Outline

- Announcements
  - Lab 6 is due April 30th: Please sign up for demos
  - Final exam on May 4th, 2:00 – 3:50pm
  - Questions?

- Secondary storage structures
  - Disk structure
  - Disk scheduling
  - Reliability concerns

- Protection
  - Goals of protection
  - Domain of protection
  - Access matrices

[Silberschatz/Galvin: Chapters 13, 19]

Secondary Storage

- Non-volatile repository of data
- Magnetic tape
- Magnetic disk
  - Hard disks
  - Floppy drives
- Optical technology
  - CD-ROM, ..., DVD-ROM
    - Differences in compression, number of layers (semi-transparent gold over reflective silver) bandwidth of laser, etc.
    - Holographic storage: microscopic mirrors and lasers

Structure of Secondary Storage

- Main form of secondary storage is the disk
  - Disk made up of multiple magnetic platters
    - Each platter has up to two surfaces
  - Each surface has a number of tracks
    - Vertical arrangement of tracks into cylinders
    - Each track contains multiple sectors

- Mechanics
  - Seek: Position read/write head
  - Rotation: Wait for appropriate sector
  - Transfer data

- Examples
  - 1.44 MB floppy: 2 surfaces, 80 tracks, 36 sectors
    - Seek = 94 ms, Rotation = 300 rpm, Transfer = 500 Kb/s
  - 18.3 GB drive: 29 surfaces, 274 tracks, 136 sectors
    - Seek = 8-10 ms, Rotation = 10020 rpm, Transfer = 30-40 MB/s
Using Multiple Disks

• An alternative to using disks independently
  – the secondary storage device consists of multiple physical disks working cooperatively
  – each component disk has a separate physical channel

• Possible organization
  – contiguous "sub-blocks" are stored on different disks: striping
  – to access a block, all of its sub-blocks are accessed in parallel
  – improves transfer time, no change in seek time or rotational latency

Optimizing Disk Accesses

• Typically, the seek time is the dominant cost
  – despite improvements in areal density: 60% a year since 1988
  – storage and access methods seek to optimize it

• Single request
  – optimize layout and access of data
    • e.g., store and access information in groups of physical sectors
    • can lead to internal fragmentation on disks
  – we talked about this in Lectures 16 and 17

• Multiple requests
  – can schedule these requests to optimize disk performance
  – requests can be buffered in a queue
    • subsequently, as the current access completes, the next access can be started
  – various "priorities" are possible

Disk Scheduling: First Come First Served

• Easy to implement
  – the pending requests are maintained in a queue
    • the next request to be serviced is the one at the head of the queue
    • new requests are added at the tail

• Disadvantages
  – ignores seek times

Disk Scheduling: Shortest Seek Time First

• The next request is to/from the closest track
  – rationale: seek time is smaller for nearer tracks
    • not proportional to the track difference
  – results in better average disk throughput, lower service delays
  – analogous to shortest job first scheduling: can lead to a form of starvation

Request sequence:
98, 183, 37, 122, 14, 124, 65, 67
Head at: cylinder 53
Head movement of 640 cylinders

Request sequence:
98, 183, 37, 122, 14, 124, 65, 67
Head at: cylinder 53
Head movement of 236 cylinders
A SSTF Drawback

- Given a sequence of requests, is SSTF scheduling optimal?
- No, because
  - while the next seek is the nearest, cumulative seek durations need not be globally minimum
  - the approach is what is called a "greedy scheme"
    - generally quite good but rarely globally optimum
  - example
    - 5,6,3,9 distance 1+3+6=10
    - 5,3,6,9 distance 2+3+3=8

Disk Scheduling: Scan Scheduling

- The “elevator” algorithm
  - Starting from one end, process requests in order of increasing track number
  - when no more requests are available, reverse direction
- Performance improvement over SSTF: no starvation
  - favors middle tracks (encountered twice) over end tracks

Request sequence: 98, 183, 37, 122, 14, 124, 65, 67
Head at: cylinder 53
Head movement of 208 cylinders

Disk Scheduling: Circular Scan

- SCAN avoids starvation, but provides non-uniform wait times
  - at each direction reversal, looks at the tracks recently visited
- C-Scan scheduling (circular scan) scans in only one direction
  - at end of scan, return to other end immediately

Request sequence: 98, 183, 37, 122, 14, 124, 65, 67
Head at: cylinder 53

Disk Scheduling: LOOK and C-LOOK

- Variants of SCAN and C-SCAN where the arm goes only as far as the final request in each direction
  - Saves on redundant movement
Choosing between Disk Scheduling Schemes

Issues
- The usual tradeoff between
  - the amount of work done by the scheme
  - and the quality of the scheduling
- File allocation strategy
  - e.g. FCFS is OK with contiguous, but not linked allocation
- Relative locations of files and directories
  - some schemes attempt to put these close together

- Disk scheduling typically implemented in the disk controller
  - very sensitive to the implementation (of the disk)
  - different tradeoffs between seek times and rotational latency
  - SSTF and LOOK are typical choices

Reliability Concerns
- Recovery from disk failures involving large amounts of information
  - backups are expensive but the obvious solution
- A better solution are RAIDs
  - (Redundant Array of Inexpensive Disks)
- The elementary approach (called mirroring): RAID Level 1
  - maintains a copy of each disk
  - wasteful and expensive
- A somewhat less expensive approach RAID Level 3
  - given k disks in the array
  - use the first (k-1) for storage: data striped across disks at byte level
  - use the last disk for parity
  - if any one disk fails, its contents can be recomputed

Block-Interleaved Parity (RAID Level 4)
- Parity for a set of bits is 1 if number of 1 bits is even
- Basic scheme
  - for n blocks b[1] ... b[n]
  - store parity block (per-bit) on parity disk

Rotating Block-Interleaved Parity (RAID Level 5)
- RAID Level 4 takes a performance hit on small, random requests
  - All requests access the parity drive, which becomes a bottleneck
- Solution: Distribute the parity information
  - No loss in reliability
  - Good performance properties
Reliability Concerns - 2

- Recovery from disk failures where only a portion of the information is lost
  - data-transfers may result in
    - success
    - partial failure: the destination sector contains incorrect information
    - total failure: the destination is unchanged

- The object of stable storage is to recover from partial failures
  - write multiple copies of each block
    - typically two
  - a write is complete when all copies are written
  - to recover from a failure
    - after writing, check all blocks
    - if a detectable error exists, then use the other value in all copies
    - if not, the "value" of the block is that of the last (second) copy

Protection

- The overall need
  - initially caused by the explosion in multiprogramming
  - recently by the explosion in networking
  - needed multiple co-existent processes not to encroach/interfere
    - accidentally or maliciously ...
    - into unwanted regions

Solution:
- Define specific domains that specify the privileges of a user/process/procedure to access system resources
  - For each resource, what are the operations that the process can do
    - Resources: files, memory segments/pages, printer, ...
- Provide mechanisms for enforcing these privileges
  - Hardware: Protection faults, privileged instructions, exceptions
  - Software: System calls

Domain of Protection

- Components
  - Logical processes
  - Classes of logical objects that processes can access in various ways
    - e.g. files, directories, devices
  - Each object class has a set of rights
    - e.g. read, write, delete

- A process at any given time
  - is in a particular domain
  - inherits all of its privileges
    - Domains are the granularity at which privileges are specified
What is a Domain?

- Formally, a domain is a set of access rights
  - an access right is a pair: <object, rights>
    - Example could be <payroll-file, read-only>

- A process can only be in one particular domain at any given time
  - Association can be …
  - … static: need OS mechanisms to modify access rights
  - or dynamic: need OS mechanisms to switch domains

- Domains can
  - contain each other
  - be non-overlapping
  - change over time, i.e. can be modified

Examples

- Dual-mode model of operating system execution
  - Two domains: system (monitor) and user
  - System domain has complete rights
  - User domain cannot execute privileged instructions

- Unix
  - Three kinds of domains: users, groups, “other”
  - Access rights are specified for each such domain
    - E.g., chmod operations for setting file access rights
  - Switching the domain corresponds to changing the user identification or group identification temporarily

Access Matrices

- Used to describe domains
  - Each row is a domain
  - Each column is an object
  - Entry (i, j) indicates the privileges associated with domain i to access object j

- Example

<table>
<thead>
<tr>
<th></th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>printer</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>r</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D2</td>
<td></td>
<td></td>
<td>pr</td>
<td></td>
</tr>
<tr>
<td>D3</td>
<td></td>
<td>r</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>D4</td>
<td></td>
<td>r</td>
<td>rw</td>
<td></td>
</tr>
</tbody>
</table>

Mechanism vs. Policy

- The OS supports means of specifying and enforcing preferred access patterns via the access matrix and hence supports the mechanism
- The actual decisions are entries in the matrix and denote policy

- Typically, the owner of an object determines the corresponding privileges and hence entries in the appropriate column
Next Lecture

- Protection (cont’d)
  - Implementing access matrices
    - Access control lists
    - Capabilities
  - Language-based protection

- Security
  - the security problem
  - authentication
  - one-time passwords
  - encryption schemes
  - program and system threats

Silberschatz/Galvin: Chapters 19 and 20