的命令，符号参数在前，新建的文件名在后。

【例4-17】 符号连接的使用举例。

$ who > users_on

$ ln –s users_on active_users

$ ls –l active_users

lrwxrwxrwx 1 fang kermit 8 Jul 26 16:57 active_users->users_on

（可以使用下面的命令，列出当前目录树中所有的符号连接文件：）

$ ls -lR | grep -- '->'

$ find . -exec ls -l {} \; | grep -- '->'

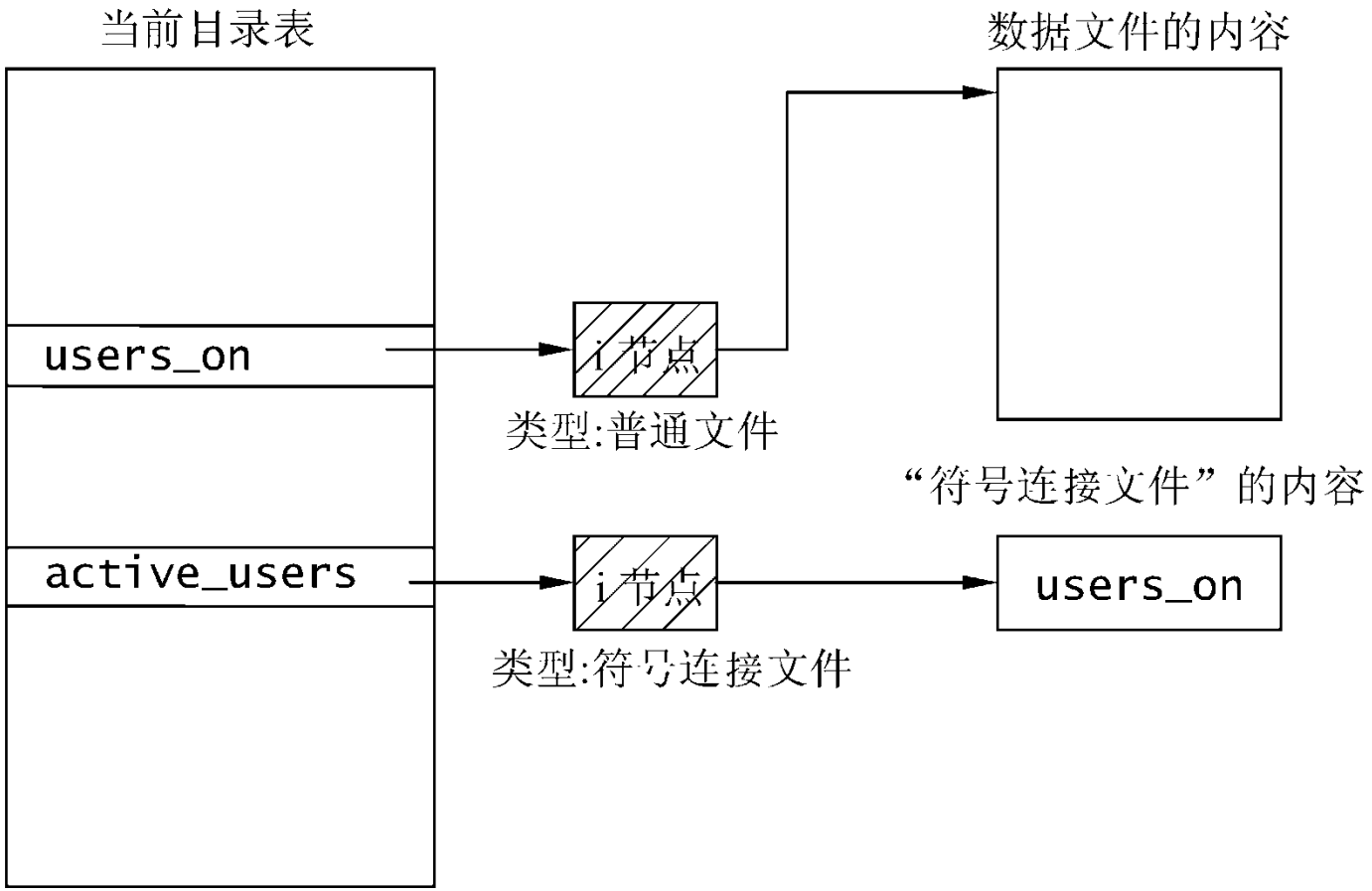
$ cat active_users

fang tty00 Jul 5 14:49

sun tty01 Jul 5 11:31

liang tty03 Jul 5 15:50

dong tty11 Jul 5 09:45


当前目录表
数据文件的内容
users_on
i 节点
类型:普通文件
“符号连接文件”的内容
active_users
i 节点
users_on
类型:符号连接文件

- 将active_users展开成该符号连接文件存储的字符串users_on的操作，不是由shell完成，而是由操作系统内核的文件管理模块完成。
- 操作系统内核模块处理符号连接的方法是，open，creat等系统调用，提供一个文件的路径名。在逐个翻译路径名中用斜线分割开的路径分量时，若系统发现符号连接，就把符号连接的内容加到路径名的剩余部分的前面，翻译这个名字产生结果路径名。若符号连接包含绝对路径名，使用绝对路径名。否则，根据文件层次结构中该连接的位置（不是根据调用进程的当前的工作目录）和符号连接的内容，继续分析路径名。

【例4-18】 符号连接的展开方式。

设/a/b/c是一个符号连接文件内容为p/g/r，不是绝对路径，那么，访问文件/a/b/c/d/e 则实际访问/a/b/p/g/r/d/e。文件c记录了符号p/g/r，处理时将符号粘贴替换到路径分量c的相应位置。

设/a/b/c是一个符号连接文件，内容为/p/g/r，是绝对路径，那么，访问文件/a/b/c/d/e 则实际访问/p/g/r/d/e。

【例4-19】 符号连接与硬连接的区别。

设目录结构如图4-8所示，当前目录为d。

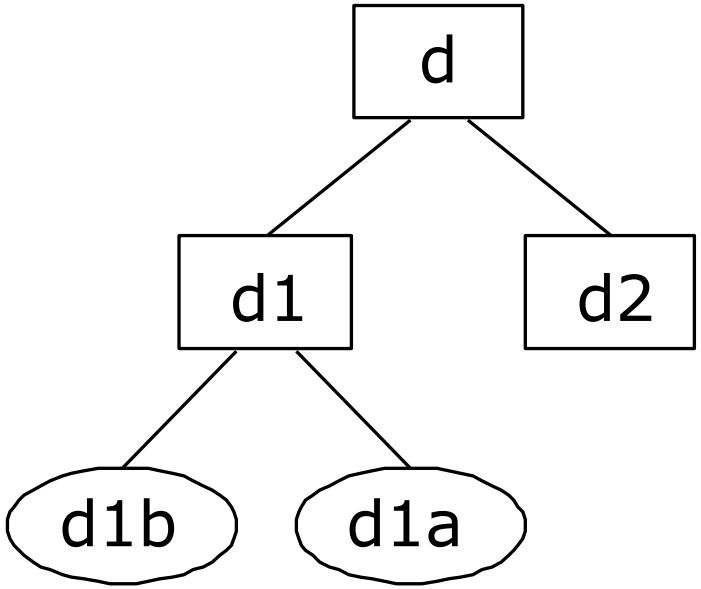


图4-8 目录结构举例

（1）ln -s d1/d1b d1/dx

在d1目录下新建文件dx，在创建d1/dx符号连接文件的时候，系统原封不动地将这个符号字符串存储在文件d1/dx对应的文件存储区中。只有在打开文件d1/dx读写访问时，操作系统的文件系统才进行符号连接处理，实际访问d1/d1/d1b，而不是d1/d1b。如果这样的文件不存在，访问会失败。

（2）ln d1/d1b d1/dx

在d1目录下新建文件dx。硬连接的处理方式是检索出文件d1/d1b的i节点号，在d1目录下新创建的目录项dx使用这个i节点号，硬连接操作完成。如果d1/d1b文件事先不存在，硬连接就会失败。访问文件d1/dx和访问d1/d1b是等价的。在这点上，符号连接和硬连接的处理方式不同。

【例4-20】 观察UNIX命令对符号连接文件的操作。

（1）创建一个符号连接文件

ln -s users_on active_users

无论事先文件users_on是否存在，总可以建立符号连接文件active_users。

（2）读

读取active_users，例如cat active_users，实际上读取文件users_on。文件users_on如果不存在，读操作就会失败。

（3）写

用vi编辑器改写active_users，实际改写了文件users_on。文件users_on的最后一次修改时间也随之变化。而ls -l active_users的最后一次修改时间不变。执行命令touch active_users同样改变users_on的最后一次修改时间。

（4）覆盖文件

使用命令who > active_users，则实际上文件users_on被覆盖。如果文件users_on事先不存在，那么就会创建新的文件users_on。

（5）执行硬连接

执行ln active_users file2，那么file2的i节点号与文件users_on相同。

（6）修改文件的权限

执行chmod a+w active_users，那么文件users_on的权限发生变化。

（7）删除文件

执行rm active_users，则实际上符号连接文件被删除，文件users_on不变。

总之，一旦建立了符号连接,只有删除操作删除的是符号连接文件自己。在符号连接存在期间，所有对符号连接的操作都将访问符号连接所引用的文件,而不是符号连接文件本身。

【例4-21】 符号连接与硬连接的比较，符号连接与目录层次的关系。

对图4-9的文件层次结构，设当前目录为d。

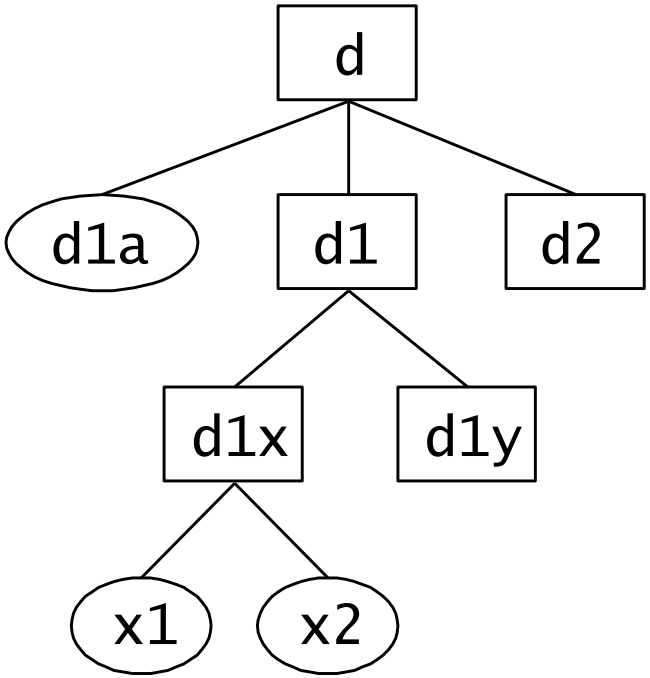


图4-9 目录结构举例

（1）ln -s d1a dx

cat dx实际访问d1a，如果rm d1a后，cat dx仍然试图访问d1a文件。由于文件d1a不存在，会出错。反过来，如果rm dx，那么，对d1a没有任何影响。

（2）ln d1a dx

rm d1a后，使用硬连接cat dx仍能访问以前的文件。

（3） ln -s d1/d1x dx

cd dx后当前目录变为d1x，再执行cd ..，则当前目录变为d1而不是d。

4.8.3 硬连接与符号连接的比较和应用

- 硬连接利用了文件系统内部的存储结构，很简单地实现了信息共享。
- 符号连接依靠操作系统中文件系统模块的软件处理，实现了信息共享。
- 硬连接只适用于文件，不允许用户通过命令对目录实现硬连接，以保障文件系统中目录的树形结构不被破坏。
- 每个文件系统有自己独立的一套i节点，因此，不同子文件系统之间，不可能用硬连接实现文件的共享。

- 硬连接能够完成的功能，使用符号连接都可以做到。符号连接在操作系统内核里用软件实现，同硬连接相比要多占用系统的一部分开销，包括软件处理占用的CPU时间和调入符号连接文件需要的磁盘操作的时间。
- 根据符号连接实现的原理，符号连接完全可以适用于目录，也可以将符号连接文件和符号连接引用的目的文件安排在不同的文件系统中，从而比硬连接更灵活。
- 由于符号连接可以用于任意的目录，目录下的文件可以是符号连接文件，当然可以是任意目录，如果是它的上级目录，就完全可能构造出符号连接的死循环。

【例4-22】 使用硬连接和符号连接共享数据文件以节约存储空间。

在家庭相册中有几百个照片文件，每个文件都有大约700KB。可以为这些文件建立多个目录。使用符号连接或者硬连接都能做到两个人合影的同一张照片，同时处在某个年度目录中，两个不同的家庭成员目录中，还可以放入某次活动的专题目录中。这样，同一个文件，都可以从4个目录中检索到，而不需要将一个文件复制4份。使用符号连接和硬连接还有些区别，使用符号连接时，删除某个700KB的文件，系统就会释放出700KB的空间，可能会导致某些引用这个文件的符号连接不再能正常打开文件；但是，使用硬连接，需要删除所有的相关目录项才能够释放文件的存储空间。

【例4-23】 使用符号连接对程序和用户透明地变更目录组织结构。

UNIX系统中有/usr/adm目录，一般用于存储一些系统软件的日志文件，/usr/tmp可以存放一些临时文件，/usr/spool存放系统收到但用户未读的邮件和打印任务队列。如果系统管理员需要对目录系统重新进行规划，希望这些文件不再存储在/usr目录中，而是把这些每日都会变化的数据信息存放到新建立的/var目录下，用/var/adm、/var/tmp、/var/spool分别代替原先的目录。创建三个符号连接：

ln -s /var/adm /usr/adm

ln -s /var/tmp /usr/tmp

ln -s /var/spool /usr/spool

4.9 系 统 调 用

- 系统调用（System call）是一个专用名词，是操作系统的内核提供的编程界面。
- 使用C语言时，提供一些C语言的函数。通过这些函数，调用操作系统提供的功能。
- 系统调用是应用程序和操作系统进行交互的惟一手段，应用程序只能通过系统调用，才能够申请和访问硬件资源。前面介绍的命令，都使用了UNIX的系统调用，都是运行在用户态的程序。

- 系统调用函数，例如用于文件操作的open、read、write和close等，和普通的函数库调用，例如字符串操作的strcpy和strlen等，从C语言程序员的角度看起来用法一样，都是调用一个函数，但是，处理却不一样。
- 系统调用的函数体一般非常简单，依赖于具体的操作系统，它准备好系统调用所需要的参数之后，立刻产生一个软件中断，系统从用户态进入到内核状态，函数的功能由操作系统的内核代码实现。
- 普通函数库如同程序员自己编写的程序一样，通过函数调用，利用程序的堆栈，在用户态完成函数内部的处理后返回。有些库函数，例如缓冲文件操作的fopen、scanf、printf和fclose等，在函数里完成一些必要处理之后再引用系统调用。

- 为了便于不同UNIX的系统之间C语言源程序之间的移植，UNIX的系统调用函数的名字，参数排列顺序，参数的类型定义，返回值的类型，以及实现的功能，都属于POSIX标准来规范的内容。
- UNIX系统的内核界面非常简捷。Windows也支持大部分的POSIX函数标准，所以，许多UNIX的系统调用函数在Windows系统下也可以使用，Windows提供了功能等价的函数。

- UNIX的系统调用一般都返回一个整数值。为便于记忆，在errno.h头文件中定义了许多宏，用名字代替这些整数值。这些宏都有E前缀，例如：EACCESS、EIO、ENOMEM等，对每个系统调用来说，在手册页中都列出可能的出错情况和对应的错误代码的宏名字。
- errno便于用程序识别错误原因，但是，这样的数字代码很不便于操作员理解失败原因。库函数strerror可以将数字形式的错误代码，转换成一个可以供操作员阅读的字符串，这是一个重要的函数。

  char *strerror(int *errno*);

- 与系统调用执行失败有关的另一个函数是perror()，函数原型为：

  void perror(char **string*);

4.10 文件和目录的访问

4.10.1 文件存取

- 操作系统的文件管理模块的主要功能是提供了一种便捷的手段管理存储在外存中的数据。
- 基本方法是，将用户看来逻辑上相关的一组数据组织在一起，叫做“文件”，系统提供了便于记忆的“文件名”，应用程序就可以根据文件名来访问存储在外存中的数据。
- 操作系统的文件管理模块，负责管理外存的分配和释放，负责检索出文件中数据在磁盘中的存储位置。
- UNIX操作系统内核提供给应用程序员的文件，看起来是前后相继的连续的无格式字节流。

1. 文件描述符

例：文件中有1000个数据，每个数据512B，需要顺序的读出这1000个数据。类似下面的函数界面：

get_data(char **filename*, int *position*, void **buf*, int *len*);

每次这个函数执行时，就需要执行一系列的操作：

① 根据函数参数提供的文件名字符串，查找目录，得到文件对应的i节点号；

② 从文件系统的i节点区调入i节点；

③ 判断用户对文件有没有读操作权限；

④ 再根据*position*，查阅i节点里存放的索引表，得到文件的这个数据块在磁盘上的位置；

⑤ 从磁盘上读入这个数据块，将数据返回给应用程序的*buf*缓冲区。

- 所谓的文件“打开”操作，就是完成包括上述的①②③在内的文件访问前的准备工作，把i节点等数据读入到内核中。
- 在内核中还可以维持一个文件的访问位置的指针，这样用户在连续访问文件的时候就不用再提供***position***参数了。
- 为了使应用程序可以在一个文件访问未完成的时候，同时访问别的文件，当多个文件交叉访问时，操作系统内核必须知道到底要访问哪个文件。需要对已“打开”的文件建立一个索引。

- 对于计算机来说，处理一个字符串结构的名字远不如处理一个整数值更有效，所以，这个索引是个整数，叫做“文件描述符”。内核根据它很容易定位到相关的诸如文件当前读写位置和文件i节点的信息。
- UNIX系统中，每个进程有自己独立的一套文件描述符，按照UNIX和C语言的惯例，文件描述符是从0开始编号的整数，它是操作系统内核中进程PCB（process control block，进程控制块）结构里的一个数组的下标。
- 从此以后，再操作文件时，多次的read或write直接提供“文件描述符”作参数而不是文件名，内核就可以做到高效地访问文件。
- 文件访问结束之后，没有必要再保留那些信息，于是，可以执行所谓的“关闭”操作，释放这些信息占用的存储空间。

打开文件的系统调用open有两种格式：

#include <fcntl.h>

int open(char **filename*, int *flags*);

int open(char **filename*, int *flags*, mode_t *mode*);

打开方式	作　　用
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_CREAT	如果提供的文件名对应文件已存在，该位无意义。否则，创建新文件，只有这时候，open 的第三个参数才有意义。第三个参数，是创建新文件时使用的访问权限代码
O_TRUNC	如果文件原先有数据，打开时截为零。其实就是删光了原先的数据
O_APPEND	每次向文件中写入数据，都写到文件尾。这个标志很有用，尤其是用于向已有的日志文件中追加数据，或者记账和计费数据。这个标志可以用于多个并发的进程都想写在同一个指定文件尾的情形。由于多个并发进程会有自己独立的当前写位置，使用这个标志打开文件，就可以做到每次写文件之前，首先寻找到文件尾，达到多个并发进程按时间累计数据的效果。为了达到这样的效果，每个进程都必须按这种方式打开文件

- 为了使终端的I/O更方便，系统在启动一个程序时自动为程序准备好了三个文件描述符，分别是0、1、2，分别称作标准输入，标准输出，标准错误输出。
- 所有这三个文件默认地指向当前终端。
- 程序不必要执行open就可以直接使用这三个文件描述符。
- 0是只读方式打开的文件描述符，对应当前终端的键盘输入；1和2是只写方式打开的文件描述符，对应终端的屏幕输出。如果程序用open打开其他文件，就会有文件描述符3、4等。

2. 读写文件

读写文件的系统调用分别是read和write。

int write(int *fd*, void **buf*, int *len*);

int read(int *fd*, void **buf*, int *len*);

- write返回值是真正写入的字节数。一般情况下，如果该字节数小于*len*,则说明发生了错误。
- read返回值是实际读的字节数。对于读操作来说，返回的字节数小于*len*是正常的，因为在文件尾时，当前剩余可读的字节数可能会小于*len*，read的返回值，如果为0，说明读到了文件尾部。
- 在内核中，为每个打开的文件保留一个记录当前操作位置的指针，叫做文件的当前读写位置指针，读写共用一个指针，影响read和write操作的起始位置。

使用**read**或者**write**读写文件时，每次读写的字节数太少，会导致效率低。在注重效率的操作中，最好不要使用长度***len***过小的缓冲区***buf***。

缓冲区大小（B）	读出整个文件的时间（s）	缓冲区大小（B）	读出整个文件的时间（s）
1	130.45	512	5.71
4	33.56	1024	5.65
8	17.44	4096	5.64
16	9.64	8192	5.64
64	6.21	16384	5.64
256	5.83		

- 系统调用**lseek**提供了一种手段，用于强制设定文件的当前读写位置指针，下次的读写从这个位置开始。

#include <sys/types.h>

#include <unistd.h>

off_t lseek(int *fd*, off_t *offset*, int *whence*);

***fd*是文件描述符，*offset*是一个字节偏移量，函数返回值是下次读写位置，*whence*是指偏移量*offset*的参考基准。**

whence	文件读写位置指针的确定
SEEK_SET	读写指针移动到 offset 指定的位置（绝对值）
SEEK_CUR	读写指针移动到当前读写指针加 offset 后的位置
SEEK_END	读写指针移动到文件尾加 offset 后的位置，也就是文件中字节数加 offset 后的位置

3．关闭文件

关闭文件的调用很简单，提供一个“文件描述符”作为参数。

int close(int *fd*);

文件访问的另一种方式是使用C语言的标准函数**fopen，fprintf，fscanf，fgets，fread，fwrite，…，fclose。这一组函数，是在系统调用open，read，write，…，close基础上构建的库函数。**

4. 文件操作举例

【例4-24】 使用系统调用函数编程序以十六进制打印数据文件的内容。

这里的例子是一个使用open/read/close方式访问文件的例子。程序比UNIX的命令od要简单得多。允许提供多个文件名，将文件内容以十六进制打印出来。打开文件时，有简单的出错处理。

```
$ cat myod.c
#include <stdio.h>
#include <errno.h>
#include <fcntl.h>

int main(int argc, char **argv)
{
    int fd, k, i, len;
```

```
unsigned char buf[512];

for (k = 1; k < argc; k++) {
    fd = open(argv[k], O_RDONLY);
    if (fd == -1) {
        fprintf(stderr, "Open file \"%s\": %s (ERROR %d)\n",
            argv[k], strerror(errno), errno);
        continue;
    }
    while ( (len = read(fd, buf, sizeof(buf))) > 0) {
        for (i = 0; i < len; i++)
            printf("%02x ", buf[i]);
    }
    if (len < 0)
        perror("read data");
    close(fd);
}
printf("\n");
```

```
    return 0;
}
$ cc myod.c -o myod （编译链接生成可执行文件）
$ ./myod ../liang/.profile xyz
Open file "../liang/.profile": Permission denied (ERROR 13).
Open file "xyz": No such file or directory (ERROR 2).
$ ./myod *
90 36 59 ea 60 c4 90 a1 db 61 33 09 f6 b1 37 a9 d4
……
```

4.10.2 目录访问

- 在文件系统中，目录和普通文件一样，占用文件存储区中的空间。目录中的数据就是“文件名-i节点”对。能够吗？
- 在早期的UNIX中，每个目录项占16B，包括两字节的i节点号和最长14B的文件名，像普通文件那样用open打开目录文件读取其中的数据，程序自己来分析i节点号和文件名。
- 使用了长文件名之后，目录的存储结构发生了变化，应用程序须了解目录文件的存储结构。系统提供了几个库函数，用于操作目录表文件。

【例4-25】 与普通数据文件相同的方式访问目录文件。

在AIX系统中执行4.10.1小节中的应用程序myod，使用open和read调用读取目录文件的内容。

```
$ ls -ia /usr/adm
 2048 .          2072 dev_pkg.fail  2055 ras        2071 wtmp
    2 ..         4098 ffdc          4119 sa         2103 xferlog
 8193 SRC        2102 ftp.pids-all  4096 streams
 4118 acct       4106 invscout       2073 sulog
 2049 cron        4097 nim          4101 sw
$ ./myod /usr/adm
08 00 2e 00 00 00 00 00 00 00 00 00 00 00 00 00 02 2e 2e 00 00 00 00 00 00 00 00 00 00 00
00 08 01 63 72 6f 6e 00 00 00 00 00 00 00 00 00 00 08 07 72 61 73 00 00 00 00 00 00 00 00
00 00 10 00 73 74 72 65 61 6d 73 00 00 00 00 00 00 00 08 17 77 74 6d 70 00 00 00 00 00 00
00 00 00 00 10 01 6e 69 6d 00 00 00 00 00 00 00 00 00 00 00 00 10 02 66 66 64 63 00 00 00
00 00 00 00 00 00 00 10 05 73 77 00 00 00 00 00 00 00 00 00 00 00 00 08 18 64 65 76 5f 70
6b 67 2e 66 61 69 6c 00 00 10 0a 69 6e 76 73 63 6f 75 74 00 00 00 00 00 00 20 01 53 52 43
00 00 00 00 00 00 00 00 00 00 00 00 10 16 61 63 63 74 00 00 00 00 00 00 00 00 00 00 10 17
73 61 00 00 00 00 00 00 00 00 00 00 00 00 00 00 08 19 73 75 6c 6f 67 00 00 00 00 00 00 00
00 00 08 36 66 74 70 2e 70 69 64 73 2d 61 6c 6c 00 00 08 37 78 66 65 72 6c 6f 67 00 00 00
00 00 00 00
```

$ od -c /usr/adm

```
0000000  \b  \0   .  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000020  \0 002   .   .  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000040  \b 001   c   r   o   n  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000060  \b  \a   r   a   s  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000100 020  \0   s   t   r   e   a   m   s  \0  \0  \0  \0  \0  \0
0000120  \b 027   w   t   m   p  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000140 020 001   n   i   m  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000160 020 002   f   f   d   c  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000200 020 005   s   w  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000220  \b 030   d   e   v   _   p   k   g   .   f   a   i   l  \0  \0
0000240 020  \n   i   n   v   s   c   o   u   t  \0  \0  \0  \0  \0  \0
0000260     001   S   R   C  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000300 020 026   a   c   c   t  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000320 020 027   s   a  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000340  \b 031   s   u   l   o   g  \0  \0  \0  \0  \0  \0  \0  \0  \0
0000360  \b   6   f   t   p   .   p   i   d   s   -   a   l   l  \0  \0
0000400  \b   7   x   f   e   r   l   o   g  \0  \0  \0  \0  \0  \0  \0
```

UNIX目录操作的库函数：

#include <dirent.h>

DIR *opendir(char **dirname*);

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

int closedir(DIR **dir*);

- 提供给opendir一个目录名，得到一个句柄。以后的操作，就直接使用这个句柄代替目录名。
- readdir使用已经打开的目录句柄，每次调用获得一个目录项的数据。返回值是一个指针，指针指向的结构体记录了i节点号和文件名，分别放在dirent结构体的d_ino和d_name成员中。readdir返回NULL,标志目录表已经读到尾。
- 不再使用的目录句柄，用closedir关闭。

【例4-26】 目录表的访问举例。

下面的例子，使用目录操作的函数，给定一个目录名，列出该目录下的所有文件名和对应的i节点。类似ls -i的执行结果。

```
$ cat myls.c
#include <stdio.h>
#include <dirent.h>
#include <errno.h>

int main(int argc, char **argv)
{
   DIR *dir;
   struct dirent *entry;

   if (argc != 2) {
      fprintf(stderr, "Usage : %s <dirname>\n", argv[0]);
      exit(1);
   }
```

```
    dir = opendir(argv[1]);
    if (dir == NULL) {
        printf("Open directory \"%s\": %s (ERROR %d)\n",
            argv[1], strerror(errno), errno);
        return 1;
    }

    while ((entry = readdir(dir)) != NULL)
        printf("%d %s \n", entry->d_ino, entry->d_name);

    closedir(dir);
    return 0;
}
$ cc myls.c -o myls
$ ./myls /etc
4110 .
2 ..
4111 consdef
4112 csh.cshrc
4113 csh.login
```

```
4114 dlpi.conf
4115 dumpdates
4116 environment
4117 filesystems
4118 group
4250 inittab
4120 magic
4121 motd
8564 objrepos
4122 passwd
4123 profile
 ...
$ ./myls ../liaing
Open directory "../liaing ": No such file or directory (ERROR 2)
$ ./myls ../liang
Open directory "../liang": Permission denied (ERROR 13)
$
```

表4-6 与目录表相关的操作函数

功　　能	函 数 原 型
获取目录表的访问句柄	**DIR *opendir(char *dirname);**
读出目录中的一个目录项	**struct dirent *readdir(DIR *dir);**
关闭目录表的访问句柄	**int closedir(DIR *dir);**
删除文件	**int unlink(char *pathname);**
建立硬连接	**int link(char *oldpath, char *newpath);**
建立符号连接	**int symlink(char *symbol, char *path);**
文件改名	**int rename(char *oldpath, char *newpath);**
修改当前工作目录	**int chdir(char *path);**
当前工作目录名	**int getcwd(char *buf, int size);**
创建目录	**int mkdir(char *pathname, mode_t mode);**
删除目录	**int rmdir(char *pathname);**

4.11 获取文件的状态信息

UNIX系统提供的系统调用命令stat，可以获取文件或者目录的状态信息，这些信息大部分来自于i节点中的管理信息。

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

int stat(char *path, struct stat *statbuf);
    /* return 0 on success or -1 on error */

int fstat(int fd，struct stat *statbuf);
    /* returns 0 on success or -1 on error */
```

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

（1）st_dev: i节点所在设备的设备号。例如：SCO UNIX系统里，这是个16b整数，两个字节，高字节为主设备号,低字节为次设备号。例：根文件系统下文件状态中的st_dev为0x0128，是根文件系统设备/dev/root的主设备号（0x01）和次设备号（0x28）。有了st_dev就可以知道文件存储在哪个设备的文件系统中。

（2）st_ino: i节点的编号。

（3）st_mode: 文件方式。总共16b，含有文件的9b基本存取权限，SUID权限和SGID权限各占一个比特；以及文件的类型，占用若干比特。在<sys/stat.h>中定义了若干个常量,可判断文件的访问权限和文件类型。判断文件类型的方法，检查表达式 st_mode & S_IFMT，可能为下列几种值之一：

S_IFREG　普通磁盘文件（Regular file）

S_IFDIR　目录文件（Directory）

S_IFCHR　字符设备文件（Character device）

S_IFBLK　块设备文件（Block device）

S_IFIFO　管道文件（FIFO，先进先出）

S_IFLNK　符号连接文件（Symbolic link）

（4）st_nlink: i节点的link数。

（5）st_uid和st_gid: 文件主ID,用户组ID。在系统内部存储时不存储那些长度不定的字符串格式的用户名或者组名。这不利于节约存储空间和程序处理的时间。在创建用户和组的时候，每个组名字或者用户名字，都对应一个整数ID。系统提供一些名字和ID之间相互转换的函数。

（6）st_size：文件大小，文件中有效数据的字节计数。

（7）st_atime，st_mtime，st_ctime：与文件相关的时间。

文件的“访问（Access）”时间st_atime，指最后一次读取或者把文件当作一个程序文件加载执行的时间。这两个操作会影响文件的访问时间。

文件的“修改（Modify）”时间st_mtime，最后一次写文件的时间。

文件的“改变（Change）”时间st_ctime，最后一次写文件，会影响这个时间。另外，文件i节点记录的一些文件属性信息发生改变，这个时间也发生变化。

（8）st_rdev：当文件是字符设备文件或块设备文件时,该项中记录了设备号，包括主设备号和次设备号。例如：SCO UNIX中，st_dev占16b，共两个字节，高位字节是主设备号，低位字节是次设备号。Linux中是64b，也是含有主设备号和次设备号。对于其他类型文件，该项无意义。

使用目录操作的一组函数（opendir，readdir，closedir）以及stat调用可以编写出像ls这样的命令。

使用文件操作的系统调用open，read，……，或者在这组系统调用之上的缓冲I/O函数库fopen，fgets，……，可以构筑出诸如od，wc，grep，cp，awk等命令。

- 使用目录操作的一组函数（opendir，readdir，closedir）以及stat调用可以编写出像ls这样的命令。
- 使用文件操作的系统调用open，read，……，或者在这组系统调用之上的缓冲I/O函数库fopen，fgets，……，可以构筑出诸如od，wc，grep，cp，awk等命令。

4.12 设备文件

- UNIX的设备分为块设备和字符设备。
- 块设备面向磁盘等可以随机访问的存储设备，统一定义一种可以随机地根据块号进行操作的驱动界面，提供给文件系统使用，以便于使用UNIX的文件系统管理外存设备上的信息。文件系统中的命令mkfs，mount等，使用块设备文件名，映射块设备。
- 除了块设备之外的所有设备，包括终端、打印机等，都算作字符设备。

使用ls -l时，对于设备文件，列出主设备号和次设备号。例如：

$ ls -l /dev/hdc1 /dev/fd[01] /dev/tty1

brw-rw---- 1 root floppy 2, 0 Jan 30 2003 /dev/fd0

brw-rw---- 1 root floppy 2, 1 Jan 30 2003 /dev/fd1

brw-rw---- 1 root disk 22,1 Jan 30 2003 /dev/hdc1

crw------- 1 root root 4, 1 Jun 2 20:32 /dev/tty1

- 主设备号对应物理设备的一类。主设备号被用来定位所使用的设备驱动程序，次设备号的作用很简单，用来区分这一类物理设备的哪一个。
- 所谓的“设备驱动程序（Device Driver）”是按照操作系统内核规定的接口标准事先编制好的一组子程序库，这些子程序库中含有多个事先约定好的函数入口，通过静态或者动态链接的手段，加载到操作系统内核中，等待操作系统内核在合适的时机调用相关的函数。

- 设备驱动程序中有多个入口函数，假定是**xxopen()，xxwrite()，xxclose()，xxintr()**。设备驱动程序安装时注册这些函数，安装完成之后，会得到一个主设备号，主设备号与驱动程序相关联，根据主设备号，可以检索到已经登记了的这个设备的设备驱动程序的入口函数。
- 设备驱动程序入口表：主设备号常常是内核中的一个数组下标，数组的每个元素是个结构体，每个结构体对应一个设备驱动程序，结构体的多个成员都是C语言的函数指针，指向加载到内核的驱动程序的函数**xxopen()，xxwrite()，xxclose()，xxintr()**，等等，这个数组被称作“设备驱动程序入口表”。

在文件系统中创建8个设备文件/dev/ttyE0 ~ ttyE7。那么，执行命令 who > /dev/ttyE0。

- 系统会像普通文件操作一样通过open系统调用打开文件/dev/ttyE0，分析了文件的i节点，它不是普通磁盘文件，而是一个字符设备文件
- 以主设备号为数组下标，检索设备驱动程序入口表，找到open操作时应调用的函数的指针，函数指针指向设备驱动程序的函数xxopen()。调用这个设备驱动程序的xxopen()函数，设备驱动程序根据需要，对硬件作某些初始化等，然后返回，open结束。

- who命令的输出，导致write系统调用，系统分析出这是对字符设备的操作，以主设备号为数组下标，检索设备驱动程序入口表，找到write操作时应调用的函数指针，函数指针指向设备驱动程序的函数xxwrite()，调用设备驱动程序的函数xxwrite()。
- 设备驱动程序通过函数被操作系统调用而得到数据，如何处理这些数据，就是设备驱动程序自己的任务。它可以根据这一特定硬件的功能，控制某些设备寄存器，将数据通过串行通信口一个个发送出去。设备产生中断时，操作系统调用设备驱动程序的xxintr()函数。

本例中有八个串行通信口，内核调用驱动程序入口函数时，提供了次设备号，驱动程序才知道该向哪个串行通信口输送数据。创建设备文件的命令是mknod,例如：

mknod /dev/ttyE0 c 16 0

其中，/dev/ttyE0是要创建的设备文件名，c指的是字符设备，16是主设备号，0是次设备号。

- 虚拟设备：如果一个设备驱动程序不操作任何物理硬件，也是完全可以的。它像普通设备驱动程序一样提供设备驱动程序入口，通过这些入口，可以和内核交互作用，得到某些数据。这些设备叫做“虚拟设备”。
- 虚拟设备文件/dev/random是一个随机数生成器，根据系统键盘和鼠标等外设的活动生成二进制的随机数数列。
  - 命令od -x /dev/random可以观察这些随机数序列。在系统外设都不活动时，这个设备暂停供应随机数序列。
- 另一个虚拟设备文件/dev/urandom，无论外设是否活动，只要你的程序读取该设备，都会源源不断地供应随机数序列。

- 虚拟设备文件/dev/null
  - 所有写入该设备的数据统统被丢弃；
  - 任何时候都可以打开该设备读取，每次都读不到任何数据，就遇到文件结束，同读取一个含有0字节的磁盘文件效果一样。
- 虚拟设备文件/dev/zero
  - 向这个文件中写入数据，如同写入/dev/null，所有数据被丢弃。
  - 从文件/dev/zero中读取时，程序的任何一次read系统调用，都会返回一连串的字节，每个字节都是二进制0B。

【例4-27】 设备文件/dev/null的使用举例。

（1）自编的程序文件myap有许多输出，将输出显示在终端屏幕上会导致显示太多而且程序执行很慢，通过./myap > myap.log的方法，因为程序过多的输出挤占磁盘空间。可以使用命令./myap > /dev/null丢弃所有输出。需要丢弃程序输出时经常使用虚拟设备/dev/null。

（2）检索当前目录以及当前目录的所有下属子目录中的C语言头文件，查找所有含有inode的行。可以使用下面的命令：

```
find . -name '*.h' -exec grep -n inode {} \;
find . -name '*.h' -exec grep -n inode /dev/null {} \;
```

4.13 文件和目录的权限

4.13.1 权限控制的方法

- UNIX的每个文件和目录，都有一组权限与这个文件或者目录相对应，存放在它的i节点中，用于控制用户对它的访问。
- 每个文件都有惟一的属主和组号，都记录在i节点中。这些信息在进行权限判断时使用。
- 在i节点中记录的权限有9b，描述3个级别，分别是文件主、同组用户、其他用户。每个级别的权限包括3部分：读权限、写权限、执行权限。

1. 普通文件的权限

- 如果用户对文件具有读权限，那么用户可以读取文件。
- 如果用户对文件具有写权限，那么它可以修改文件的内容。
- 如果用户对文件具有可执行权限，那么，就可以执行这个文件。

2. 目录的权限

从UNIX的文件系统的组织结构上看，所谓“目录”，也组织成像普通磁盘文件一样的文件，其中存储了多个由“文件名-i节点号”组成的目录项。目录的读写权限，就是对这个目录表文件的读写权限。

- 如果对目录没有读权限，那么，目录表文件不许读，ls列出目录的操作会失败。
- 如果对目录没有写权限，那么，所有会导致目录文件修改的操作都被禁止。例:创建文件，删除文件，文件改名，复制文件到该目录，创建子目录，删除子目录。
- 对目录有执行权限，意味着在分析路径名的过程中可以检索该目录。

- 目录不可写，就可以保护目录下的所有的文件不可写？

  其实不然。在一个只允许读的目录下修改一个已经存在的文件，不需要修改目录表中的任何数据。但是，i节点中的数据和文件内容要发生变化，只要文件自身有可写属性，这些操作就能正常完成。

- 一个文件具备只读属性，文件不能被删除？

  文件所处的目录有写权限，那么，删除这个文件就完全可能。也就是说文件的只读权限可以保护文件不被误写，但不能保护文件被误删。因为删除文件，不需要打开文件写，只需要修改文件所处的目录，只要目录可写就可以了。

3．访问合法性验证的顺序

用户在操作一个文件的时候，系统根据登录用户的用户名、组名以及文件i节点中存储的文件主和组，判断该使用3级权限的哪一级。判断的方法是：

（1）若文件主与登录用户相同，则只使用文件主权限，不再查组和其他用户的权限。

（2）若文件主与登录用户不同，但文件i节点中记录的组号与登录用户的组号相同，则只使用组权限，不使用关于其他用户的权限。

（3）若文件主与登录用户不同，文件i节点中记录的组号与登录用户的组号也不同，则使用文件关于其他用户的权限。

4.13.2 查看文件和目录的权限

使用ls命令，有关选项-l和-d。 例如:

ls -l 可以查当前目录下所有文件和子目录的权限

ls -1d . 列出当前目录自身的权限

4.13.3 chmod:修改权限

只有超级用户和文件主，才允许修改文件的存取权限。修改文件存取权限的命令是**chmod**。该命令有两种方式修改文件的权限。

1. 字母形式

命令格式为：

chmod [ugoa][+-=][rwx] *file-list*

描述用户级用一个或者多个下列的字母表示。

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

下面的符号定义要执行的操作：

+ 给指定用户增加存取权限

- 取消指定用户的存取权限

= 给指定用户设置存取权限，其他权限都取消

下面的字母定义存储权限：

r 读权限

w 写限

x 执行权限

例如：

chmod u+rw *

chmod go-rwx *.[ch]

chmod a+x batch

chmod u=rx try2

2．数字形式

数字形式的命令格式为：

chmod *mode file-list*

权限的描述，使用3个八进制数字，分别描述文件主、同组用户、其他用户的权限。这种方法将文件的权限强迫设置为某一个值。

例如：权限640，3个八进制数字分别为6、4、0。文件主可读写，同组用户可读，其他用户不可读写。

八进制：　6　4　0

二进制：　110　100　000

权限：　rw-　r--　---

例：chmod 640 *.[ch] makefile *.log

【例4-28】 文件和目录的权限对文件和目录操作的限制。

下面是上机操作的例子，可以看出文件和目录权限的作用。

（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 user文件主取消写权限

$ 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） 目录文件有写权限

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

（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.

（目录读权限）

$ pwd

/usr/jiang

$ chmod u-r .

$ ls （不可读的目录无法列出其中文件）

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，但失败，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的权限，可恢复当前目录权限）

（子目录ttt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.
```

4.13.4 umask:改变文件创建状态掩码

1. umask命令

- umask是一个shell内部命令，用于决定文件和目录的初始权限。
- 在UNIX中，umask是进程属性的一部分，每个进程都对应一个umask值，umask命令用于修改shell进程自身的umask属性。
- 一般将umask命令放到自动执行批处理文件中，如：用csh作登录shell时，可以将umask命令加入到.login文件或.cshrc文件中；用sh作登录shell时，可以将umask命令加入到.profile文件

例：**umask** 打印当前的**umask**值

umask 022 强制将**umask**值设置为八进制的**022**

掩码值的含义是在新创建的文件和目录中，不含有掩码值中列出的权限。

掩码值：　**022**

二进制：　**000010010**

掩掉的权限：**----w--w-**

2. 系统调用umask

掩码值是进程属性的一部分，系统调用函数**umask**用于修改当前进程的掩码值。函数原型为：

int umask(int *new_mask*);

new_mask为指定的新掩码值，函数返回值为原先的掩码值。

- **umask和open中的权限参数共同确定了新文件的初始权限。**

  设umask为077时，用C程序：

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

  open创建的文件的权限为0666，屏蔽掉077后为0600，即rw-------。

- **系统调用chmod可以修改文件的权限,它不受umask的影响。函数原型为：**

  int chmod(char **path*, mode_t *mode*);

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

4.13.5 SUID权限和SGID权限

设用户liu记录了在本UNIX系统中的某些用户的月工资清单，记录在文件list.txt中，假定文件list.txt的内容如下：

$ cat list.txt

```
#=============================================================
# 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        1782   1500        200        175        3307
  liang   2074    梁振宇      03        1560   1400        180        90         3050
  sun     3087    孙东旭      03        1804   1218        106        213        2915
  tian    2076    田晓星      04        1832   1450        230        245        3267
  liang   2074    梁振宇      04        1660   1450        230        70         3270
  sun     3087    孙东旭      04        1700   1310        283        270        3023
#=============================================================
#       注：煤气费不再从工资中扣除，由煤气公司自行收缴。
```

先看下面由用户liu编写的C源程序query.c。

```
$ awk '{printf("%2d %s\n",NR,$0)}' query.c
 1 #include <stdio.h>
 2 #include <string.h>
 3 void main(void)
 4 {
 5    FILE *f;
 6    char line[512], login_name[64];
 7    f = fopen("list.txt", "r");
 8    if (f == NULL) {
 9        perror("*** ERROR: Open file \"list.txt\" ");
10        exit(1);
11    }
12    while (fgets(line, sizeof(line), f)) {
13        if (line[0] == '#') {
14            printf("%s", line);
15        } else {
16            if (sscanf(line, "%s", login_name) > 0) {
17                if (strcmp(login_name, getlogin()) == 0)
18                    printf("%s", line);
```

```
19          }
20       }
21    }
22    exit(0);
23 }
$ cc query.c -o query
$ 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
$ query
*** ERROR: Open file "list.txt" : Permission denied
$ cat list.txt
cannot open list.txt: Permission denied
$
```

- 在UNIX中文件query的文件主用户liu可以给文件query增加SUID（设置用户标识）权限。用户liu的操作过程如下：

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

- 增加了SUID权限以后，ls -l命令显示的文件主对该文件的操作权限为rws，在执行权限处显示字母s而不是字母x。

这样，用户liang就可以通过query命令查询只许liu可读的文件list.txt，尽管用户liang没有对文件list.txt的读权限。用户liang的操作过程如下：

```
$ 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    1560   1400   180    90    3050
  liang   2074   梁振宇   04    1660   1450   230    70    3270
#=================================================
#       注：煤气费不再从工资中扣除，由煤气公司自行收缴
$ cat list.txt
cannot open list.txt: Permission denied
$
```

- 文件主**liu**对可执行文件**query**授予**SUID**权限意味着，无论哪个用户，只要拥有对文件**query**的可执行权，**query**程序执行时，使用文件主**liu**的权限来访问文件。
- 其看到的内容受限于可执行文件**query**内部的程序设计。使用这种方法，限制其他用户只能以文件主提供的程序来访问文件。

- UNIX每启动一个程序，系统就创建一个进程，每个进程都有两个UID(用户标识号)：实际的UID和有效的UID，记录在内核进程proc结构中的p_uid域和p_suid域。一般情况下，用户启动一个可执行的程序时，进程的实际UID和有效UID相等，就是用户自己。

- 进程在打开一个文件时，文件系统将根据进程的有效UID而不是实际UID，与文件所有者UID之间的关系和文件自身的权限进行访问合法性验证。

- 当一个可执行程序有SUID权限后，启动这一程序创建新进程，新进程的实际UID和有效UID就不再相等，实际UID是启动这一程序的用户的UID，而有效UID为可执行文件的文件主的UID，而UNIX对文件访问权限的检查使用进程的有效UID。

- 删除一个文件的SUID权限使用chmod u-s命令。例：

- **SGID（设置组标志）权限与SUID权限类似。一个拥有SGID权限的可执行文件在执行过程中，当需要进行文件访问合法性验证时，如果需要判断进程是否与被访问文件的文件主同组时，进程的组标识符GID将使用可执行文件的文件主的组GID。**

增加SGID权限时，文件必须事先有组执行权限，命令格式为：

chmod g+s *file-list*

删除SGID权限的命令：

chmod g-s *file-list*

有SGID权限的文件在ls -l列表时，组执行权限处显示小写字母s，而不是x。