CS323 Operating Systems
File System

Yuanyuan Zhou
Lecture 23
3/19/2003

Content of this lecture

- Administrative announcements
- Disk scheduling
- File systems basic concepts
- Summary
- The hardest question:
  - Using swap to implement mutex

Administatives

- Regrading Period
  - MP1: until this Friday, 3/21 5pm
  - Midterm1: until Friday, 3/21 5pm
    - Pick/up your midterm from TA's office
  - Submit written request to TA
  - After this deadline, no regrading request will be granted!!
- 3/31 & 4/2 lectures
  - Given by TA: Jeff

Review

- Power management
  - Power-down after idle for some time
- Disk internal and trends

Disk Performance

- Seek
  - Position heads over cylinder, typically 5.3 – 8 ms
- Rotational delay
  - Wait for a sector to rotate underneath the heads
  - Typically 8.3 – 6.0 ms (7,200 – 10,000RPM) or ½ rotation takes 4.15-3ms
- Transfer bytes
  - Average transfer bandwidth (15-37 Mbytes/sec)
- Performance of transfer 1 Kbytes
  - Seek (5.3 ms) + half rotational delay (3ms) + transfer (0.04 ms)
  - Total time is 8.34ms or 120 Kbytes/sec!
- What block size can get 90% of the disk transfer bandwidth?

Disk Behaviors

- There are more sectors on outer tracks than inner tracks
  - Read outer tracks: 37.4MB/sec
  - Read inner tracks: 22MB/sec
- Seek time and rotational latency dominates the cost of small reads
  - A lot of disk transfer bandwidth are wasted
  - Need algorithms to reduce seek time

Block Size (Kbytes)	% of Disk Transfer Bandwidth
1Kbytes	0.5%
8Kbytes	3.7%
256Kbytes	55%
1Mbytes	83%
2Mbytes	90%

Observations

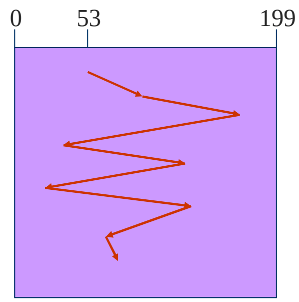
- Getting first byte from disk read is slow
  - high latency
- Peak bandwidth high, but rarely achieved
- Need to mitigate disk performance impact
  - Do extra calculations to speed up disk access
    - Schedule requests to shorten seeks
  - Move some disk data into main memory – file system caching

Disk Scheduling

- Which disk request is serviced first?
  - FCFS
  - Shortest seek time first
  - Elevator (SCAN)
  - C-SCAN (Circular SCAN)
- Look familiar?

FIFO (FCFS) order

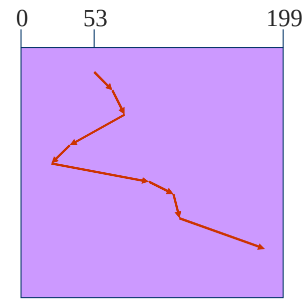
- Method
  - First come first serve
- Pros
  - Fairness among requests
  - In the order applications expect
- Cons
  - Arrival may be on random spots on the disk (long seeks)
  - Wild swing can happen
- Analogy:
  - Can elevator scheduling use FCFS?



98, 183, 37, 122, 14, 124, 65, 67

SSTF (Shortest Seek Time First)

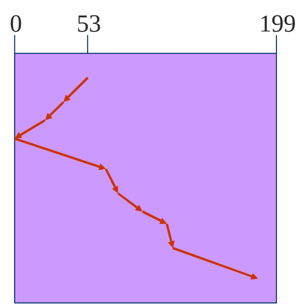
- Method
  - Pick the one closest on disk
  - Rotational delay is in calculation
- Pros
  - Try to minimize seek time
- Cons
  - Starvation
- Question
  - Is SSTF optimal?
  - Can we avoid starvation?
- Analogy: elevator



98, 183, 37, 122, 14, 124, 65, 67
(65, 67, 37, 14, 98, 122, 124, 183)

Elevator (SCAN)

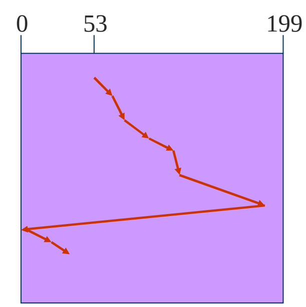
- Method
  - Take the closest request in the direction of travel
  - Real implementations do not go to the end (called LOOK)
- Pros
  - Bounded time for each request
- Cons
  - Request at the other end will take a while



98, 183, 37, 122, 14, 124, 65, 67
(37, 14, 65, 67, 98, 122, 124, 183)

C-SCAN (Circular SCAN)

- Method
  - Like SCAN
  - But, wrap around
  - Real implementation doesn't go to the end (C-LOOK)
- Pros
  - Uniform service time
- Cons
  - Do nothing on the return



98, 183, 37, 122, 14, 124, 65, 67
(65, 67, 98, 122, 124, 183, 14, 37)

History of Disk-related Concerns

- When memory was expensive
  - Do as little bookkeeping as possible
- When disks were expensive
  - Get every last sector of usable space
- When disks became more common
  - Make them much more reliable
- When processor got much faster
  - Make them appear faster

Disk Versus Memory

Memory

- Latency in 10's of processor cycles
- Transfer rate 300+MB/s
- Contiguous allocation gains ~10x

Disk

- Latency in milliseconds
- Transfer rate 5-50MB/s
- Contiguous allocation gains ~1000x

On-Disk Caching

- Method
  - Put RAM on disk controller to cache blocks
    - Seagate ATA disk has .5MB, IBM Ultra160 SCSI has 16MB
    - Some of the RAM space stores "firmware" (an OS)
  - Blocks are replaced usually in LRU order
- Pros
  - Good for reads if you have locality
- Cons
  - Expensive
  - Need to deal with reliable writes

Why Files?

- Physical reality
  - Block oriented
  - Physical sector #s
  - No protection among users of the system
  - Data might be corrupted if machine crashes
- Filesystem model
  - Byte oriented
  - Named files
  - Users protected from each other
  - Robust to machine failures

Group Discussion

- What functions should file systems provide?

File System Requirements

- Users must be able to:
  - create, modify, and delete files at will.
  - read, write, and modify file contents with a minimum of fuss about blocking, buffering, etc.
  - share each other's files with proper authorization
  - transfer information between files.
  - refer to files by symbolic names.
  - retrieve backup copies of files lost through accident or malicious destruction.
  - see a logical view of their files without concern for how they are stored.

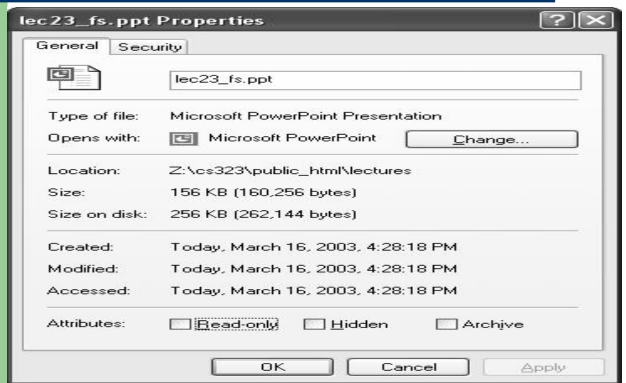
File Concepts

- Files
- Directory structures
- Partitions (possible)
- File Concept: OS abstracts from the physical properties of its storage device to define a logical storage unit, called file. Files are mapped by the OS onto physical devices.

File Attributes

- Name: symbolic file name, the only information kept in human-readable form. Many OS support two part file names (name.extension)
- Type**:** needed for systems that support different types.
  - Regular files - user information, regular files are generally either ASCII or binary files.
  - Directories - system files for maintaining the structure of the file system
  - Character special files - related to input/output and used to model serial I/O devices such as terminals, printers, and networks
  - Block special files - used to model disks
- Location: pointer to a device and to the location of the file on that device.
- Size: current size and maximal possible size
- Protection: Access-control information.
- Time, date, user identification: creation time, last modification, last use.

Example: Windows NTFS File Atributes



File Operations

- **Create a file**
- **Write a file**
- **Read a file**
- **Reposition within a file**
- **Delete a file**
- **Append to a file**
- **Copy a file**

File Structures

- Byte sequence
  - Read or write a number of bytes
  - Unstructured or linear
- Record sequence
  - Fixed or variable length
  - Read or write a number of records
- Tree
  - Records with keys
  - Read, insert, delete a record (typically using B-tree)

File Structures Today

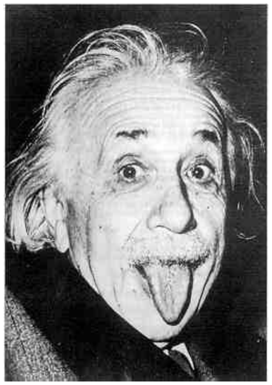
- Stream of bytes
  - Simplest to implement in kernel
  - Easy to manipulate in other forms
  - Little performance loss
- More complicated structures
  - Hardware assist fell out of favor
  - Special-purpose hardware slower, costly

File Types

- ASCII – plain text
- A Unix executable file
  - header: magic number, sizes, entry point, flags
  - Text (code)
  - Data
  - relocation bits
  - symbol table
- Devices
- Everything else in the system

So What Makes Filesystems Hard?

- Files grow and shrink in pieces
- Little *a priori* knowledge
- 6 orders of magnitude in file sizes
- Overcoming disk performance behavior
- Desire for efficiency
- Coping with failure



Reminder