第八章 文件系统

概念

1. 文件:由操作系统管理的、存储在外存上的、数据的逻辑单元，为程序提供了按名存取的外存使用模式
2. 文件得逻辑结构:程序员和用户看到的文件组织形式,是对外存的抽象,例如记录结构,索引结构等等.
3. 文件的物理结构:文件的存放形式,程序看不到.
4. 文件分配方法:从逻辑块号到物理块号的映射
5. 逻辑块号:将文件按照块的大小划分,块在文件内的编号;
6. 物理块号:块在文件存储区内的编号;
7. **文件分配表**:FAT,**整个文件系统一个文件分配表**,其中的行号对应块号,存储此块的下一块的块号.
   用于链接分配内存的改进.
8. 索引表:索引表记录着文件所在的数据块,非连续分配.存放在磁盘中,单独占用一个盘块.
9. 文件的构成:文件名;描述部分(文件属性),数据部分
10. 索引节点:存访文件属性和文件得位置
11. 索引节点表:**整个系统一个**,存访全部文件属性和位置.
12. 文件包含了文件得信息,包含文件名,文件属性,文件位置.
13. **硬连接**:不同的文件名对应同一个索引节点号.
14. **软连接**:记录所连接文件的路径名
15. 保护域:权限(二元组)的集合,表示一种身份.
16. 访问控制表:从文件的角度描述用户的权限,按照访问矩阵按列压缩.(谁可以访问当前的文件)
17. 用户权限表:从用户角度描述对文件的操作.访问矩阵按行压缩.
18. 物理卷:如硬盘,磁带SSD等提供存储空间.
19. 逻辑卷:也被称为分区,是连续的线性存储空间,用于建立文件系统.
20. 装入程序:装入程序是指可处理所有的与指定的基地址或起始地址有关的可重定位的地址的程序.

内存分配方式

- 连续分配方式
  - 改进:扩展(每次分配一个扩展
  - 缺点:存在内存碎片,文件长度不能动态变化,需要连续的内存空间;
  - 优点:实现简单,存取速度快,支持随机访问和顺序访问.使得作业访问磁盘时需要的寻道数和寻道时间最短.
- 链接分配方式
  - 改进:FAT,文件分配表(系统启动时被读入内存
  - 缺点:不能随机访问(使用FAT是支持随机访问的),只能通过指针顺序访问,可靠性差,存储链接指针需要花费一定的空间.
  - 优点:消除了磁盘空间碎片,大大提高了磁盘空间的利用率.文件大小可以不固定.
- 索引分配
  - 单级索引不能满足大文件的需求,文件很大时需要多个索引表.
  - 多级索引需要访问多次外存依次获得各级索引表,对小文件没有必要使用多级索引;
  - 改进:Unix混合索引策略.
  - **优点**:可以直接查找,便于进行增删,查找效率高;
  - 缺点:索引表对存储空间的开销.

空闲块存储

- 连续空闲空间管理

序号	起始块号	空闲块数量
0	10000	320
...	...	...

存储开销:空闲列表;

时间开销:分配与回收空闲区;

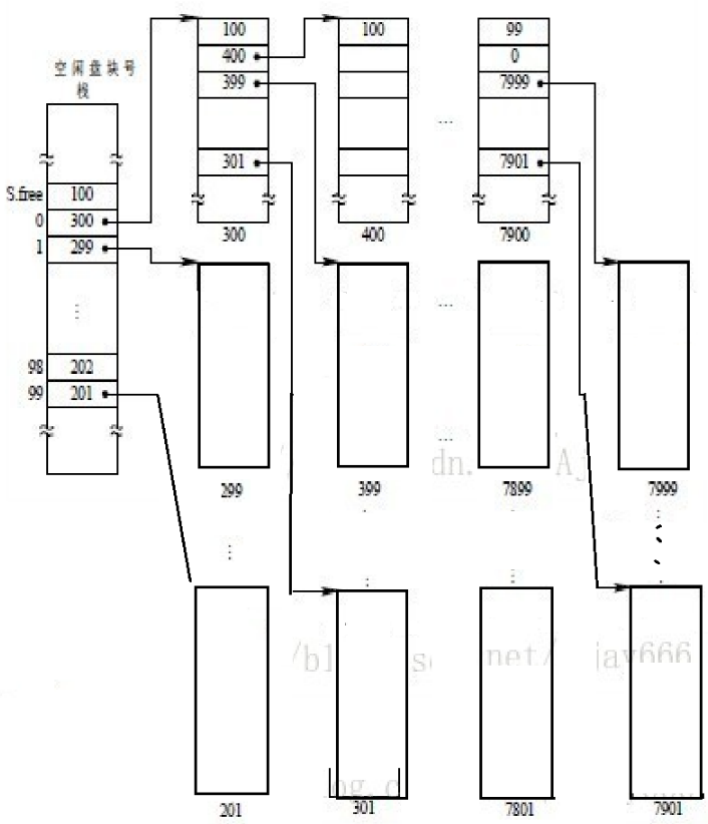
- 空闲块链

空闲块链成一个链表,下一块指针存放在当前块中.

存储开销:一个头指针

时间开销:分配回收空闲块,一次磁盘读写

- 成组空闲块链(存储在内存中



存储开销:第一组索引占用的内存;其他空闲块索引利用空闲块

时间开销:一百次读->一次读磁盘

- 位图

连续的一组存储单元,一位对应一个盘块.

存储开销:连续空间

时间开销:分配空闲块查找位图;回收直接修改

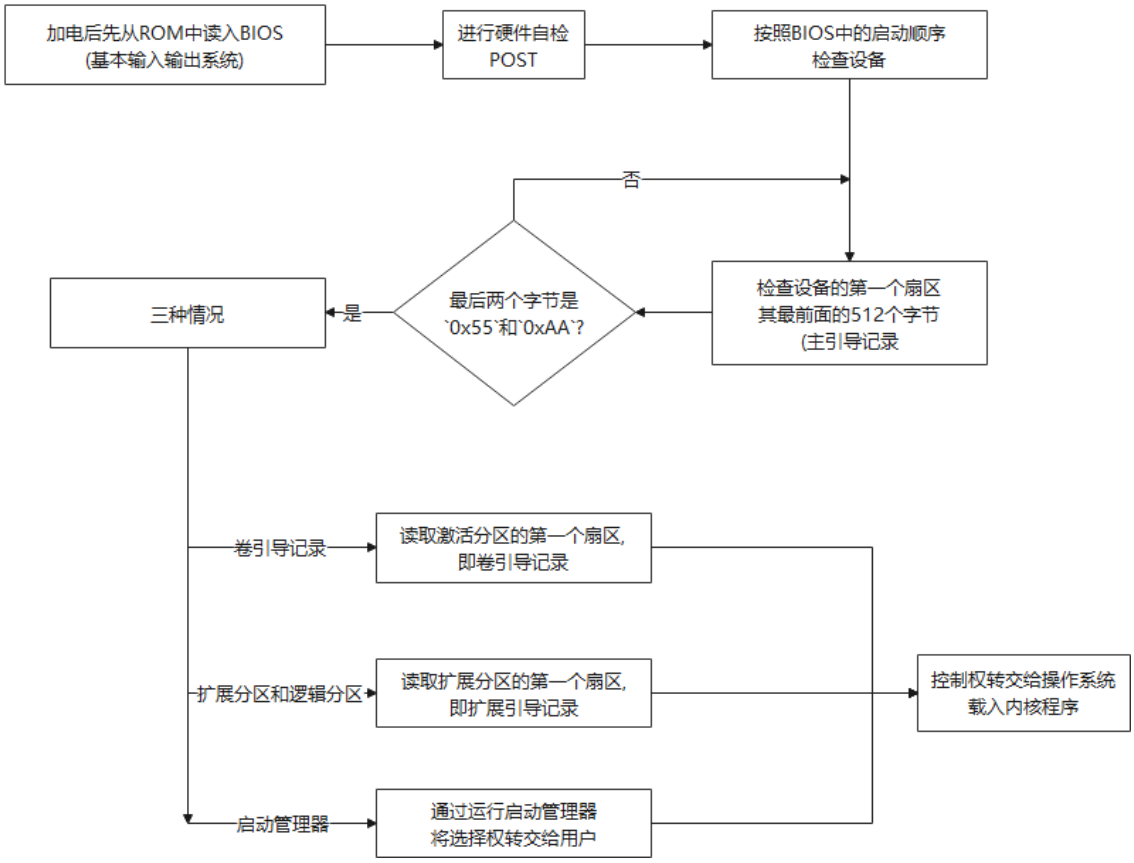
目录

- 简单目录中,文件名与文件一一映射
- 目录也是一种特殊的文件->表

- 通过索引节点号查找索引节点表,获得索引节点,从而可以从中读取到文件属性和文件得位置.
- 有了索引节点,目录中就将文件名和索引节点号进行对应了.
- 文件的查找过程
    - 用文件名的索引节点号检索索引节点表,得到文件的位置;
    - 取出文件的内容;
    - 从中检索下一级文件名,得到下一级文件名的索引节点号;

文件的共享

- 共享的内容
    - 文件数据
    - 文件属性
- 软连接与硬连接
    - 硬连接采用引用计数
    - 软连接采用符号连接
- Linux中文件操作例子:
    - 文件主:RWX
    - 同组用户:RW _
    - 其他用户:_ _ X
- 引导控制块



每个硬盘中只有一个主引导块;

一个扇区最多只能有四个分区(即主引导记录中最多有四个分区表.

第九章 同步与互斥

概念:

- 同步:在并发条件(环境)下,保持操作之间的偏序关系的行为;(由应用程序来实现同步,而不是通过操作系统
- 互斥:对操作之间并发执行的约束.
- 临界资源:不能被并发使用的资源
- 临界区:访问临界资源的程序段,临界区不能并发执行
- 入口代码:执行 判断+等待+标记
- 出口代码:执行 通知+撤销标记
- 忙式等待:如果条件不满足,进入阻塞状态,对应于if语句;
- 非忙式等待:如果条件不满足,一直保持检查条件,对应while语句
- 管程:高级程序设计语言中定义的一种能够用于进程互斥和同步的数据结构,以及能够同步进程和改变管程中数据的一组操作;

同步与互斥

- 异同点:
  - 相同点:
    - 两者都规定了,某两个操作不能同时进行;(时序
    - 不满足条件时都要进入等待状态
    - 运行过程中动态判定
  - 不同点
    - 同步在时序上具有固定的偏序关系;
    - 互斥时序上是临时的偏序关系(不能同时执行
- 实现互斥的方法
  - 软件方法1(**一组变量**):`flag[0]=True`代表线程0要进入临近区;(初值都为`False`

```
线程 T0
do {
      flag[0] = T;
      while (flag[1]);
      // critical section
      flag[0] = F;
      remainder section
   } while (true);
```

```
线程 T1
do {
      flag[1] = T;
      while (flag[0]);
      // critical section
      flag[1] = F;
      remainder section
   } while (true);
```

  - 软件方法2(**一组变量**):`flag[0]=True`代表线程0要进入临界区;与上面的不同点在于初值.

```
线程 T0
do {
      while (flag[1]);
      flag[0] = true;
      // critical section
      flag[0] = false;
      remainder section
   } while (true);
```

```
线程 T1
do {
      while (flag[0]);
      flag[1] = true;
      // critical section
      flag[1] = false;
      remainder section
   } while (true);
```

    想象这样一种情况:

    > T0->运行完flag[0]=false;
    >
    > 线程T1得到运行权,跳出循环;
    >
    > 然后T0得到运行权,进入while失败,运行flag[0]=true;
    >
    > 此时两者都可进入临界区;

- 软件方法3(**两种变量**): `flag[0]=True` 代表线程0要进入, `turn=0` 代表线程0取得进入权;

```
线程 T0
do {
        flag[0] = T;
        turn = 1;
        while (flag[1] && turn == 1);
        // critical section
        flag[0] = F;
        remainder section
    } while (true);
```

```
线程 T1
do {
        flag[1] = T;
        turn = 0;
        while (flag[0] && turn == 0);
        // critical section
        flag[1] = F;
        remainder section
    } while (true);
```

Peterson 方法

  该方法又被称为 `Peterson` 方法;

- 硬件方法(关中断):

  进入临界区前关中断,临界区中线程不被调度,退出临界区后开中断;

  适用于单处理机,适用于内核;

- 硬件方法(TS指令): `Lock=False` 代表开锁, `Lock=True` 代表加锁

  Test_and_set:指令具有**原子性**.

  - bool lock = FALSE;
  - 互斥代码:

```
do {
    while (test_and_set(&lock))
       ; // do nothing
    // critical section
    lock = false;
    // remainder section
} while (true);
```

  巧妙之处在于,每次只有当lock为False的那一次检查会跳出循环,同时会将lock置为True

- 硬件方法(SWAP指令): `Lock=False` 代表开锁, `Lock=True` 代表加锁

  bool lock = 0;

  互斥代码如下:

```
while (true)
{
    key = TRUE;
    while ( key == TRUE)
        Swap (&lock, &key );
    // critical section
    Lock = FALSE;
    // remainder section
}
```

  巧妙之处在于,lock一旦成为false,只会有一个进程的key被交换为false,其他进程再次检测时,lock已经被交换成了true;

硬件方法总结:

ISA提供,指令可在用户态执行;忙时等待导致CPU利用率低;都没有实现有限等待;

不能保证没有插队现象;

- 操作系统的方法(锁):锁实际上是布尔型变量;

  原子操作:`acquire()` 获取锁,加锁;(尝试执行,如果已经加锁,将被阻塞进入死循环

  原子操作:`release()` 释放锁,解锁;

  - Boolean lock = false;
  - 互斥代码:

```
do {
        acquire(lock);
        // critical section
        release(lock);
        remainder section
} while (TRUE);
```

```
acquire() {
      while (lock)
          ; // busy wait
      lock = True;
}

release() {
      lock = false;
}
```

- 操作系统(信号量):PV操作(wait,signal操作)

  信号量分为忙式等待和非忙式等待;

```
Semaphore S;

wait(S) {
      while (S <= 0)
            ; // busy wait
      S--;
}

signal(S) {
      S++;
}
```

```
wait(semaphore *S) {
   S->value--;
   if (S->value < 0) {
      block();// 等待;调度
   }
}
signal(semaphore *S) {
   S->value++;
   if (S->value <= 0) {
      从S->list中选择一个线程唤醒;
   }
}
```

- 高级语言方法(管程):
  - 条件变量具有两个方法:wait,signal;还有一个阻塞队列;

    可以在临界区内挂起进程;
  - 方法:执行互斥
  - 管程相关的队列:条件变量的队列;就绪队列;外围等待队列;