CongJie Huang
CS162
4/8/09

Naming

*Any system has file descriptors, a data structure or record to describe a file.
*File descriptor information has to be stored on disk so it will stay around even when the operating system doesn't. (Assuming the disk contents are permanent.)
*In Unix, the file description is stored in a fixed-size array on disk. File descriptors also contain protection and accounting information.
*A special area of disk is used for this, so that it won't be overwritten.

Inode Fields

*Reference count (# time open) - how many processes open.
*Number of links to file
*Owner's user ID, owner's group ID
*Number of bytes in file
*Time file was last accessed, time last modified, last time Inode changed
*Disk block address, indirect blocks
*Flags: Inode is locked, modified, or waiting on another lock.
*File mode: (type of file: char, special, directory, block special, regular, symbolic link, socket)
*Protection information: (set user ID on execution , set group ID on execution, read, write, execute permission, sticky bit)
*Count of shared locks in Inode
*Count of exclusive locks on Inode
*Unique identifier
*File system associated with this Inode.
*Quota structure controlling this file

Of course, users want to use text names to refer to files. Special disk structures called directories are used to tell what descriptor indices correspond to what names.

Approach #1: Have a single directory for the whole disk. Use a special area of disk to hold the directory.
*Directory contains <name, index> pairs.
*Problems
**If one user uses a name, no one else can.
**If you can't remember the name of a file, you may have to look through a very long list.
**Security problem - people can see you file names (which can be dangerous)
*Old personal computers (pre-Windows) work this way.

Approach #2: Have a separate directory for each user (TOPS-10 approach). This
is still clumsy: names form a user's different projects get confused. Still
can't remember names of files.
 *IBM's VM is similar to this. Files have 3 part names: <name, type,
  location>, where location A,B,A, etc. (i.e. which disk) Very painful.
  (Also file names limited to 8 characters).

Approach #3: Unix Approach: Generalize the directory structure to a tree.
 *Directories are stored on disk just like regular files (i.e. file descripto
  with 13 pointers, etc)
 **User programs can manipulate directories almost like any other file. Only
   special system programs may write directories.
 *Each directory contains <name, file descriptor index (Inode#)> pairs. The
  file pointed to by the index may be in another directory. Hence, get
  hierarchical tree structure. Names have slashes separating the levels of
  the tree.
 *There is one special directory, called the root. This directory has no
  name, and it the file pointed to by descriptor 2 (descriptors 0 and 1 have
  other special purposes).
 **Note that we need ROOT. Otherwise, we would have no way to reach any
   files. From root, we can get anywhere in the file system,
 *Full name is the path name=, i.e. full name from root.
 **A directory consists of some number of blocks of DIRBLKSIZ bytes, where
   DIRBLKSIZ (Directory BLock Size) is chosen such that it can be transferred
   to disk in a single atomic operation (e.g. 512 bytes on most machines)
 **Each directory block contains some number directory entry structures, which
   are of variable length. Each directory entry has info at the front of it,
   containing its Inode number, the length of the entry, and the length of the
   name, contained in the entry. These are followed by the name padded to a
   4-byte boundary with null bytes. All names are guaranteed null
   terminated.
 *Note that in a Unix, a file name is not the name of a file. It is only a
  name by which the kernel can search for the file. The Inode is really the
  "name" of the file.
 **Each pointer from a directory to a file is called a hard link.

 **In some systems, there is a distinction between a "branch" and a "link"
   where the link is a secondary access path, and the branch is the primary
   one (goes with ownership).
 **You "erase" a file by removing a link to it. In reality, a count is kept
   of the number of links to a file. It is only really erased when the last
   link is removed.
 **To really erase a file, we put the blocks of the file on the free list.