# Base

# 第二章 进程管理

多道程序环境，要是程序运行，必须先为之创建进程。创建进程第一件事，便是将程序和数据装入内存。如何将一个用户源程序变为一个可在内存中执行的程序，通常要一下几个步骤：

* 编译compiler：编译程序将用户源代码编译成若干个**目标模块**，生成可执行目标集合。
* 链接link：链接程序将编译后形成的一组目标模块，及它们需要的库函数链接在一起，形成一个完整的**装入模块**。将各个模块合成一个整体
* 装入load：装入程序将装入模块装入内存，构造PCB，形成进程，开始运行(使用物理地址)。

# 处理机调度与死锁

# 第四章 存储器管理

## 4.1 存储器的层次结构

[磁盘I/O那些事](https://cloud.tencent.com/developer/article/1058303)

## 4.2 程序的装入和链接

程序的装入

* 绝对装入：逻辑地址就是物理地址，程序中使用的绝对地址，既可在编译或汇编时给出，也可由程序员直接赋予（程序修改要改所有地址）。通常程序采用符号地址，在编译或汇编时，将这些符号地址转换为绝对地址。只适合早期对硬件直接编程、单道环境下使用。
* 可重定位：绝对装入只能将目标模块装入到内存事先指定的位置，可重定位根据内存情况装入适当位置。**地址转换通常再装入时一次完成**，**以后不再改变**，故称为静态重定位。
* 动态运行重定位装入：重定位不允许程序运行时在内存中移动位置。实际运行中在内存中位置可能经常要改变。动态运行时的装入程序把装入模块装入内存后，并不立即把装入模块中的相对地址转换为绝对地址，而是推迟到程序执行，**通过重定位寄存器动态转换**。

程序的链接

根据链接时间不同，链接分为如下三种

* 静态链接：
* 装入时动态链接：目标模块在装入内存时，采用边装入边链接的链接方式。
* 运行时动态链接：改进装入是链接，推迟到程序执行时链接，按需加载，节省内存。

## 4.3 连续分配方式

连续分配方式，是指为一个用户程序分配一个连续的内存空间。它主要包括单一连续分配、固定分区分配和动态分区分配。

### 4.3.1 单一连续分配

内存分为系统区和用户区。只能用于单用户，单任务的操作系统

### 4.3.2 固定分区分配

可运行多道程序，用户区内存分为若干固定大小区域。**每个分区只装入一道作业**。

划分分区的方法

* 分区大小相等。
* 分区大小不相等。根据程序的大小分配适当的分区。

内存分配

为便于内存分配，通常将分区按大小排队，并为之建立一张**分区说明表**，其中各表项包括每个分区的起始地址、大小及状态（是否已分配），如图a。当有用户程序要装入时，便检索该表，以找到合适的分区给予分配并将其状态置为”已分配”。未找到合适分区则拒绝为该用户程序分配内存。存储空间的分配情况如图(b)所示



### 4.3.3 动态分区分配

根据进程的需要，动态分配内存空间。这种分区方法不预先将内存划分，而是在进程装入内存时，根据进程的大小动态地建立分区，并使分区的大小正好适合进程的需要。因此系统中分区的大小和数目是可变的。

分区分配算法

* 首次适应(FirstFit)算法：空闲分区以地址递增的次序链接。分配内存时顺序查找，找到大小能满足要求的第一个空闲分区。
* 最佳适应(BestFit)算法：空闲分区按容量递增形成分区链，找到第一个能满足要求的空闲分区。
* 最坏适应(WorstFit)算法：又称最大适应(Largest Fit)算法，空闲分区以容量递减的次序链接。找到第一个能满足要求的空闲分区，也就是挑选出最大的分区。
* 邻近适应(NextFit)算法：又称循环首次适应算法，由首次适应算法演变而成。不同之处是分配内存时从上次查找结束的位置开始继续查找。

### 4.3.4 伙伴系统

固定分区和动态分区方式都有不足之处。固定分区限制了活动进程数目，当进程大小与空闲分区大小不匹配时，内存空间利用率很低。动态分区方式算法复杂，回收空闲分区需要进行分区合并等，系统开销较大。伙伴系统时以上两种内存方式的折衷方案

### 4.3.5 对换

当内存空间还是满足不了需求时，引入“对换”思想：

在多道程序环境下，在内存中的某些进程由于某事件尚未发生而被阻塞运行，但它却占用了大量的内存空间，另一方面，却有很多作业在外存上等待，因无内存而无法进入内存运行，这是对系统资源的浪费.

对换把内存中暂时不能运行的进程或者不用的程序和数据调出到外存上，腾出空间，再把已具备运行条件的进程调入内存。对换单位分类：

* 整体对换(或进程对换)：以整个进程为单位，广泛用于分时系统。
* 页或段对换：以页或段为单位。支持虚拟存储系统。

实现进程对换，系统必须具备的功能：对换空间的管理；进程的换出、换入操作

对换空间的管理

在具有对换功能的OS中，通常把外存分为文件区和对换区。前者用于存放文件，后者用于存放内存换成的进程。

进程的换出、换入操作

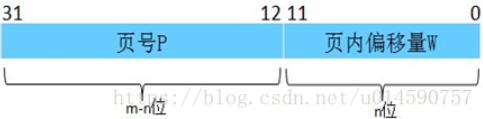
## 4.4 基本分页存储管理方式

* 连续分配：形成很多碎片，虽然通过紧凑方法可整理碎片，但付出很大开销。
* 离散分配:一个进程直接分**散地装入到不相邻接分区中**。分配单位是页称为分页存储管理，分配单位是段，称为分段存储管理。**无需紧凑**。
* 基本分页管理：不具备页面对换功能，不支持虚拟存储功能，要求把每个作业全部装入内存方能运行。

### 4.4.1 页面和页表

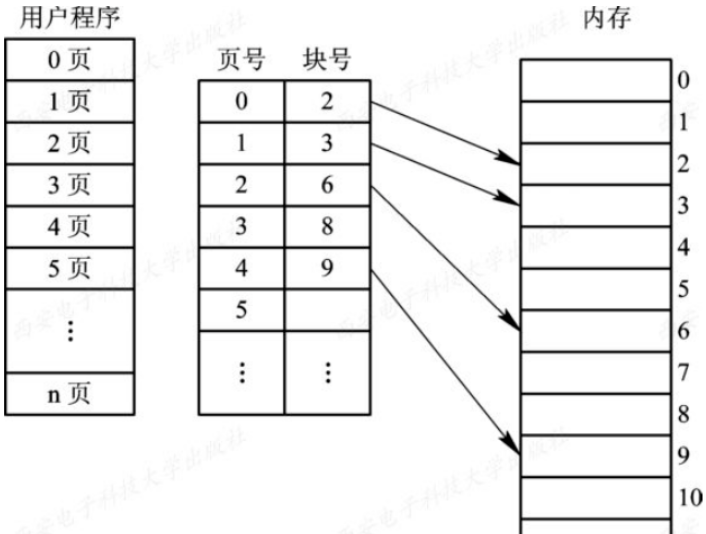
页面

* 页面或页：一个进程的逻辑地址空间分成若干个大小相等的片
* 块或页框：内存分成与页面相同大小的若干个存储块。
* 页面大小:大小应适中。太小虽可减小碎片，但每个进程占用的页面多，导致页表过长，占用内存，还会降低页面换进换出的效率。若页面较大，虽可减少页表长度，提高页面换进换出的速度，却又会使页内碎片增大。页面大小应是2的幂，通常为512B~8KB。
* 地址结构：页号P+页内位移量W（或称为页内地址）。如下，若逻辑地址空间是2的m次方，页面大小是2的n次方（字节），那么逻辑地址的高m-n位是页号，低n位是页内偏移量。



页表

实现从页号到物理块号的地址映射。

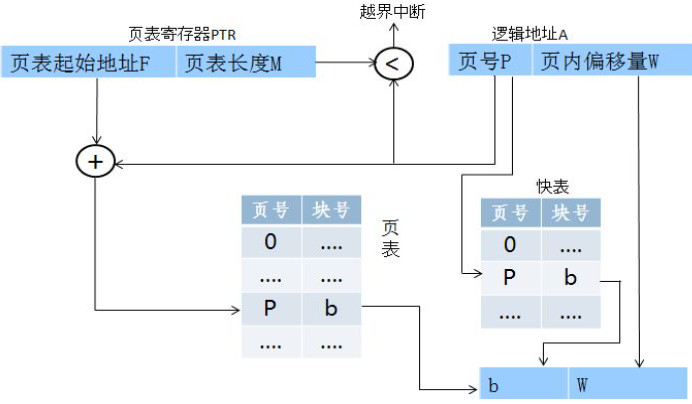


如图：在进程地址空间内所有页（0~n），依次在页表中有一页表项，记录了相应页在内存中对应的物理块号。进程执行时，通过查找该表，即可找到每页在内存中的物理块号。

### 4.4.2 地址变换机构

该机构的基本任务是实现从逻辑地址到物理地址的转换。地址变换任务是借助于页表来完成的

基本的地址变换机构



页表存在内存，这使CPU在每存取一个数据时，都要两次访问内存。

* 第一次访问页表，找到指定页的物理块号，再将块号与页内偏移量W拼接，形成物理地址。
* 第二次访问内存时，才是从第一次所得地址中获得所需数据（或向此地址中写入数据）。

采用这种方式将使计算机的处理速度降低近1/2.可见，以此高昂代价来换取存储器空间利用率的提高，是得不偿失的。

具有块表的地址变换机构

增设高速缓冲寄存器，或称块表。缓存页表。

### 4.4.3 两级或多级页表

## 4.5 基本分段存储管理方式

为了提高内存利用率，存储管理方式从固定分区到动态分区分配，进而又发展到分页存储管理。引入分段存储管理方式主要是为了满足用户（程序员）在编程和使用上多方面的要求

## 4.6 虚拟存储器

前面介绍的各种存储器管理方式有一个共同的特点，即**要求将一个作业全部装入内存后方能运行**。于是，出现了下面的两种情况：

* 有的作业很大，所需空间超过了内存总容量，不能全部被装入内存，致使该作业无法运行。
* 有大量作业要求运行，但由于内存容量不足以容纳所有这些作业，只能将少数作业装入内存让它们先运行，而将其他大量的作业留在外存上等待。

# 第五章 设备管理

## 5.1 IO系统

## 5.2 IO控制方式

### 5.2.1 程序IO方式

在早期的计算机系统中，由于无中断机构，处理机对I/O设备的控制，采取程序I/O方式（Programmed I/O方式）。在程序I/O方式中，由于CPU的高速性和I/O设备的低速性，致使CPU 的绝大部分时间都处于等待I/O设备完成数据I/O的循环测试中，造成对CPU的极大浪费。在该方式中，CPU之所以要不断地测试I/O设备的状态，就是因为在CPU中无中断机构，使 I/O设备无法向CPU报告它已完成了一个字符的输入操作

程序I/O方式又称忙–等待方式，即在处理机向设备控制器发出一条I/O指令启动输入设备、输出数据时，要同时把状态寄存器中的忙/闲标志busy置为1，然后便不断地循环测试busy。当busy=1时，表示输入机尚未输完一个字（符），处理机应继续对busy进行测试；直至busy=0，表明输入机已将输入数据送入控制器的数据寄存器中，于是处理机将数据寄存器中的数据取出，送入内存指定单元中，接着，再启动去读下一个数据，并置busy=1。 △ 此方式造成对CPU的极大浪费。

### 5.2.2 中断驱动IO控制方式

在现代计算机系统中，对I/O设备的控制，广泛采用中断驱动（Interrupt—Driven）方式。在I/O设备输入每个数据的过程中，由于无须CPU干预，因而可使CPU与I/O设备并行工作。仅当输完一个数据时，才需CPU花费极短的时间去做些中断处理。可见，这样可使CPU和I/O设备都处于忙碌状态，从而提高了整个系统的资源利用率及吞吐量。

当某进程要启动某个I/O设备工作时，便由CPU向相应的**设备控制器**发出一条I/O命令，然后立即返回继续执行原来的任务。设备控制器便按照该命令的要求去控制I/O设备。此时，CPU与I/O设备并行操作。　 例如，从终端输入一个字符的时间约为 100ms ， 而将字符送入终端缓冲区的时间小于 0.1ms 。 若采用程序 I/O 方式， CPU 约有 99.9ms 的 时间处于忙 — 等待中。 采用中断驱动方式后， CPU 可利用这 99.9 ms 的时间去做其它事情，而仅用 0.1 ms 的时间来处理由控制器发来的中 断请求 。 可见，中断驱动方式可以成百倍地提高 CPU 的利用率。△ 中断驱动方式可以成百倍地提高CPU的利用率。

### 5.2.3 直接存储器（DMA）IO控制方式

DMA控制方式的引入

虽然中断驱动I/O比程序I/O方式更有效，但它是以字（节）为单位进行I/O的，若将这种方式用于块设备的I/O，显然将会是极其低效的。为了进一步减少CPU对I/O的干预，而引入了直接存储器访问（Direct Memory Access）方式

数据传输的基本单位是数据块；所传输的数据是从设备直接送入内存的,或者相反；整块数据的传送是在控制器的控制下完成的

### 5.2.4 IO通道控制方式

I/O通道方式是DMA方式的发展，它可进一步减少CPU的干预，即把对一个数据块的读写为单位的干预，减少为对一组数据块的读写及有关的控制和管理为单位的干预。同时，又可实现CPU、通道和I/O设备三者的并行工作，从而更有效的提高了整个系统的资源利用率。

通道程序

通道是通过执行通道程序，并与设备控制器来共同实现对I/O设备的控制的。通道程序是由一系列的通道指令（或称通道命令）所构成。通道指令与一般的机器指令不同，在它的每条指令中包含下列诸信息：

## 5.3 缓冲管理

## 5.4 IO软件

### 5.4.2 中断处理程序

### 5.4.3 设备驱动程序

### 5.4.4 设备独立性软件

### 5.4.5 用户层的IO软件

## 5.6 磁盘存储器的管理

# 第六章 文件管理

## 6.1 文件和文件系统

### 6.6.1 文件，记录和数据项

数据项：字段

记录：一组相关数据项。对象，一条数据。记录有一个唯一标识。

文件：分为有结构文件（若干记录，看成表）和无结构文件（看成字符流）。

### 6.6.2 文件类型和文件系统模型

文件类型

按用途分类：

* 系统文件：
* 用户文件：用户的源代码，目标文件，可执行文件或数据所构成的文件。
* 库文件：标准子例程构成，允许用户调用

按组织形式和处理方式分类

* 根据文件的组织形式和系统对其处理方式
* 普通文件：由ASCII码或二进制码组成的字符文件。
* 目录文件：由文件目录组成，目录文件也是由字符序列构成，因此对其操作同普通文件。
* 特殊文件：特指各类IO设备。

### 6.6.3 文件操作

用户通过文件系统提供的系统调用实施对文件的操作。

* 创建文件：系统先分配必要的外存空间，并在文件系统目录中建立目录项。
* 读文件：须在系统调用中给出文件名和应读入的内存目标地址。系统查找目录，得到外存位置。

## 6.2 文件的逻辑结构

### 6.2.1 文件逻辑结构类型

文件的逻辑结构有结构文件和无结构文件。

有结构文件

指由一个以上的记录构成的文件，每个记录用于描述实体集的一个实体。

* 定长记录
* 变长记录

据用户和系统的需要，可采用多种方式来组织这些记录

* 顺序文件：记录按照某种顺序排列所形成的文件，
* 索引文件：记录为可变长度时，通常建立一张索引表，并为每个记录设置一个表项，加快对记录检索的速度
* 索引顺序文件：上述两种的结合，为文件建立一张索引表，为每一组记录中的第一个记录设置一个表项。

无结构文件

源程序，可执行文件等采用无结构文件形式，即流式文件。其长度以字节为单位。

### 6.2.2 顺序文件

逻辑记录的排序

* 串结构：记录之间的顺序与关键字无关，通常由时间决定。
* 顺序结构：记录按关键字排列。

顺序文件的优缺点

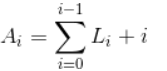
* 批量存取时，即每次要读或写一大批记录。效率是所有逻辑文件中最高的；
* 在交互应用的场合，如果用户(程序)要查找或修改单个记录，系统便要去逐个地查找诸记录。这时，顺序文件所表现出来的性能就可能很差，尤其是当文件较大时，情况更为严重。
* 增加或删除困难。

### 6.2.3 索引文件

定长记录文件，查找第i个记录，可根据下式获得第i个记录相对于第一个记录的地址：



然而，对于可变长记录的文件，要查找第i个记录时，必须顺序地查找前i-1个记录，从而获得相应记录的长度Li，然后才能计算出第i个记录的首址：



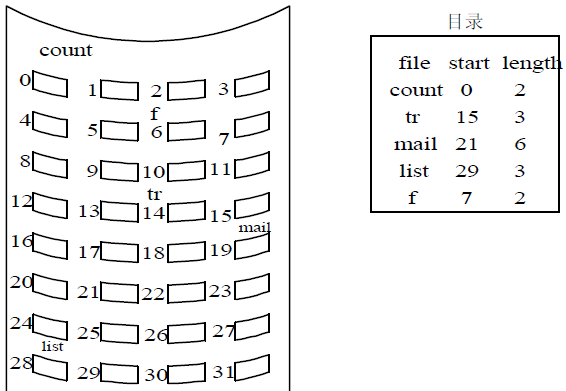
可见，定长记录，除了可以方便实现顺序存取外，还可方便地实现直接存取。而变长记录较难实现存取了。

## 6.3 外存分配方式

文件的物理结构直接与外存分配有关。

### 6.3.1 连续分配

每个文件分配一组相邻接的盘块。通常，它们都位于一条磁盘道上，读写时不必移动磁头。这样形成的文件结构称为顺序文件结构，此时的物理文件称为顺序文件。



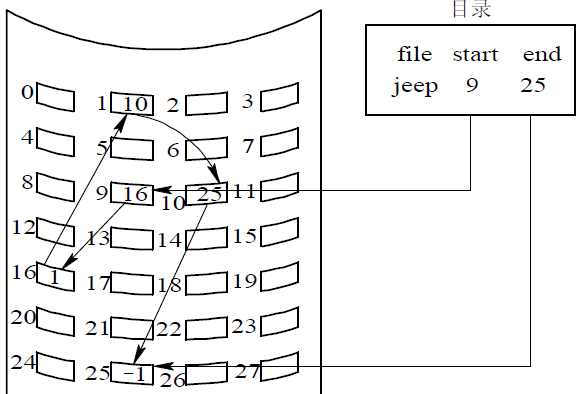
* 优点：顺序访问容易。访问速度快。
* 缺点：1.连续的存储空间。会产生碎片，降低外存利用率。2.必须事先知道文件长度。有时很难知道。有的文件动态增长。

### 6.3.2 链接分配

离散分配。

隐式链接

目录中每个目录项包括指向链接文件第一盘块和最后一个盘块的指针。

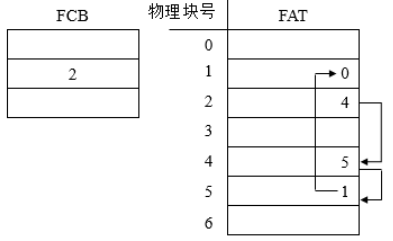


缺点

* 隐式链接只适用于顺序访问，对随机访问时极其低效的。
* 隐式链接可靠性差，只要其中任一指针出错，都将导致整个链断开。

显示链接

把用于链接文件各物理块的指针，存放在内存的一张链接表中。该表在整个磁盘仅设置一张。



表的序号从0开始，直至N-1，N为盘块总数，在每个表项中存放链接指针，即下一个盘块号。

在该表中，文件的第一个盘块号填入FCB。

由于分配给文件的所有盘块号都在该表中，故称该表为文件分配表FAT（File Allocation Table）。查找记录的过程在内存中进行的，因而提高了检索速度，减少了访问磁盘的次数。

磁盘上有多少块，这个FAT表就有多少项，磁盘上的块和FAT中的项一一对应。

### 6.3.4 索引分配

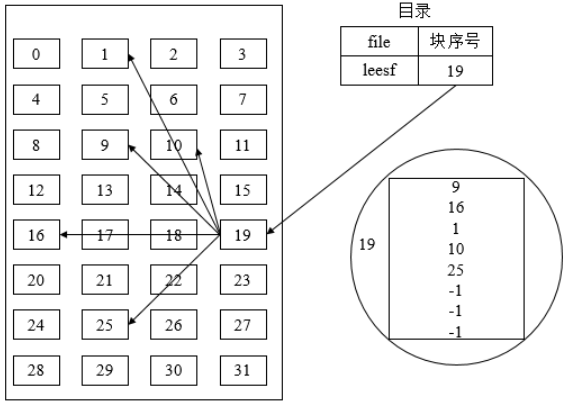
单级索引分配

链接分配方式虽然解决了连续分配方式所存在的问题，但又出现另外两个问题：

(1)不能高效的直接存取。存取一个较大的文件，需首先在FAT中顺序地查找许多盘块号。

(2)FAT占用较大的内存空间。一个文件所占的盘块号随机分布在FAT，只有将整个FAT调入内存。

为每个文件分配一个索引表，记录分配给文件的所有盘块号。在建立一个文件时，只需要在为之建立的目录项中填上指向该索引块的指针。

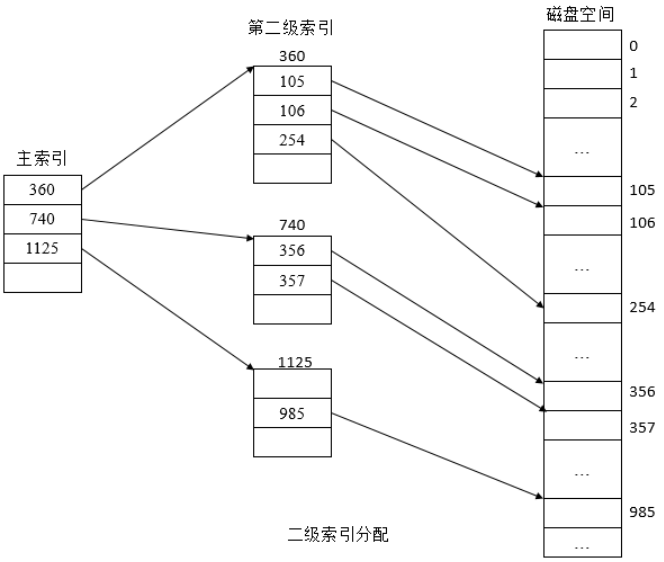


支持直接访问。当要访问第i个盘块时，可直接从索引块找到盘块号。

问题：每个文件分配一个索引块，可能花费较多的外存空间（小文件）。

多级索引分配

当文件太大时，索引块太多，单级索引是低效的。此时，为这些索引块再建立一级索引，称为第一级索引，还可再建立索引，称为第二级索引等等。称为多级索引分配。



每个盘块号占4Byte(i)

|  |  |  |  |
| --- | --- | --- | --- |
| 盘块size(A) | 盘块号 | 一级索引 | 二级索引 |
| 1KB | 256=2^8 | 256\*1K=256K | 256 × 256\*1K = 64 M |
| 4KB | 1024=2^10 | 1024\*4K=4M | 1024\*1024\*4K=4G |
| 16KB | 4096=2^12 | 4096\*16KB=64M | 4096\*4096\*16K=256G |

盘块号=A/i ,若完全使用4b，盘块号为2^32,最大空间达46T

## 6.4 目录管理

文件目录也是一种数据结构，用于标识文件及物理地址，供检索时使用。

### 6.4.1 文件控制块和索引节点

为了能对一个文件进行正确的存取，必须为文件设置用于描述和控制文件的数据结构，称之为”文件控制块“。文件控制块和文件一一对应，而文件控制块的有序集合被称之为文件目录，即一个文件控制块就是一个文件目录项。

文件控制块

FCB：File Control Block。通常含有三类信息：基本信息、存取控制信息、使用信息。

1）基本信息

* 文件名
* 文件物理位置
* 文件逻辑结构
* 文件的物理地址

2）存取控制信息

存取控制信息包括文件的存取权限，核准用户的存取权限以及一般用户的存取权限。

3）使用信息

使用信息包括文件的建立日期和时间，文件上一次修改的日期和时间，以及当前使用信息。

索引节点

当文件很多时，文件目录可能要占用大量的盘块，在查找的过程中，先将存放目录文件的第一个盘块中的目录调入内存，然后把用户所给定的文件名和目录项中的文件名逐一对比。若未找到指定文件，则再将下一个盘块中的目录项调入内存。

在检索目录文件时，只用到了文件名，而其他描述的信息用不到。在有的系统，如UNIX，采用了把文件名和文件描述信息分开的方法，使文件描述信息单独形成一个称为索引结点的数据结构，简称为i结点，在文件目录中的每个目录项由文件名和指向该文件所对应的i结点的指针所构成。

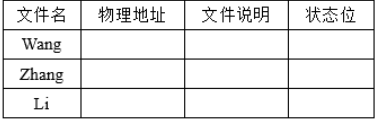


磁盘索引节点

### 6.4.2 目录结构

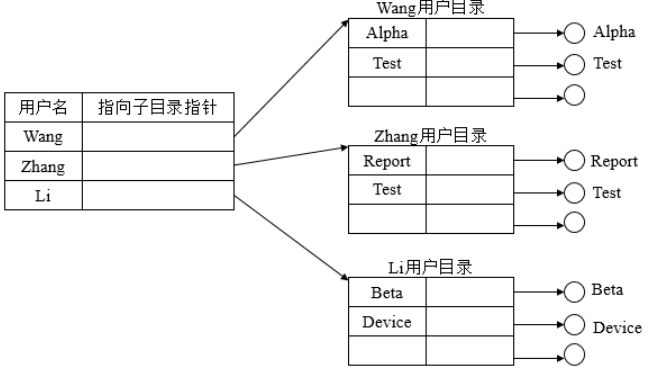
单级目录结构

整个系统只建立一张目录表，每个文件占一个目录项。单级目录的优点是简单能够实现目录管理的基本功能-按名存取，但是查找速度慢，不允许重名。



两级目录结构

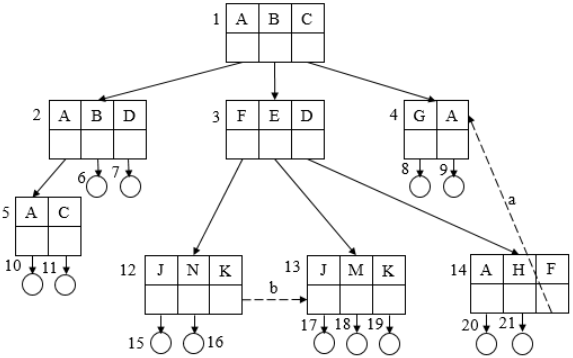
为每个用户建立一个单独的用户文件目录。系统中有一个主文件目录，每个用户目录文件都占有一个目录项，其目录项包括用户名和指向用户目录文件的指针。



两级目录结构提高了检索目录的速度,在不同的用户目录中，可以使用相同的文件名。不同用户还可使用不同的文件名来访问系统中同一个共享文件。

多级目录结构

对于大型文件系统，通常采用三级或三级以上的目录结构，以提高对目录的检索速度和文件系统的性能。多级目录结构又称为树形目录结构，主目录被称为根目录，把数据文件称为树叶，其他的目录均作为树的结点。

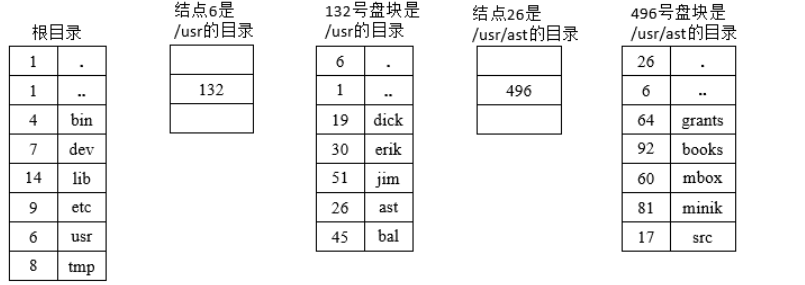


### 6.4.3 目录查询技术

用户要访问一个已存在的文件时，系统首先利用用户提供的文件名对目录进行查询，找出该文件的文件控制块或对应索引结点，然后，根据FCB或索引结点中所记录的文件物理地址（盘块号），换算出文件在磁盘上的物理位置，最后，再通过磁盘驱动程序，将所需文件读入内存。目前常用的方式有线性检索法和Hash方法。

线性检索法

又称为顺序检索法，在树形目录中，用户提供的文件名是由多个文件分量名组成的路径名，此时须对多级目录进行查找，假定用户给定的文件路径名为/usr/ast/mbox，则查找过程如下。



Hash方法

系统利用用户提供的文件名并将它转换为文件目录的索引值，再利用该索引值到目录中去查找，这将提高检索速度

## 6.5 文件存储空间的管理

### 6.5.1 空闲表法和空闲链表罚

空闲表法

空闲表法属于连续分配方式，同内存的动态分配。为每个文件分配一块连续的存储空间，系统也为外存上所有空闲区建立一张空闲表，每个空闲区对应于一个空闲表项，其中包括表项序号、该空闲区的第一个盘块号、该区的空闲盘块号等信息，再将所有空闲区按其起始盘块号递增排列。



分配算法；与内存的动态分配类似，采用首次适应算法，循环首次适应算法等。当文件较小时，采用连续分配方式，当文件较大时，可采用离散分配方式。

空闲链表法

空闲链表法是将所有空闲盘区拉成一条空闲链。链表分成，空闲盘块链和空闲盘区链。

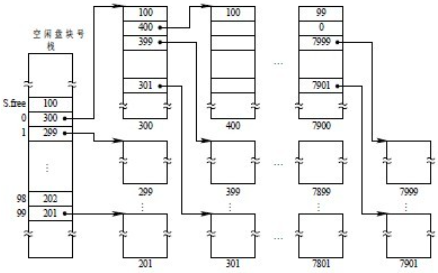
* 空闲盘块链，这是将磁盘上的所有空闲空间，以盘块为单位拉成一条链，当用户因创建文件而请求分配存储空间时，系统从链首开始，依次摘下适当数目的空闲盘块分配给用户，当删除文件而释放空间时，系统将回收的盘块依次插入空闲盘块链的末尾，其优点是用于分配和回收一个盘块的过程简单，但在为文件分配盘块时，可能要重复操作多次。
* 空闲盘区链，这是将磁盘上的所有空闲盘区（每个盘区可包含若干个盘块）拉成一条链，在每个盘区上除了含有只是下一个空闲盘区的指针外，还应有能指明本盘区大小（盘块数）的信息。盘区分配与内存的动态分配类似，可采用首次适应算法，在回收盘区时，同样也要将回收区和相邻接的空闲盘区相合并，在采用首次适应算法时，可以采用显式链接法提高检索速度，在内存中为空闲盘区建立一张链表。

### 6.5.2 位示图法

### 6.5.3 成组链接法

闲表法和空闲链表法都不适用于大型系统，因为这会使空闲表或空闲链表很长，在UNIX采用的成组链接法，结合上述两种方法。

* 空线盘块的组织，空闲盘块栈用来存放当前可用的一组空闲盘块的盘块号（最多含100个号），以及栈中尚有的空闲盘块号数N，顺便指出，N兼做栈顶指针使用，栈是临界资源，系统设置一把锁供进程互斥访问。其中，S.free(0)是栈底，栈满时栈顶为S.free(99)。
* 文件区中的所有空闲盘块被分成若干个组，如每100个盘块作为一组。
* 将每一组含有的盘块总数N和该组所有的盘块号记入其前一组的第一个盘块S.free(0)~S.free(99)中，这样，由各组的第一个盘块可链接成一条链。
* 将第一组的盘块总数和所有的盘块号记入空闲盘块号栈中，作为当前可供分配的空闲盘块号。
* 最末一组只有99个盘块，其盘块号分别记入其前一组的S.free(1)~S.free(99)中，而在S.free(0)中则存放0，作为空闲盘块链的结束。



当系统要为用户分配文件所需的盘块时，须调用盘块分配过程来完成。该过程首先检查空闲盘块号栈是否上锁，如未上锁，便从 栈顶取出一空闲盘块号，将与之对应的盘块分配给用户，然后将栈顶指针下移一格。若该盘块号已是栈底，即S.free(0)，这是当前栈中最后一个可分配的 盘块号。由于在该盘块号所对应的盘块中记有下一组可用的盘块号，因此，须调用磁盘读过程，将栈底盘块号所对应盘块的内容读入栈中，作为新的盘块号栈的内 容，并把原栈底对应的盘块分配出去(其中的有用数据已读入栈中)。然后，再分配一相应的缓冲区(作为该盘块的缓冲区)。最后，把栈中的空闲盘块数减1并返回。

在系统回收空闲盘块时，须调用盘块回收过程进行回收。它是将回收盘块的盘块号记入空闲盘块号栈的顶部，并执行空闲盘块数加1操作。当栈中空闲盘块号数目已达100时，表示栈已满，便将现有栈中的100个盘块号，记入新回收的盘块中，再将其盘块号作为新栈底。

# 第七章 UNIX系统内核结构