文件管理

文件系统实现

文件系统层次结构

- 层次结构
  - 1.用户调用接口 — 为用户提供与文件及目录相关的系统调用
  - 2.文件目录系统 — 管理文件目录、查目录表、进程打开文件表等
  - 3.存取控制验证模块 — 实现文件保护、访问控制权限验证
  - 4.逻辑文件系统 与 文件信息缓冲区 — 根据文件的逻辑结构将记录号转换为逻辑地址
  - 5.物理文件系统 — 把逻辑地址(逻辑记录所在的相对块号)转换为物理地址
  - 6.辅助分配模块 — 管理辅存空间、分配空间、回收空间
  - 7.设备管理程序模块 — 分配设备、启动、释放、读写等
- 例子 —

> 例如，用户要查看文件 F 中的内容，对操作系统发出命令（操作系统有面向用户的接口），于是就向过了第一层用户调用接口。
>
> 操作系统收到命令后，需要查找目录以查找文件 F 的索引信息，可能是 FCB，也可能是索引块。找过了第二层文件目录系统。
>
> 通过目录找到文件 FCB 后，需要查看文件 FCB 上的信息，看看那个用户有权有读到该文件的权限，于是经过了存取控制验证模块。
>
> 用户通过验证后，就真正开始寻址。对应第 3 重的步骤了。我们的这样的意识，抽象系统的寻址往往要分两段逻辑地址，再将其物理地址。于是在开始寻址时，操作系统给这逻辑文件系统与文件信息缓冲区，得到了相应文件的内容的信息缓冲区。
>
> 把逻辑地址转换为物理地址？是在物理文件系统中完成的。
>
> 至此为止，目录表已经完成，寻址也成功，我们只心中的系统里的连续空间直接如何管理，有要解决该块空间，相应表最交的辅助分配模块，若要把该块空间分配给设备用于输入/输出，则把任务交给设备管理程序模块。

目录实现

- 目录实现的定义及作用
  - 操作系统利用路径名找到相应目录项，目录项中提供了提找文件磁盘块所需要的信息、作用即查找 — 定义/作用
  - 目录查询通过在磁盘上反复搜索，需要不断I/O，开销大；可以将文件目录调度到内存，使用文件时只需要在内存中查目录，降低了磁盘访问次数 — 可以放入内存优化系统速度
- 目录实现的方法
  - 线性列表
  - 哈希表

文件实现(物理结构)

- 文件分配方式(非空闲块)
  - 文件分配的引入目的
    - 如何为文件分配磁盘块
    - 类似内存分页，磁盘的存储单元也被分成一个个块，磁盘块的大小与内存块、页面大小一般相同
    - 操作系统为文件分配存储空间，以块为单位 — 为内存和磁盘之间的IO数据交换以块为单位进行，和读入一页/写出一页一样
    - 用户通过文件逻辑地址(逻辑块号，缺内地址)来操作文件
  - 1.连续分配
    - 实现方案 — 为每个文件在磁盘上分配一组连续的块 — 类似二维内存分配
    - 优点 — 连续分配存取速度快(支持顺序/随机访问)，且实现简单只需给磁盘块排序
    - 优缺点
      - 文件长度不宜动态增加（物理上采用连续分配的文件不宜扩展），类比数组增删需大量移动
      - 会产生外部碎片，存储空间利用率降低，类比内存管理分配
  - 2.链接分配
    - 实现思想 — 采用离散分配的方式，消除了外部碎片，且易增删
    - 分配方式
      - 1.隐式分配
        - 实现方案 — 每个文件对应一个磁盘块的链表，每个磁盘块都有指向下一个盘块的指针，目录包括首尾指针 — 类似链表连接
        - 缺点 — 隐式链式分配无法之间访问盘块，只能通过指针顺序访问文件，且指针消耗存储空间；稳定性差，知指针丢失，导致后续连接数据丢失
      - 2.显式分配
        - 实现方案 — 把用于链接文件各物理块的指针，放在内存的一张链接表，文件分配表FAT在启动时读入内存，用显内存提高检索速度
        - 优点 — 支持顺序访问、随机访问，不会产生外部碎片，方便文件扩展(只需查FAT空闲区域，加载文件链末尾)
  - 3.索引分配
    - 实现方案 — 每个文件的所有磁盘号都集中放在一张索引表中，每个文件都有一个索引块(存放该块址数组)
    - 优点 — 支持随机访问，没有外部碎片
    - 缺点 — 存储开销，若索引块大文件放不下
    - 处理大文件的机制
      - 链接方案 — 一个索引块(就是一个磁盘块本身支持读写)不够，就将多个索引块链接起来，以处理大文件
        - 缺点 — 若想要访问文件的最后一个逻辑块，就必须通过指针顺序找到前面所有的索引块，低效
      - 多层索引 — 使第一层索引块指向第二层索引块，类似多级分页
        - 例题 — 假设磁盘块大小为1KB，一个索引表项占4B，则一个磁盘块只能存放256个索引项。
          若某文件采用两层索引，则该文件的最大长度可以到256*256*1KB = 65,536 KB = 64MB
          只知某逻辑块号即可以查找到该表项中的哪个表项。如，要读取 1026 号逻辑块，则：
          1026/256 = 4，1026%256 = 2
          因此可以先将一级索引表调入内存，查找 4 号表项，将其对应的二级索引表调入内存，再查找二级索引表 的2号表项即可知道 1026 号逻辑块存放的磁盘块号了。
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          !
          ![