第四部分　文件和目录、文件系统、管道

第四部分讨论文件和文件系统。

文件系统由一组文件、控制表和目录组成，它们都存放在同一存储设备，例如磁盘包上。

本部分包含与文件系统相关的下列内容：

- 创建的存取文件。
- 径由目录定位文件。
- 组织和维护文件系统。

本部分也包括与“管道”(“pipe”)特殊性有关的代码。

第18章　文件存取和控制

每一种操作系统的大部分代码似乎都与数据管理和文件管理有关， UNIX也不例外。

18.1　源代码第四部分

源代码第四部分包含13个文件。其中前4个包含其他各例程需要的公共说明：

- “file.h”说明“file”结构及数组。
- “filsys.h”说明“安装”(“mounted”)文件系统的“超级块”(“super block”)结构。
- “ino.h”说明记录在“安装”设备上的“inodes”结构。
- “inode.h”说明“inode”结构及数组。

下面2个文件，“sys2.c”和“sys3.c”包含与文件存取及控制有关的系统调用的代码(“sys1.c”和“sys4.c”包含在源代码的第二部分)。

下面5个文件，“rdwri.c”、“subr.c”、“fio.c”、“alloc.c”和“iget.c”提供了与文件管理有关的各主要例程，它们也提供面向i/o的系统调用和各基本i/o例程之间的一条纽带。

文件“nami.c”中代码的主要功能是：按给出的文件路径名在目录结构中搜索，找到相应的“inode”。

最后，“pipe.c”是管道文件的“设备驱动程序”。

18.2　文件特征

在概念方面而言，一个UNIX文件是一个具有名字的字符串，它存放在一个外设上 (或内存中)，并能通过适合于常规外部设备的机构对其进行存取。

我们将会提及， UNIX文件并不包含记录结构。但是，在UNIX文件中可以插入“新行”符，这样就可将整个文件分成多个子字符串，这与记录有些类似。

设计UNIX文件系统的一个基本思想是使文件独立于设备，使文件名惟一地决定该文件的所有相关属性。

18.3 系统调用

UNIX提供下列对文件进行处理的系统调用：

#	名字	行	#	名字	行
3	read	5711	14	mknod	5952
4	write	5720	15	chmod	3560
5	open	5765	16	chown	3575
6	close	5846	19	seek	5861
8	creat	5781	21	mount	6086
9	link	5909	22	umount	6144
10	unlink	3510	41	dup	6069
12	chdir	3538	42	pipe	7723

18.4 控制表

数组“ file ”和“ inode ”是文件存取机构中的关键组成部分。

18.4.1 file(5507)

数组“ file ”定义为也被命名为“ file ”的结构类型。

若“ file ”数组中某个元素的“ f_count ”项值为0，则可认为该元素是未分配使用的。

每一条“ open ”或“ creat ”系统调用都会导致在“ file ”数组中分配一元素。该元素的地

的一项中。而“ u.u_ofile ”数组中该项的下标号则传送给

roc ”创建后代进程，这些后代进程继承它们父进程

数器“ f_count ”，它表示引用此“ file ”元素的当前进程数。

78)、“ dup ”(6079)和“ falloc ”(6857)增1；由“ closef ”

生不能被打开)减1。

该文件为读打开或为写打开，以及它是否是一“ 管道 ”文

”文件。)

ıode ”的指针(5511)，它指向“ inode ”表中的一项，以及

它是一个逻辑指针，指定文件中一个字符的位置。

ode ”(index node,索引节点)的结构数组。

“ i_count ”项值为0，则可认为该元素是未分配使用的。

正常 i/o操作引用的文件、每个正被执行或已被执行并且

个进程的工作目录，“ inode ”数组中都分别包含1项。

一“ inode ”项。inode项说明相应文件的一般特性。

18.5 要求专用的资源

每个文件都要专用某些系统资源。当一个文件存在但并不以任何方式正被引用时，它要求使用下列资源：

1) 一个目录项(在一个目录文件中的16个字符)。

2) 一个磁盘“ inode ”项(存放在磁盘上一个表中的32个字符)。

3) 磁盘上的1块或多块，也可以是0块(每块512个字符)。

另外，若文件正因某种目的而被引用，则它还要求：

4) 一个内存“ inode ”项(inode数组中的32个字符)。

最后，若用户程序已为读或写打开了该文件，则还要使用另一些资源：

5) 一个“ file ”数组项(8个字符)。

6) 在用户程序“ u.u_ofile ”数组中的1项(每个文件一个字，它指向file数组项)。

为了以正常方式分配和释放这些资源，建立了一套机构。在下面的表中列出了与此有关的各主要过程的名字：

资 源	获 得	释 放
目录项	namei	namei
磁盘“ inode ”项	ialloc	ifree
磁盘存储块	alloc	free
内存“ inode ”项	iget	iput
“ file ”表项	falloc	closef
“ u_ofile ”项	ufalloc	close

18.6 打开一个文件

当一道程序希望引用一已存在的文件时，它必须“ 打开 ”该文件以构造通向该文件的一座桥梁(注意，在UNIX中，进程通常继承其父进程或祖先进程的打开文件，所以一个进程所需的所有文件常常已经是隐式打开了的)。若所需的文件并不存在，那么应当先创建该文件。我们先分析这第二种情况：

18.7 creat(5781)

5786：“ namei ”(7518)将一路径名变换成一个“ inode ”指针。“ uchar ”是一个过程的名字，它从用户程序数据区一个字符一个字符地取得文件路径名。

5787：一个空“ inode ”指针表示出了一个错，或者并没有具有给定路径名的文件存在。

5788：对于出错的各种条件，请见UPM的CREAT(II)。

5790：“ maknode ”(7455)调用“ ialloc ”创建一内存“ inode ”，然后对其赋初值，并使其进入适当的目录。注意，显式地清除了“ 粘住 ”位 (ISVTX)。

18.8 open1(5804)

此过程由“ open ”(5774)和“ creat ”(5793、5795)调用，它们传送的第3个参数，“ trf ”的

值分别是0、2和1。值2表示的情况是：不存在所希望名字的文件。

5812：当“ trf ”值为0时，第2个参数“ mode ”可取值01(FREAD)、02(FWRITE)或03(FREAD｜FWRITE)，否则只能取值02。

5813：在所希望名字的文件已存在情况下，调用“ access ”(6746)对所希望的活动模式进行存取权检查，“ access ”作为副作用可能设置“ u.u_error ”。

5824：若此文件正被“ 创建 ”，则调用“ itrunc ”(7414)删除其原先的内容。将测试条件更改为“（trf==1) ”可以改善此处的代码。请证明这确实如此。

5826：调用“ prele ”(7882)以“ unlock ”(解锁)索引节点。你可能会问：在何处对索引节点上锁?为什么?

5827：注意，“ falloc ”(6847)首先调用“ ufalloc ”(6824)。

5831：“ ufalloc ”将用户文件标识数留在“ u.u_ar0〔R0〕”中。为什么将此条语句安排在此处而不是在5834行之后呢?

5832：调用“ openi ”(6702)，以便在要求任一设备的特有动作时调用特殊文件的处理程序(对于磁盘文件，没有动作)。

5839；在构造“ file ”数组项时若出错，调用“ iput ”释放“ inode ”项。

我们将会观察到，多方面分工合作才实现打开一个文件的功能。“ falloc ”和“ open1 ”对“ file ”表项赋初值；“ iget ”、“ ialloc ”和“ maknode ”则对“ inode ”表项赋初值。

注意，“ ialloc ”清除新分配“ inode ”中的“ i_addr ”数组，而“ itrunc ”则清除以前存在的“ inode ”中的“ i_addr ”数组，所以，在“ creat ”系统调用之后，没有磁盘块与该文件相关连，该文件的长度类型现在是“ 小文件 ”。

记：一道程序希望引用一已存在的文件。

的第2个参数值为0，表示要找到命名的文件。(u.u_arg〔0〕

字符串规定了一个文件路径名。)

这是由在用户程序设计约定和内部数据表示之间失配而造

取许可权(5813)，但不释放现已存在的文件(5824)。

用“ falloc ”(5827)，并未直接调用其他资源分配例程。自需要分配目录项、磁盘“ inode ”项和磁盘块。如果需要，用的副作用而分配的，但是……在何处对它赋初值呢？

地去除一个用户程序和一个文件之间的连接，所以可将调用。

用户程序的文件标识通过 r0传送给“ close ”，该值由“ getf ”(6619)查验。然后释放“ u.u_ofile ”项。最后，调用“ closef ”。

18.12 closef(6643)

“ closef ”由“ close ”(5854)和“ exit ”(3230)调用。(因为大多数文件并不被显式关闭，而是在用户程序终止执行时隐式关闭，所以后者更普遍一些。)

6649：若此文件为管道，则清该管道的“ i_mode ”，然后唤醒正等待此管道的进程，或正等待信息或空间的进程。

6655：若这是引用该文件的最后1个进程，则调用“ closei ”(6672)以对特殊文件进行结束处理，然后调用“ iput ”。

6657：将“ file ”项的引用计数减1。若该值现在为0，则该项是可用的。

18.13 iput(7344)

“ closei ”的最后一个动作是调用“ iput ”。在需要删除对一个内存“ inode ”的连接并对其引用计数减1时，都需调用“ iput ”。

7350：若在此点上引用计数值是 1，则将释放“ inode ”。在进行这种处理时，应对该“ inode ”上锁。

7352：若对该文件的连接数是0(或更少)，则将释放该文件(见下面)。

7357：“ iupdat ”(7374)更新记录在磁盘“ inode ”中的存取和更新时间项。

7358：“ prele ”解锁“ inode ”。为什么在此处和7363行都要调用“ prele ”呢?

18.14 删除文件

新文件作为永久文件自动进入文件目录。关闭文件不
们在7352行所见到的，当内存“ inode ”项中的“ i_nlink
时，将删除该文件。在创建文件时，该字段由“ makno
(5941)可将其值加1，系统调用“ unlink ”(3529)则可将其值

创建临时“ 工作文件 ”的程序应当在其终止前执行“
注意，“ unlink ”系统调用本身并没有删除文件。当引用计数
才删除该文件。

为了减少在程序或系统崩溃时遗留下来的临时文件所带

1) 在打开临时文件后立即对其执行“ unlink ”操作。

2) 应在“ tmp ”目录下创建临时文件。在文件名中包括
(见getpid(3480))。

18.15 读和写文件

在详细查看执行“ read ”系统调用的代码之前，先让我
作过程。

```
……read(f,b,n);／*用户程序*／
        {发生陷入}
2693    trap
        {#3系统调用}
5711    read();
5713    rdwr(FREAD);
```

用户进程执行系统调用激活运行在核心态的“ trap ”。“ trap ”识别＃３系统调用，然后通过“ trapl ”调用例程“ read ”，它又调用“ rdwr ”。

```
5731 rdwr

5736  fp = getf (u.u_ar0[R0]);
5743  u.u_base = u.u_arg[0];
5744  u.u_count = u.u_arg[1];
5745  u.u_segflg = 0;
5751  u.u_offset[1] = fp->f_offset[1]
5752  u.u_offset[0] = fp->f_offset[0]
5754  readi(fp->f_inode);
5756  dpadd(fp->f_offset,
            u.u_arg[1]-u.u_count);
```

“ rdwr ”包含了很多“ read ”和“ write ”操作共用的代码。它调用“ getf ”(6619)将用户进程提供的文件标识变换成“ file ”数组中一项的地址。

注意，该系统调用的第1个参数是以不同于另外2个参数的方式传送的。

将“ u.u_segflg ”设置为0，这表示此操作的目的地址在用户地址空间中。在以一个 inode 指针参数调用“ readi ”后，将要求传送的字符数减去剩余未传输字符数 (在u.u_count中)，加至文件位移量中。

```
6221 readi

6239  lbn = lshift (u.u_offset, -9);
6240  on = u.u_offset[1] & 0777;
6241  n = min (512 - on, u.u_count);
6248  bn = bmap(ip, lbn);
6250  dn = ip->i_dev;
6258  bp = bread (dn, bn);
6260  iomove (bp, on, n, B_READ);
6261  brelse (bp);
```

i部分：一个逻辑块号“ lbn ”，以及一个块内索引“ on ”。
的较小者：“ u.u_count ”和块内尚余字符数 (在这种情况下
·步对此说明)，还应考虑尚余留在文件中的字符数 (对这种

设备编号，“ bn ”是在该设备 (磁盘)上的实际块号，这是由
]。
求的磁盘块，若需要，则将其从磁盘复制到内存中。
专送至目的区，然后执行计数操作。

有很多相似之处，两者共享很多代码。系统调用“ read ”

(5711)和“ write ”(5720)，然后立即调用“ rdwr ”，它执行下列操作：

5736：将用户程序文件标识变换成指向相应文件表项的指针。

5739：检查所要求的操作(读或写)是否与文件打开时的读/写方式符合。

5743：用各参数在“ u ”中设置几个标准单元。

5746：从此开始对“ 管道 ”文件进行特殊处理。

5755：按读、写要求分别调用“ readi ”或“ writei ”。

5756：更新文件位移量，使其增加实际传送的字符数，同时也将实际传送的字符数返回。

18.17 readi(6221)

6230：如果不需要传送字符，则立即返回。

6232：设置“ inode ”标志以表示该“ inode ”已被存取。

6233：如果该文件是一字符特殊文件，则调用相应设备的“ read ”（读）过程，并以该设备标识作为参数传送给该过程。

6238：开始传输数据的循环，每次传送字符数最多为 512，该循环在满足下列两个条件之一时结束(6262)：

• 发生一不可克服的错误。

• 已传送了要求数量的字符数。

6239：“ lshift ”(1410)将“ u.u_offset ”数组中的两个字并接，然后右移 9位，并切除为16位。这规定了要引用的文件逻辑块号；

6240：“ on ”是块内字符位移量。

6241：在本块内剩余字符数(亦即on之后的字符数)和要

者，并将其赋予“ n ”。注意，“ min ”(6339)将其参数处理为

6242：如果此文件不是特殊块设备文件，那么……

6243：将文件位移量与当前文件长度进行比较。

6246：将“ n ”设置为所要求字符和文件中余留字符两

6248：调用“ bmap ”，将文件逻辑块号变换成其宿主

要对“ bmap ”作更多说明，现在我们应注意到“ bmap ”作

6250:从“ inode ”取设备标识，并将其赋予“ dn ”。

6251：如果此文件是特殊块设备文件，那么……

6252：将“ inode ”项中的“ i_addr ”字段值赋予“ dn

字段值相同)。

6253：将预读块变量“ rablock ”设置为下一个物理块

6255：如果该文件中各块正被顺序读，于是……

6256：调用“ breada ”，它读所希望的块，然后启动预

6258：只是读所希望的块。

6260：调用“ iomove ”，将信息从缓存传送到用户区。

6261：将该缓存释放回av列表。

18.18 writei(6276)

6303：如果要写的字符数小于块长，则将该块中的先前内容先读至一缓存中，这样就可保留相应部分，否则只可分配可用缓存。

6311：没有“ 预写 ”（write ahead）设施，但是对后面部分的字符尚未更改的缓存则有“ 延迟写 ”（delayed write)。

6312：如果文件位移量现在超过了已记录的文件字符尾端，那么很显然此文件变大了！

6318；为什么再次设置“ IUPD ”标志是所希望的?(参见6285行。)

18.19 iomove(6364)

此过程开始处的注释对其功能作了很详细的说明。在 1244行、1252行、6542行和6517行可分别找到“ copyin ”、“ copyout ”、“ cpass ”和“ passc ”。

18.20 bmap(6415)

对“ bmap ”功能的一般说明可参见UPM的FILE SYSTEM(V)。

6423：不支持长度大于2^{15}块(2的15次方块，即2^{24}字符)的文件。

6427：从“ 小 ”文件算法开始 (“ 小 ”文件指的是其长度小于等于 8块，亦即4096个字符的文件)。

6431：若块号是8或更大，则应将“ 小 ”文件扩展为“ 大 ”文件。注意这是“ bmap ”的副作用，仅当“ bmap ”由“ writei ”调用时，才会发生这种文件扩展 (readi调用bmap时决不会

文件都是从“ 小 ”文件开始的，并且决不会显式地改变成

是不可逆转的！

备的空间块列表中分配一块。然后为此块分配一缓存，最

i_addr ”数组的8个缓存地址复制至该缓存区中，然后擦除

该缓存，并使该缓存为“ 延迟写 ”缓存。

需要，得到下一块。

：

statl	(6045)
dup	(6069)
owner	(6791)
suser	(6811)