Operating Systems (CS 3430/5430) Lecture 29 Name:_____

1. File systems need to keep track of which disk blocks are used or free. Some options are:

- bitmap: one bit per block (0 ≡ free and 1 ≡ used)
  + easy to look for space in a region of disk
  - slow to search entire disk for extent of free size

Disk Block #:	0	1	2	3	4	5	6	7	8	9	10 ...
Free block bitmap:	0	1	1	1	0	1	1	0	0	1	0 ....

  a) When looking for an extent of free space of a specified size, would *first-fit* or *best-fit* allocation policy work best?

- divide disk space into chunks known as *block groups* (UNIX/Linux, e.g., 128 MB) with a bitmap for each block group and the OS maintaining the amount of free space per block group.
  b) In ext3fs, why are regular files placed in the same block group as the parent directory if possible?

  c) In ext3fs, similarly new subdirectories placed into the parent directory's block group if it isn't too crowded; otherwise, subsequent block groups looked through for one that's not too crowded. Why does the OS care if the block group is too crowded?

2. A new file being written is often buffered in RAM until it is closed, called *delayed allocation*. What advantages does *delayed allocation* have?

3. Metadata about files includes:

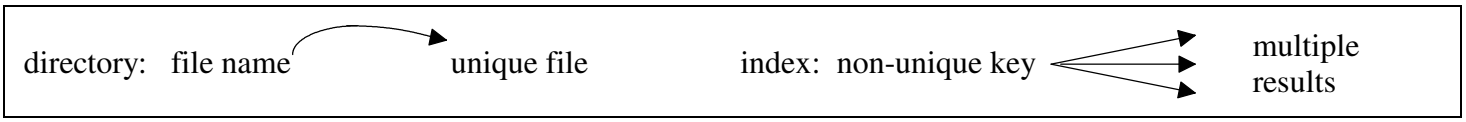
- naming information (reference count if multiple names)
- misc.: timestamps (POSIX: last accessed, written, modified), hidden file or not (NTFS), file's icon location on GUI display (Mac HFS)
- file size in bytes
- location of file's disk blocks:
  - Linux - inodes - sequence of disk block numbers (some indirectly accessed) -- see figure 8.10
  - NTFS - extent map (starting file block, length in blocks, starting disk block) most fit in 1 KB "inode"
  - Mac HFS - first 8 extent map entries in inode with others in B+ tree shared by all files
    a) What would the keys of the B+ need to contain?

  - Linux XFS - stores all extent map entries in a file specific B+ tree -- see figure 8.13
- access control information (ACLs)
  - POSIX - fixed length ACL - three numbers in inode: (1) file's owning user, (2) file's owning group, (3) - 9-bit mode (rwx owner, rwx permission, rwx other permissions), file/directory, I/O device, etc.
  - Windows NTFS - more general ACLs. Implementation evolved over time: early versions full ACLs stored in each file's inode (1 KB); starting with Windows 2000 - store ACLs in centralized database that files reference.
    b) Why is a centralized DB of ACLs more efficient?

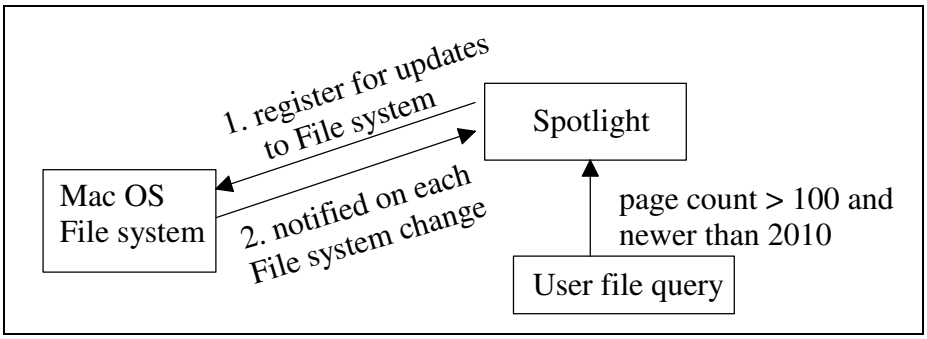
4. Accessing files via directories or indexing are blurring:
- directories - file system provides access by path/file name
- indexes - file/database provide access to entries by portion of the contents (unique - key or non-unique attributes, e.g, all files with > 100 pages) Mac OS X Spotlight and Windows Vista - content-based access to file

Commonality: maps from key to object

Differences: index "query" might not be single object, index query might involve multiple attributes (page limit, newer than), directory: whole file vs. database index given only matching rows of table



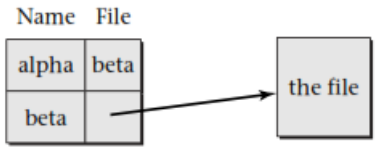
Mac OS X Spotlight is actually middleware that registers with the file system to be notified about changes

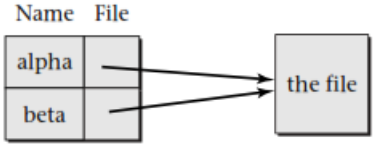


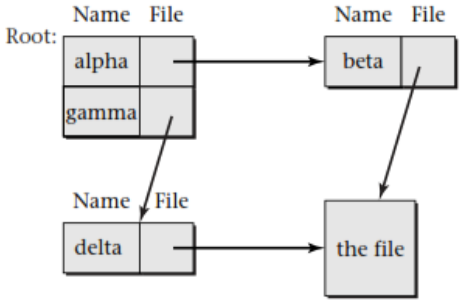
a) If new software installed with new file type (.doc, .pdf, etc.), how are queries like "page count" performed?

5. *File linking* - file can be reached through multiple names. Two types:
- *hard link* - two difference (possibly in different directories) names associated with same file
- *soft/symbolic link* - one name refers to another name (indirect reference to a file)

a) Which of the following figures refer to hard vs. soft links?







b) Which would have the problems of:
- long chains of references
- dangling references
- loops in the references - ELOOP error POSIX error code

c) Which is more flexible have the ability to reference directories or reference files on separate file systems?

6. Directory and Index data structures can be implemented in various ways:
- UNIX directories traditionally used unordered linear list of key/value pairs. What disadvantage does this have?

- hash table: option in Postgre SQL database index, BSD FFS file system (build directory hash table from file list if directory used) What advantages and disadvantages do hash tables have?

- B+ tree - dominant data structure for DB indexes and newer file systems directories
  - MS NTFS and SGI XFS - B+ tree for each directory
  - Mac OS X HFS Plus - single B+ tree for all directories. What keys would be needed in the B+ tree?

7. File system metadata integrity - needs to handle system crash

a) Why would the file system want to cache metadata changes in RAM?

b) What problems occur if caching file system changes in RAM if the system crashes?

8. Kinds of metadata integrity violations:

- irreparable corruption, e.g., two files using the same block
- noncritical reparable corruption - e.g., lost block not used or on free list -- repair when convenient
- critical reparable corruption - block both in use and on free list -- repair before system can return to normal operation

Two strategies to maintain metadata integrity (2 variants each0

I. Each logical change to the metadata state can be accomplished by writing a single block to persistent storage.

- journaling - single block commit record in write-ahead log. What needs to be done on crash recovery?

- shadow paging - create new metadata structure rather than modify old. Single block write to point to new structure instead of old.

II. Each logical change to metadata state can involve multiple block writes, but the order of the updates is carefully controlled so after a crash any inconsistencies will be reparable kind.

- synchronous write - update order controlled by actually writing updates to metadata blocks in persistent storage immediately without buffering in RAM
  What problems?

- soft updates - buffer updates in RAM, but track dependencies among them. Write blocks to persistent storage using dependency information

What problems?