

Slide 1

### Administrivia

- Reminder: Homework 3 due Monday (extended by request). Small corrections/clarifications about programming problems sent by e-mail earlier today.
- (E-mail about class before holiday sent . . .)

Slide 2

### Minute Essay From Last Lecture — Some Responses

- Different requirements — based on hardware, what the system is used for, etc. Different goals.
- “Better ideas”.
- Lots of design choices, so people have tried out different ones.
- No one filesystem will be good for all uses.
- Everybody likes his/her ideas best (companies too).
- Avoidance of copyright infringement (?).
- Sometimes have to keep old ones around for compatibility.
- (Also review added slide — filesystem quotas.)

### 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 3

### 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.
  - Two approaches: "Write-through cache" (Windows) — always write out modified blocks right away. Periodic "sync" to write out (UNIX).

Slide 4

### 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 5

### 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.
- If i-nodes are being used, place them so they’re fast to get to (and so maybe we can read an i-node and associated file block together).

Slide 6

## Disk Fragmentation

Slide 7

- Idea — if blocks that make up a file are (mostly) contiguous, faster to read them all. If not, “disk fragmentation”.
- How likely is disk fragmentation? Depends on filesystem, strategy for allocating space for files.
- “Defragmenter” utility can be run to correct it. Windows comes with one. Linux doesn't. The claim is that UNIX and Linux filesystems typically don't become fragmented unless the disk is close to full.

## Example Filesystem — Unix V7

Slide 8

- 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.

## Minute Essay

- This wraps up the planned lectures on filesystems. Anything you'd like to hear more about?

Slide 9