

### Administrivia

- Homework 5 due Thursday.
- Homework 6 on Web tomorrow, due next Tuesday. Short.

Slide 1

### Filesystem Implementation — Recap

- Recall idea of filesystems — directory entry for a file points to something we can use to find file's blocks:
  - First block and size of contiguous sequence.
  - First block of linked list of blocks.
  - Entry in FAT, which points to first block and holds linked lists.
  - I-node, which contains list of blocks.
- Directory entry can also contain file attributes, or they can be stored elsewhere (e.g., in i-node).
- Notice how this is somewhat analogous to memory management — similar tradeoffs.
- Must also manage free space. Issues include ...

Slide 3

### Minute Essay From Last Lecture

- One student thought Homework 4 was much harder than previous assignments. Do you agree? If so, why?  
About evenly split, a few more agreed.

Slide 2

### Blocksize

- "I/O software" can provide a device-independent blocksize (and translate to cylinder/track/sector disk addresses).
- How big should blocks be?
  - What if they're really big?
  - What if they're really small?
  - Usually compromise, also consider page size.

Slide 4

### Managing Free Space — Free List

- One way to track which blocks are free — list of free blocks, kept on disk.
- How this works:
  - Keep one block of this list in memory.
  - Delete entries when files are created/expanded, add entries when files are deleted.
  - If block becomes empty/full, replace it.

Slide 5

### Filesystem Reliability — Backups

- Why do backups? sometimes data is more valuable than physical medium, and might need to
  - Recover from disaster (rare).
  - Recover from stupidity (less rare – hence "recycle bin" idea).
- Many issues involved — which files to back up, how to store backup media, etc., etc. — see textbook.

Slide 7

### Managing Free Space — Bitmap

- Another way to track which blocks are free — "bitmap" with one bit for each block on disk, also kept on disk.
- How this works:
  - Keep one block of map in memory.
  - Modify entries as for free list.
- Usually requires less space.

Slide 6

### Filesystem Reliability — Consistency Checks

- Can easily happen that true state of filesystem is represented by a combination of what's on disk and what's in memory — a problem if shutdown is not orderly.
- Solution is a "fix-up" program (Unix `fsck`, Windows `scandisk`). Kinds of checking we can do:
  - Consistency check: For each block, how many files does it appear in (treating free list as a file)? If other than 1, problem — fix it as best we can.
  - File consistency check: For each file, count number of links to it and compare with number in its i-node. If not equal, change i-node.
  - Etc., etc. — see text.

Slide 8

### Filesystem Performance

- Access to disk data is much slower than access to memory — seek time plus rotational delay plus transfer time.
- So, file systems include various optimizations . . .

Slide 9

### Improving Filesystem Performance — Block Read-Ahead

- Idea — if file is being read sequentially, can read some blocks "ahead". (Of course, doesn't help if file is being read non-sequentially. Decide based on recent access patterns.)

Slide 11

### Improving Filesystem Performance — Caching

- Idea — keep some disk blocks in memory; keep track of which ones are there using hash table (base hash code on device and disk address).
- When cache is full and we must load a new block, which one to replace?  
Could use algorithms based on page replacement algorithms, could even do LRU accurately — though that might be wrong (e.g., want to keep data blocks being filled).
- When should blocks be written out?
  - If block is needed for file system consistency, could write out right away.
  - If block hasn't been written out in a while, also could write out, to avoid data loss in long-running program.
    - \* "Write-through cache" (Windows) — always write out modified blocks right away.
    - \* Periodic "sync" to write out (Unix).

Slide 10

### Improving Filesystem Performance — Reducing Disk Arm Motion

- Group blocks for each file together — easier if bitmap is used to keep track of free space. If not grouped together — "disk fragmentation" may affect performance.
- Place i-nodes so they're fast to get to (and so maybe we can read an i-node and associated file block together).

Slide 12

### Example Filesystem — Unix V7

- Filename restriction — each part of path name at most 14 characters.
- So, directory entry is just 14-byte name and i-node number.
- I-nodes are all stored in a contiguous array at the start of the file system (right after boot block and a "superblock" containing additional parameters).
- What's in each i-node? attributes (permission bits, numeric owner and group ID, timestamps, links count) and list of blocks — last is pointer to more blocks.
- To find a file:
  - Start with root directory — its i-node is in a known place.
  - Scan directory for first part of path, get its i-node, read it, scan for next part of path, etc.
  - Relative path names are handled by including "." and ".." in each directory, so no special code needed.

Slide 13

### Minute Essay

- Anything else you particularly want to know about filesystems?
- Advisory vote:
  - Choice 1: Homework 6 due Tuesday, solutions available Wednesday.
  - Choice 2: Homework 6 due Monday (day before exam), solutions available Monday.

Slide 15

### Next Time

- Journalling filesystems.
- Filesystems in real o/s's.

Slide 14