浙江大学出版社

胡希明　张建平　等编著

UNIX

结构分析

（核心代码的结构和算法）

UNIX 结构分析

（核心代码的结构和算法）

胡希明　张建平等 编著

浙江大学出版社
1990 年 · 杭州

(浙)新登字 10 号

UNIX 结构分析

（核心代码的结构和算法）

胡希明　张建平等编著

责任编辑　尤建忠

*　　*　　*

浙江大学出版社出版

杭州华光电子技术服务部排版

杭州富阳何云印刷厂印刷

浙江省新华书店经销

*　　*　　*

开本 787×1092　1/16　印张:44.5　字数:1539 千

1990 年 12 月第 1 版　1992 年 9 月第 3 次印刷

印数：7 001—12 000

ISBN 7-308-00595-X

TP・040　定价：17.80 元

前　言

本书纯粹是为了教学的目的而编写的。

十年前，开始接触 UNIX，在学习国内外专家关于 UNIX 的论著的同时，还选读了几个版本的全部或部分 UNIX 代码。先是第六版的 C 代码，尔后是第七版反汇编代码，再是系统 V 和 BSD4.2 的 C 代码。十年间，每年给大学生和研究生讲授操作系统、UNIX 环境等课程，同时也搞一点基于 UNIX 的 CAD、信息系统和软件工程等方面的开发研究项目。十年过去了，根据一些朋友的建议和一部分愿意跟我一起学习和工作并且对 UNIX 有兴趣的学生的要求，利用多年积累的备课笔记、读书笔记以及国内外同行的著作，编写了本书。

编写过程中得到了张建平博士、费海林、肖榕等同学的大力支持，特别是文稿的整理主要由他们承担。另外，研究生周治平、徐军、吴芳辉和黄建舟也做了不少工作。应该说，本书是由我和他们几位共同编写的。

全书拟分三卷。

第一卷　核心代码的结构和算法分析，包括系统 V 核心最基本部分的源代码。

第二卷　系统的生成和初始化，包括 init、getty、login、sh 和 csh 的结构分析和源代码。

第三卷　UNIX 工具，包括 ed、vi、make、yacc、sccs、uucp 等软件开发工具的剖析和源代码。

第一卷分六个部分：

第一部分　文件系统

分四章，介绍文件 UNIX 文件、文件系统以及系统缓冲区的内部结构(包括抽象数据结构和对抽象数据结构的操作)。同时给出有关的源程序。为阅读方便，源程序加了少量提示，此类提示一律采用/*【.....】*/的形式。另外，为保持原有程序的风格，除少数行分成二行之外，其余与源代码完全一致。

第二部分 存储管理

分四章，包括存储管理的硬件基础、系统空间管理、进程空间管理和物理内存的管理。每章均以 VAX-11 为机器模型。

第三部分　进程管理

从描述进程静态特征的抽象数据结构讲起，分五章，包括：进程的结构与状态、进程的创建、执行与终止、进程调度、中断与异常处理、进程通讯.

第四部分　输入/输出子系统

分三章，先考察进程与 I/O 子系统之间的接口，以及机器和设备驱动程序之间的接口，研究设备驱动程序的一般结构和功能，然后详细地讨论块设备和字符设备的管理。

第五部分　其　　他

共四章，包括：系统初始化过程概述，与硬件有关的汇编程序，系统生成等内容，还包括了前面几部分未曾介绍而又少不了的几个源程序，属补遗一类。

第六部分 附 录

附录包括了二个内容：

附录A： .h文件，整个系统核心除去C程序和汇编程序之外，还有就是.h文件，这是阅读理解系统C代码所必不可少的，正文中已引用了一部分，附录A给出未曾引用过的.h文件清单，引用过的则只给出文件名并注明书中引用章节。

附录B： 函数索引，为阅读代码方便，编了一份索引表，每个C函数、每个宏定义函数都给出名字和所在文件名（包括文件所在章节），并以字母顺序排列。

十年前，何志钧教授、冯树椿副教授、陈增武副教授最初安排和支持我学习UNIX，开设操作系统和UNIX课程，借此机会向他们表示感谢。在学习UNIX的过程中，曾得到过国内外许多专家和朋友的帮助和支持，我一直怀着感激的心情记着他们。前后有数十位学生除了听我的课以外，还曾经与我一起阅读并研究UNIX文献和代码，他们几乎全部都已或正在读取硕士、博士学位，有相当一部分目前仍在国外，一起学习的时候他们都曾给了我许多的支持和安慰，我想念他们。可惜的是，毛德操先生曾与我一起工作多年，经常共同研讨UNIX的优缺点，还曾试图对UNIX加以较大的改进并设计了具体方案，如今他已去了美国，不知何日能再相会。

特别要感谢系主任俞瑞钊教授，感谢他的支持和鼓励。

UNIX的出现，是几十年来计算机科学技术发展史上的一个里程碑，它造就了计算机科学技术多个领域里的一代新人。

我自己至今仍然还是一个UNIX的初学者，常常会因为一段代码或一个算法难以理解而苦恼，而一旦看到或听到国内外同行的深刻剖析，由衷感到佩服。谢谢他们，愿意继续向他们学习。

书中定有不少遗漏、欠缺或错误的地方，希望有改正与完善的机会。

胡希明
1990年8月于浙大计算机系

目　录

第一部分

文件系统

文件与进程，一静一动，是理解 UNIX 内部结构的二个主要概念，前者易于接受但往往留于表面，后者难以把握常常使人感到“只可意会，不可言传”。其实二者都是应该并且完全可以讲清楚的，问题在于要有文字性的描述、分析，还要化点时间读一部分 UNIX 源程序。许多与 UNIX 以至一般操作系统有关的概念，读了部分源程序后就可以较为确切的理解其内涵了。

文件的概念直接来源于现实生活，易于理解。一个 UNIX 的初始用户最先接受的概念就是文件，但是作为 UNIX 核心重要组成部分之一的文件系统，要做到全面、深入的理解就必较难了。

一个 UNIX 运行系统包含了近 700 个文件（包括目录文件），如果放上源程序，则文件数多达 3700 个，这还不包括大量增长的用户开发的程序。系统 V 全部源程序近 50 万行，其中核心部分代码近 3 万行，分成 180 个文件。这么大量的文件是按照何种规则加以管理的？又是怎样安放在盘上的呢？

用户对文件有多种操作，例如创建、删除、打开、关闭以及读写等等，UNIX 核心的文件子系统又是怎样实现这些操作的，或者说文件子系统提供了哪些功能，以实现用户对文件的各种操作？

当然，文件的操作是指程序（包括 UNIX 核心代码）在运行过程中对文件的操作。运行程序必须在内存，加之程序并发执行和资源共享的要求，对文件的操作，势必涉及到文件管理信息和文件数据本身与系统缓冲区（内存某一部分）之间的联系，这一联系的妥善处理直接关系到文件操作的时空开销。因此，介绍文件系统的同时，总要联系到系统缓冲区的管理。

本书第一部分共分四章，介绍 UNIX 文件、文件系统以及系统缓冲区的内部结构（包括抽象数据结构和对抽象数据结构的操作）。同时给出有关的源程序。为阅读方便，源程序加了少量提示，此类提示一律采用/*【.....】*/的形式。另外，为保持原有程序的风格，除少数行分成二行之外，其余与源代码基本一致。

第一章　文件的内部表示

从用户角度看，UNIX 文件系统具有图 1.1 所示的倒树形层次结构。

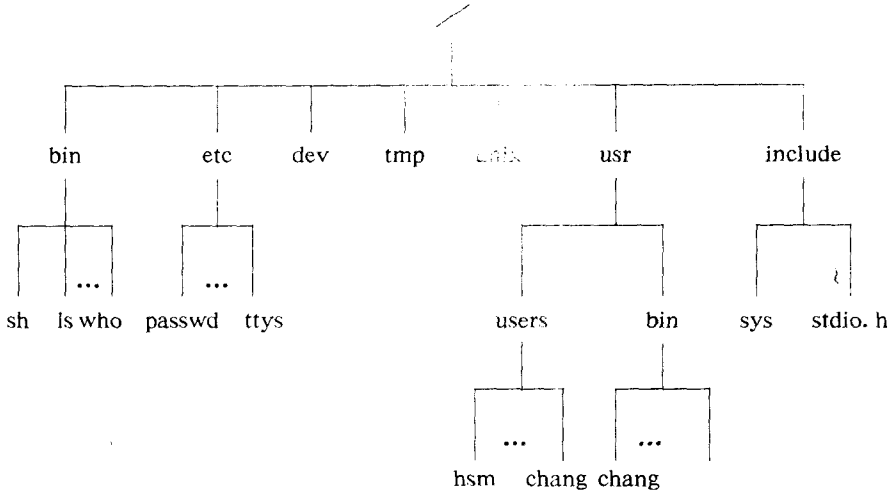

图1.1　树形文件系统示例

"/"称为文件系统的根节点，每个非叶节点都是文件系统的一个目录，而叶节点下的可以是目录，也可以是普通文件，还可以是特殊文件(设备文件)。一个具体文件则用路径名唯一确定，如：

/bin/who

这种树形结构使得 UNIX 用户对文件概念理解总含有一种"位置"的因素，而对进程概念的理解则常伴有"生命周期"的意念，这也是一静一动的意思吧。

UNIX 核心把普通文件、目录文件保存在称为块设备的磁盘或磁带上。一套系统装置可以有若干个物理磁盘，每个物理磁盘可定义成一个或多个文件系统。核心在逻辑一级上只涉及文件系统，而不涉及磁盘。也就是讲，每一个文件系统理解成一个逻辑设备，而逻辑设备到物理设备的地址映射则由磁盘设备驱动程序负责。往后，设备一词除特别说明，均指逻辑设备。

一个文件系统由逻辑块的序列组成，每块为 512 个字节或 1K 字节，同一个文件系统的逻辑块大小一样，但不同文件系统下逻辑块大小可以不同。使用 1K 字节的逻辑块可以增加磁盘与内存之间的数据传输速度。一次盘读写操作中在一个盘块上能存取的数据越多，文件存取就越快。在 BSD4.2 UNIX 系统中就允许 4K 甚至 8K 大小的盘块。但是要注意，增大盘块会增加盘块碎片，造成盘空间的浪费。例如，8K 一块，一个 12K 字节的文件只需 1.5 块，但实际上占了 2 块盘空间，第二块的一半就浪费了。有人做过统计分析，4K 一块的文件系统盘空间的浪费可达 45%！往后的叙述，均假定一块的长度为 1KB。

一个逻辑盘的空间不是全部用来放文件数据，一般是划分为几个用途各不相同的部分，如图 1.2 所示。

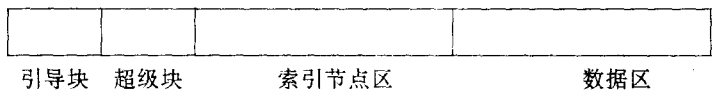

图 1.2　文件系统的静态布局

图中每一个区的用途说明如下：

引导块　——在文件系统的开头，通常为一个扇区，其中存放引导程序，用于读入并启动操作系统。一个系统，只需根文件系统中含引导块。为统一起见，每个文件系统均有一个引导块(可能是空闲的)。

超级块　——记录文件系统(逻辑盘)的当前状态，盘有多大，能存放多少文件，何处可找到空闲空间以及其他用于文件系统管理的信息。

索引节点区——接在超级块后面，存放文件系统的索引节点表，一个文件(包括目录)占据一个索引节点。文件系统格式化时系统管理人员要指明索引节点表的大小。其中第一个索引节点为这个文件系统的根节点，文件系统一般就从这个根节点开始。利用根节点，一个文件系统可以挂在另一个文件系统的非叶节点上。

数据区　——存放文件数据或用于文件管理的其他数据(如后面会讲到的一级间址块、二级间址块)，一个被使用的数据块只能属于某个文件。

本章将从索引节点开始。介绍文件系统的内部表示。

§1.1 磁盘索引节点

UNIX 中每个文件有一个唯一的索引节点，索引节点以静态形式存放在磁盘上，亦称磁盘索引节点。核心对一个文件进行各种操作时，必须把相应的索引节点信息读入内存的特设缓冲区——内存索引节点表，磁盘索引节点实际上是描述文件的第一个抽象数据结构，定义由文件 ino.h 给出：

```
/* @(#)ino.h     1.1 */
/* Inode structure as it appears on a disk block. */
struct dinode
{
    ushort          di_mode;        /* mode and type of file */
    short           di_nlink;       /* number of links to file */
    ushort          di_uid;         /* owner's user id */
    ushort          di_gid;         /* owner's group id */
    off_t           di_size;        /* number of bytes in file */
    char            di_addr[40];    /* disk block addresses */
    time_t          di_atime;       /* time last accessed */
    time_t          di_mtime;       /* time last modified */
    time_t          di_ctime;       /* time created */
};
/*
 *   the 40 address bytes:
 *   39 used; 13 addresses
 *   of 3 bytes each.
 */
```

磁盘索引节点 dinode 诸成分的含义说明如下：

di_mode　——短整数，表示文件类型和存取权限。文件类型包括普通文件、目录文件、字符设备文件、块设备文件以及管道文件等。每一位的详细定义见内存索引节点定义文件 inode.h。

di_size　——32 位长整数，表示文件最后一个字节离开文件起始点的偏移量，实际文件长度为偏移量加一。

di _ addr[40]——字符数组，实际用 39 个字节，每三个字节一组，表示文件数据安放在逻辑盘上的位置。

其余注解见文件中的注释原文。

注意，一个文件在盘上存放不一定是连续的，因此，只有起始块号和长度不足以描述文件在盘上的实际分布情形，文件不连续存放可以减少盘空间的碎片。

粗看上面定义的文件数据盘地址明细表 addr[40]，可能会误会一个文件最长只有 1K×13＝13K，其实，在程序实现时，UNIX 还做了一个巧妙的安排，即所谓直接块、一次间址块、二次间址块和三次间址块，这种办法大大缩短了索引节点的长度。

图 1.3 给出了一个普通文件盘空间安置情况的极端例子。

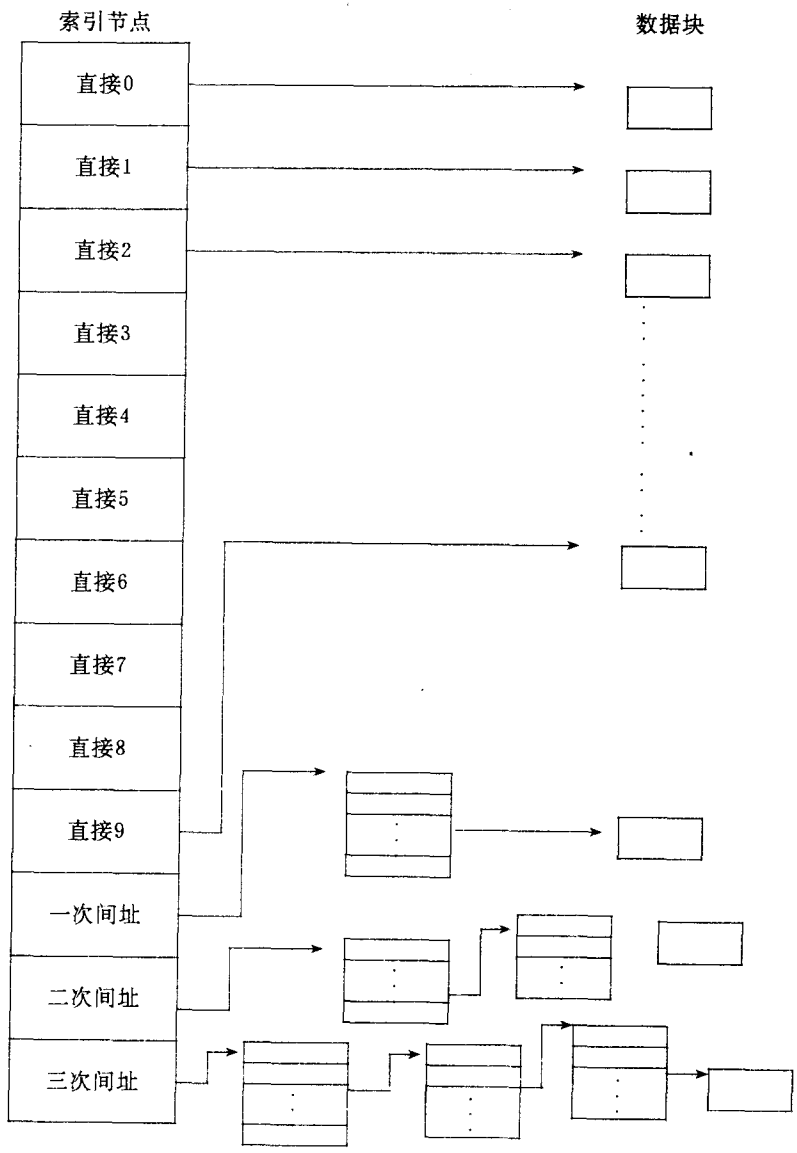

图1.3　索引节点中的直接块和间址块块

从图 1.3 不难看出，一个 UNIX 普通文件的最大长度可以推算如下：

10 个直接块，每个按 1K 字节计最多可容纳的长度＝　　10K 字节

1 个具有 256 个直接块的一次间接块最多可容纳的长度＝　　256K 字节

1 个具有 256 个一次间接块的二次间接块最多可容纳的长度＝64M 字节

1 个具有 256 个二次间接块的三次间接块最多可容纳的长度＝16G 字节

当然，di_size 为长整数，有 32 位，所以一个文件长度不能超过 4 千兆(2 的 32 次方)。采用了间址块的办法，可以大大增长文件的最大长度。但对大文件而言，显然会增加读写操作的时间开销，问题好在 UNIX 系统中大多数文件都少于 10K 字节。例如有人曾经对 19978 个文件进行过统计分析，发现其中 85% 的文件都小于 8K，48% 的文件小于 1K 字节，这当然是一个具体的特例。根据我们自己多年来使用 UNIX 的实践，这一结果基本上还是可信的，对教学单位更是如此。

普通文件的数据不管在盘上如何安放，逻辑上看是一个从 0 字节地址开始直到整个文件长度的无格式的字节流，程序可以按它们自己的实际情况去解释字节流，但这种解释与操作系统如何存放文件数据没有关系。因此，对文件数据的存取由核心提供并且对所有程序都是一样的。我们用一个普通文件的例子说明核心如何由字节偏移量确定数据所存放的盘块号以及块内偏移量。

图 1.4 给出了一个普通文件的数据分布情况。

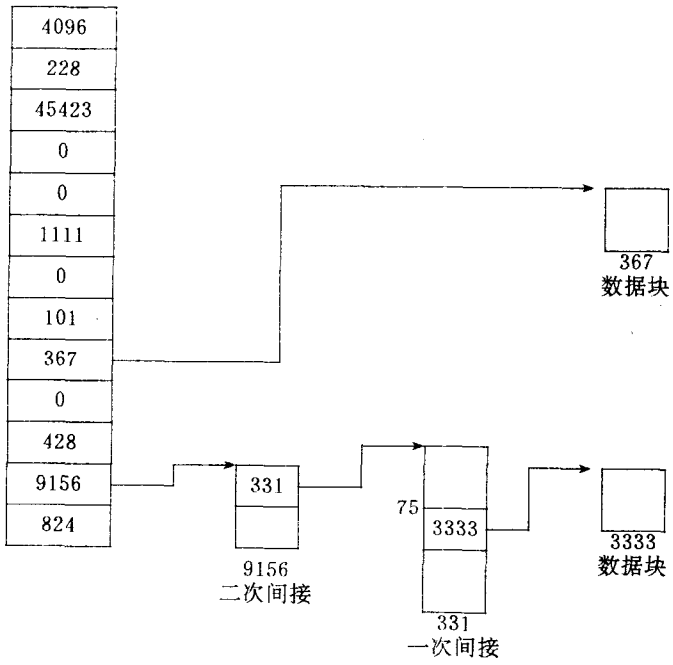

图1.4　一个普通文件的数据分布示例

如果要读取偏移量为 9000 的字节，则不难算出该字节在文件的第 8 个直接块中(从第 0 块直接块算起)。从图中可以看出应取的盘块号为 367 。在这一块的第 808 个字节(从 0 字节算起)就是文件的第 9000 个字节。

如果要读取文件的第 350000 字节，则必须存取一个二次间接块——图中序号为 9156 的那一块。由于一个间接块可容纳 256 个块，所以经由二次间接块所存取的第一个字节是第 272384 (256K＋10K)个字节；从而文件中的第 350000 字节是二次间接块中的第 77616 字节。因为每个一次间接块存取 256K 字节，所以第 350000 字节必定在二次间接块的第 0 个一次间接块中，且块号为 331。因为在一次间接块中每个间接块都包含 1K 字节，所以一次间接块中的第 77616 字节位于一次间接块的第 75 个直接块中——其块号为 3333 。最终得到：文件中的第 350000 字节是第 3333 块中的第 816 字节。

上例由偏移量到盘块号的映射，由文件 subr.c 中的函数 bmap 完成，此函数代码见 § 1.8。

§1.2 目录文件

目录也是文件，引进了目录才使文件系统具有树状层次结构。目录也由盘索引节点表示，只是 di_mode 中标明是目录文件，但其数据组成与普通文件不同。目录的数据是有结构的，它由若干项具有相同结构的登记项组成，每个登记项由一个索引节点号(2 个字节)和一个包含在这个目录下的文件名（14 个字节）组成。见定义文件 dir.h。

```
/* @(#)dir.h      1.1 */
#ifndef DIRSIZ
#define DIRSIZ   14
#endif
struct   direct
{
    ino_t    d_ino;
    char     d_name[DIRSIZ];
};
```

下面给出 etc 目录的部分数据，作为例子。

目录中的字节偏移	索引节点号(2 字节)	文件名
0	83	.
16	2	..
32	1798	init
48	1276	fsck
64	85	ciri
80	1268	motd
96	1799	mount
112	88	mknod
128	2114	passwd
144	1717	umount
160	1815	checklist
176	92	fsdbid
192	84	config
208	1432	getty
224	0	crash
240	95	mkfs
256	188	inittab

上表描绘了目录“etc”的布局。每个目录都包含“.”和“..”的文件名，其索引节点号分别是本目录和它的父目录的索引节点号。在“/etc”中的“.”的索引节点号在该文件中字节偏移量为 0 的位置上，其值为 83。“..”的索引节点号在字节偏移量为 16 的位置上，其值为 2。目录登记项可以为空，由取值为 0 的索引节点号指示。例如，在“/etc”中的第 224 字节上的登记项就为空，虽然它曾经含有过名为“crash”的文件的登记项。程序 mkfs 初始化一个文件系统 以使根目录的“.”和“..”具有该文件系统的根索引节点号。

目录在后面要讲到的由路径名到索引节点号的转换中起着重要作用。目录文件的存取权限表示方法与普通文件一样，但含义不同：

读 ——表示可以读这个目录。

写 ——表示可以增删这个目录下的目录登记项，从而改变目录内容。

执行——表示可以为寻找一个文件而搜索这个目录。

§1.3 内存索引节点

磁盘索引节点反映了文件的静态特征。为了实现对文件的操作，在系统空间内还有一张内存索引节点表，它由若干个内存索引节点组成。内存索引节点除了复制磁盘索引节点的静态信息以外还包含了当前打开文件的动态信息。详细内容见文件 inode.h。

```
/* @(#)inode.h   1.4 */
/*
 * The I node is the focus of all
 * file activity in unix. There is a unique
 * inode allocated for each active file,
 * each current directory, each mounted--on
 * file, text file, and the root. An inode is 'named'
 * by its dev/inumber pair. (iget/iget.c)
 * Data, from mode on, is read in
 * from permanent inode on volume.
 */

#define NADDR    13
#define NSADDR   (NADDR*sizeof(daddr_t)/sizeof(short))

struct  inode
{
    struct inode *i_forw;    /* hash chain forw */
    struct inode *i_back;    /* hash chain back */
    char     i_flag;
    cnt_t        i_count;        /* reference count */
    dev_t        i_dev;          /* device where inode resides */
    ino_t        i_number;       /* i number, 1-to--1 with device address */
    ushort       i_mode;
    short        i_nlink;        /* directory entries */
    ushort       i_uid;          /* owner */
    ushort       i_gid;          /* group of owner */
    off_t        i_size;         /* size of file */
    struct {
        union {
            daddr_t i_a[NADDR]; /* if normal file/directory */
            short   i_f[NSADDR];     /* if fifo's */
        } i_p;
        daddr_t i_l;            /* last logical block read (for read-ahead) */
    } i_blks;
};
extern struct inode inode[];     /* The inode table itself */
```