

目录的基本概念

这种目录结构对于用户来说有什么好处？

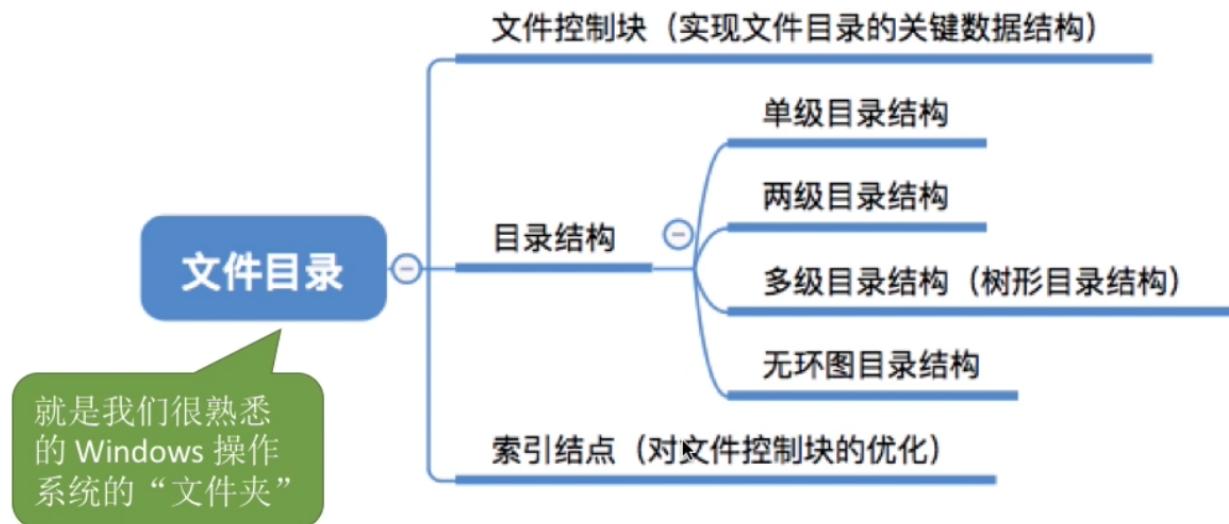
文件之间的组织结构清晰，易于查找

编程时也可以很方便的用文件路径找到一个文件。如：

```
FILE *fp;  
fp=fopen("F:\data\myfile.dat");
```

用户可以轻松实现按名存取

从操作系统的角度来看，这些目录结构应该是如何实现的？



文件控制块



“照片”目录对应的目录文件

文件名	类型	存取权限	物理位置
2015-08	目录	只读	...	外存25号块
2015-09	目录	读/写	...	外存26号块
.....			...	
2016-02	目录	读/写	...	外存27号块
.....				
微信截图_20180826102629	PNG	只读	...	外存995号块

目录文件中的一条记录就是一个“文件控制块（FCB）”

FCB 实现了文件名和文件之间的映射。使用户（应用程序）可以实现“按名存取”

FCB 的有序集合称为“文件目录”，一个FCB就是一个文件目录项。FCB 中包含了文件的基本信息（文件名、物理地址、逻辑结构、物理结构等），存取控制信息（是否可读/可写、禁止访问的用户名单等），使用信息（如文件的建立时间、修改时间等）。最重要，最基本的还是文件名、文件存放的物理地址。

需要对目录进行哪些操作？

- 搜索

当用户要使用一个文件时，系统要根据文件名搜索目录，找到该文件对应的目录项

- 创建文件

创建一个新文件时，需要在其所属的目录中增加一个目录项

- 删除文件

当删除一个文件时，需要在目录中删除相应的目录项

- 显示目录

用户可以请求显示目录的内容，如显示该目录中的所有文件及相应属性

- 修改目录

某些文件属性保存在目录中，因此这些属性变化时需要修改相应的目录项（如：文件重命名）

目录结构

单级目录结构

早期操作系统并不支持多级目录，整个系统中只建立一张目录表，每个文件占一个目录项。

单级目录实现了按名存取，但是不允许文件重名。

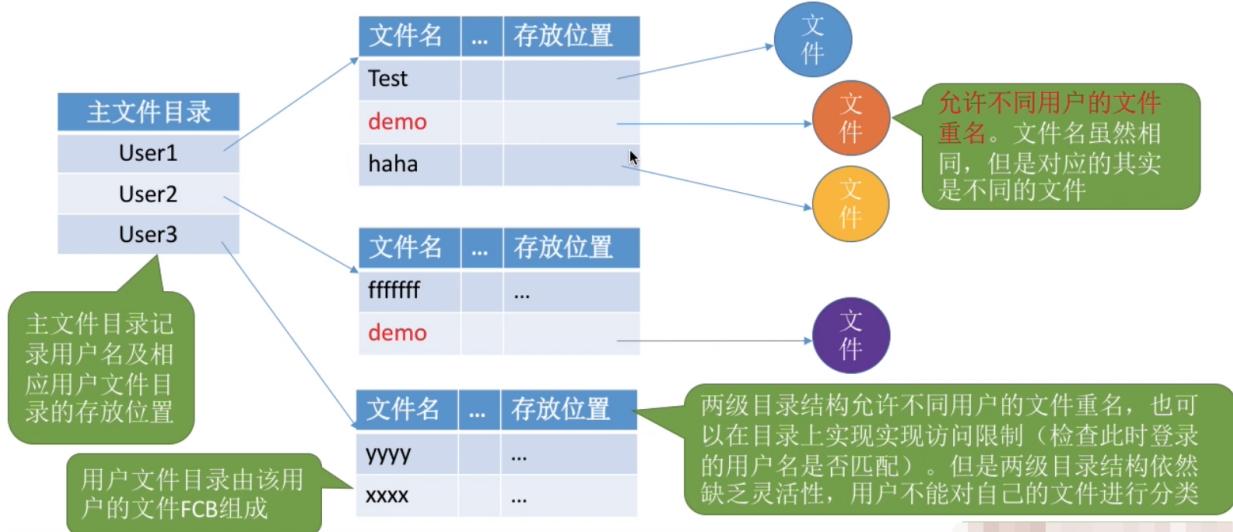
在创建一个文件时，需要先检查目录表中有没有重名文件，确定不重名后才能允许建立文件，并将新文件对应的目录项插入目录表中。

显然，单级目录结构不适用于多用户操作系统。

两级目录结构

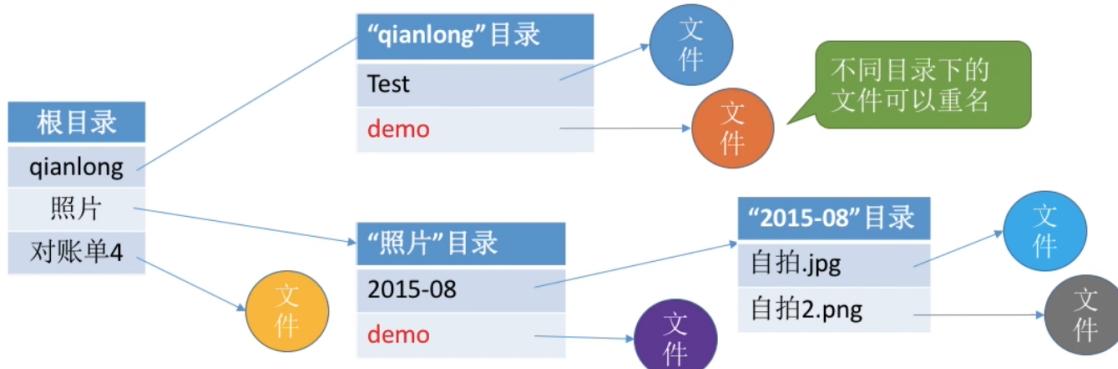
早期的多用户操作系统，采用两级目录结构。分为主文件目录（MFD, Master File Directory）和用户文件目录（UFD, User File Directory）。

早期的多用户操作系统，采用两级目录结构。分为**主文件目录**（MFD，Master File Directory）和**用户文件目录**（UFD，User File Directory）。



多级目录结构

又称树形目录结构



用户（或用户进程）要访问某个文件时要用文件路径名标识文件，文件路径名是个字符串。各级目录之间用“/”隔开。**从根目录出发**的路径称为**绝对路径**。

例如：**自拍.jpg** 的绝对路径是 **“/照片/2015-08/自拍.jpg”**

系统根据绝对路径一层一层地找到下一级目录。刚开始**从外存读入根目录的目录表**；找到“照片”目录的存放位置后，**从外存读入对应的目录表**；再找到“2015-08”目录的存放位置，再**从外存读入对应目录表**；最后才找到文件“自拍.jpg”的存放位置。整个过程**需要3次读磁盘I/O操作**。

很多时候，用户会连续访问同一目录内的多个文件（比如：接连查看“2015-08”目录内的多个照片文件），显然，每次都从根目录开始查找，是很低效的。因此可以设置一个“**当前目录**”。

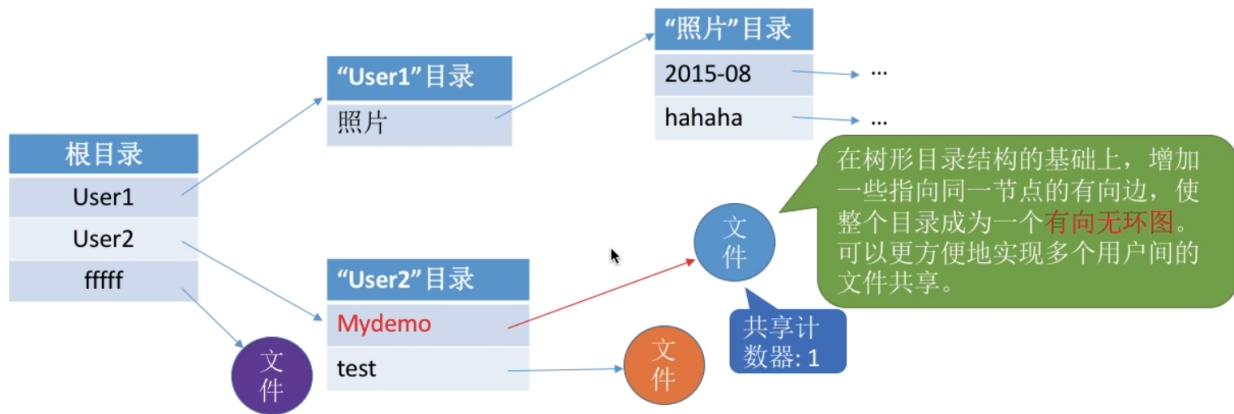
用户（或用户进程）要访问某个文件时要用文件路径名标识文件，文件路径名是个字符串。各级目录之间用“/”隔开。**从根目录出发**的路径称为**绝对路径**。例如：**自拍.jpg** 的绝对路径是 **“/照片/2015-08/自拍.jpg”**

每次都从根目录开始查找，是很低效的。因此可以设置一个“**当前目录**”。例如，此时已经打开了“照片”的目录文件，也就是说，这张目录表已调入内存，那么可以把它设置为“当前目录”。当用户想要访问某个文件时，可以使用**从当前目录出发的“相对路径”**。

在 Linux 中，“.” 表示当前目录，因此如果“照片”是当前目录，则“自拍.jpg”的相对路径为：
“**./2015-08/自拍.jpg**”。

树形目录结构可以很方便地对文件进行分类，层次结构清晰，也能够更有效地进行文件的管理和保护。但是，树形结构**不便于实现文件的共享**。为此，提出了“**无环图目录结构**”。

无环图目录结构



可以用不同的文件名指向同一个文件，甚至可以指向同一个目录（共享同一目录下的所有内容）。
需要为每个共享结点设置一个共享计数器，用于记录此时有多少个地方在共享该结点。用户提出删除结点的请求时，只是删除该用户的FCB，并使共享计数器减1，并不会直接删除共享结点。
只有共享计数器减为0时，才删除结点。

注意：共享文件不同于复制文件。在共享文件中，由于各用户指向的是同一个文件，因此只要其中一个用户修改了文件数据，那么所有用户都可以看到文件数据的变化。

索引结点 (FCB的改进)

索引结点 (FCB的改进)

文件名	类型	存取权限	物理位置
qianlong	目录	只读	...	外存7号块
QMDownLoad	目录	读/写	...	外存18号块
.....			...	
照片	目录	读/写	...	外存643号块
.....			...	
对账单4.txt	txt	只读	...	外存324号块

其实在查找各级目录的过程中只需要用到“文件名”这个信息，只有文件名匹配时，才需要读出文件的其他信息。因此可以考虑让目录表“瘦身”来提升效率。

文件名	索引结点指针
qianlong	
QMDownLoad	
.....	
照片	
.....	
对账单4.txt	

索引
结点
除了文件名之外的文件描述信息都放到这里来

思考有何好处？
假设一个FCB是64B，磁盘块的大小为1KB，则每个盘块中只能存放16个FCB。若一个文件目录中共有640个目录项，则共需要占用 $640/16 = 40$ 个盘块。因此按照某文件名检索该目录，平均需要查询320个目录项，平均需要启动磁盘20次（每次磁盘I/O读入一块）。

若使用索引结点机制，文件名占14B，索引结点指针占2B，则每个盘块可存放64个目录项，那么按文件名检索目录平均只需要读入 $320/64 = 5$ 个磁盘块。显然，这将大大提升文件检索速度。

当找到文件名对应的目录项时，才需要将索引结点调入内存，索引结点中记录了文件的各种信息，包括文件在外存中的存放位置，根据“存放位置”即可找到文件。

存放在外存中的索引结点称为磁盘索引结点，当索引结点放入内存后称为内存索引结点。

相比之下内存索引结点中需要增加一些信息，比如：文件是否被修改、此时有几个进程正在访问该文件等。

