CS 3204
Operating Systems

Lecture 22

Godmar Back

Announcements

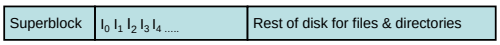
- Project 3 due Nov 11, 11:59pm
- Additional office hours scheduled

Extents

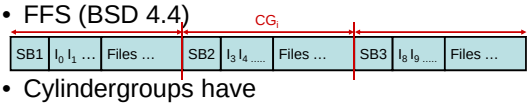
- Index-tree based scheme avoids external fragmentation, and is efficient for small files, but incurs relatively high meta-data overhead for large files
- Extents can improve that – store (bnum, length) pair to denote that file occupies blocks [bnum, … , bnum+length-1]
  - But complicates offset -> sector translation
  - Used in ext4.

Storing Inodes

- Unix v7, BSD 4.3



- FFS (BSD 4.4)



- Cylindergroups have superblock+bitmap+inode list+file space
- Try to allocate file & inode in same cylinder group to improve access locality

Positioning Inodes

- Putting inodes in fixed place makes finding inodes easier
  - Can refer to them simply by inode number
  - After crash, there is no ambiguity as to what are inodes vs. what are regular files
- Disadvantage: limits the number of files per filesystem at creation time
  - Use “df –ih” on Linux/ext3 to see how many inodes are used/free

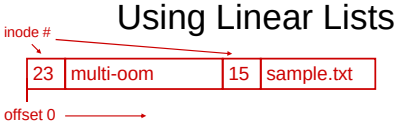
Filesystems

Directories and Name Resolution

Directories

- Need to find file descriptor (inode), given a name
- Approaches:
  - Single directory (old PCs), Two-level approaches with 1 directory per user
- Now exclusively hierarchical approaches:
  - File system forms a tree (or DAG)
- How to tell regular file from directory?
  - Set a bit in the inode
- Data Structures
  - Linear list of (inode, name) pairs
  - B-Trees that map name -> inode
  - Combinations thereof

Using Linear Lists



- Advantage: (relatively) simple to implement
- Disadvantages:
  - Scan makes lookup (& delete!) really slow for large directories
  - Could cause fragmentation (though not a problem in practice)

Using B+-Trees

- Advantages:
  - Scalable to large number of files: in growth, in lookup time
- Disadvantage:
  - Complex
  - Overhead for small directories (some filesystems switch to B+-Tree only for large directories)
- Note: some filesystems use B+-Tree not only for directory files, but for block indexes as well.
  - HFS's 'catalog' – single B+-Tree that stores inodes + directories.
  - Also done in XFS & Reiserfs

Absolute Paths

- How to resolve a path name such as "/usr/bin/ls"?
  - Split into tokens using "/" separator
  - Find inode corresponding to root directory
    - (how? Use fixed inode # for root)
  - (*) Look up "usr" in root directory, find inode
  - If not last component in path, check that inode is a directory. Go to (*), looking for next comp
  - If last component in path, check inode is of desired type, return

Name Resolution

- Must have a way to scan an entire directory without other processes interfering -> need a "lock" function
  - But don't need to hold lock on /usr when scanning /usr/bin
- Directories can only be removed if they're empty
  - Requires synchronization also
- Most OS cache translations in "namei" cache – maps absolute pathnames to inode
  - Must keep namei cache consistent if files are deleted

Current Directory

- Relative pathnames are resolved relative to current directory
  - Provides default context
  - Every process has one in Unix/Pintos
- chdir(2) changes current directory
  - cd tmp; ls; pwd vs (cd tmp; ls); pwd
- lookup algorithm the same, except starts from current dir
  - process should keep current directory open
  - current directory inherited from parent

Hard & Soft Links

- Provides aliases (different names) for a file
- Hard links: (Unix: ln)
  - Two independent directory entries have the same inode number, refer to same file
  - Inode contains a reference count
  - Disadvantage: alias only possible with same filesystem
- Soft links: (Unix: ln –s)
  - Special type of file (noted in inode); content of file is absolute or relative pathname – stored inside inode instead of direct block list
- Windows: “junctions” & “shortcuts”