Outline

- Announcements
  - Lab 5 needs to be demo-ed this week for you to receive any credit
  - Same rule applies for all previous labs
  - Questions?
- I/O systems (cont’d)
- Secondary storage structure
  - disk structure
  - disk scheduling algorithms
  - RAID structure

// Silberschatz/Galvin/Gagne: Chapter 13, Sections 14.1—14.5

(Review) Application I/O Interface

- **Goal:** Standard, uniform treatment of I/O devices
- **I/O system calls** encapsulate device behaviors in generic classes
- **Device-driver** layer hides differences among I/O controllers
  - Devices differ among several dimensions
    - character-stream (keyboards, mice) or block (disks) or network devices
    - sequential or random-access
    - synchronous or asynchronous
    - sharable or dedicated
    - read-write, read-only, or write-only
  - Device driver exports a standard interface (mapping to system calls)
    - O/Ses typically provide a back-door to directly access device driver
      - E.g., UNIX ioctl system call
  - Decoupling benefits both O/S developers and hardware manufacturers

I/O System Call Interface

**Block and character devices**

- Block devices include disk drives
  - Commands include `read`, `write`, `seek`
  - Raw I/O or file-system access
  - Memory-mapped file access possible
- Character devices include keyboards, mice, serial ports
  - Commands include `get`, `put`
  - Libraries layered on top allow line editing

**Network devices**

- Different addressing, performance characteristics from block devices
- Unix and Windows/NT include `socket` interface
  - Separates network protocol from network operation
  - Includes `select` functionality
- Approaches vary widely (pipes, FIFOs, streams, queues, mailboxes)
I/O System Call Interface (cont’d)

Clocks and timers
- Provide current time, elapsed time, timer
  - programmable interval time used for timings, periodic interrupts
  - used by the kernel I/O subsystem

- Interface Issue: Blocking vs. Nonblocking I/O
  - Blocking - process suspended until I/O completed
    - Easy to use and understand
    - Insufficient for some needs
  - Nonblocking - I/O call returns as much as available
    - Returns quickly with count of bytes read or written
    - User interface, data copy (buffered I/O)
  - Asynchronous - process runs while I/O executes
    - Difficult to use
    - I/O subsystem signals process when I/O completed
    - Can be implemented via multi-threading

Kernel I/O Subsystem

Provides a common set of services and maintains kernel data structures

1. Scheduling: I/O request ordering via per-device queue
   - For performance and fairness
   - e.g., disk request scheduling affects seek overheads

2. Buffering: Store data in memory while transferring between devices
   - to cope with device speed/device transfer size mismatch
     - Double buffering permits overlap
     - to maintain “copy semantics”
       - E.g., buffer containing write data must be reusable after call returns
       - How can this be supported if the write call is asynchronous?
       - Virtual memory mapping and protection can support efficient copying
         - Copy-on-write
         - How does this work?

Kernel I/O Subsystem (cont’d)

3. Caching: Fast memory holding copy of data
   - key to performance
   - E.g., file caches
   - How does caching differ from buffering?

4. Spooling: Merges requests for a device
   - if device can serve only one request at a time (e.g., printing)
   - System call interface permits multiple applications to print concurrently
     - Request data is put into a common directory
     - Spooler issues them one at a time to the device

5. Device reservation: Provides exclusive access to a device

6. Error handling
   - recovery from transient errors: e.g. reissuing a disk read request
   - report rest to application program

Outline

- Announcements
  - Lab 5 needs to be demo-ed this week for you to receive any credit
  - Same rule applies for all previous labs
  - Questions?

- I/O systems (cont’d)
- Secondary storage structure
  - disk structure
  - disk scheduling algorithms
  - RAID structure

[Silberschatz/Galvin/Gagne: Chapter 13, Sections 14.1—14.5]
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
  - each disk has multiple magnetic platters
  - each platter has up to two surfaces
  - each surface has 20 to 1500 tracks per surface
    - Vertical arrangement of tracks into cylinders
  - each track contains multiple sectors
    - example: 36.4 Gigabyte Seagate Cheetah Disk
      - 24 tracks/cylinder, 9801 cylinders
      - 300 sectors/track, 512 bytes/sector
    - example: 1.44 Megabyte floppy
      - 2 tracks/cylinder, 80 cylinders
      - 36 sectors/track, 512 bytes/sector

Mechanics of Accessing Data

- The read/write head is positioned
  - on the appropriate track
  - the time to reach this state is called the seek time
  - seek times on the order of 5–10 milliseconds
- The controller waits for the appropriate sector
  - this additional time is the rotational latency
  - typical rotational speeds of 5000–10000 rpm
- The controller moves information
  - to/from its buffer
  - this is the transfer time
  - typical transfer bandwidths of 20–40 MB/s
- Capacity determined by areal density
  - Commercially available: 46 GB/inch² (e.g., IBM/Hitachi Deskstar 180GXP)
  - Research labs (IBM): approaching physical limit: ~60 GB/inch²

Sector Layout in Modern Disks

- Zoned Bit Recording
  - Outer tracks have more sectors, larger transfer rates
  - IBM/Hitachi Deskstar 180 GXP series
    (30 – 180 GB capacity)
Sector Layout for Deskstar 180 GXP

<table>
<thead>
<tr>
<th>Zone</th>
<th>Physical Cylinders</th>
<th>BR/TH</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0-1351</td>
<td>0-082</td>
</tr>
<tr>
<td>1</td>
<td>1352-2532</td>
<td>1-082</td>
</tr>
<tr>
<td>2</td>
<td>2533-3714</td>
<td>1-080</td>
</tr>
<tr>
<td>3</td>
<td>3715-4898</td>
<td>1-079</td>
</tr>
<tr>
<td>4</td>
<td>4899-6080</td>
<td>1-078</td>
</tr>
<tr>
<td>5</td>
<td>6081-18512</td>
<td>0-067</td>
</tr>
<tr>
<td>6</td>
<td>18513-21887</td>
<td>0-062</td>
</tr>
<tr>
<td>7</td>
<td>21888-25014</td>
<td>0-056</td>
</tr>
<tr>
<td>8</td>
<td>25015-27236</td>
<td>0-054</td>
</tr>
<tr>
<td>9</td>
<td>27237-329206</td>
<td>0-024</td>
</tr>
<tr>
<td>10</td>
<td>32921-39242</td>
<td>0-102</td>
</tr>
<tr>
<td>11</td>
<td>39243-43409</td>
<td>0-032</td>
</tr>
<tr>
<td>12</td>
<td>43410-48527</td>
<td>0-031</td>
</tr>
<tr>
<td>13</td>
<td>48528-50577</td>
<td>0-030</td>
</tr>
<tr>
<td>14</td>
<td>50578-56742</td>
<td>0-029</td>
</tr>
<tr>
<td>15</td>
<td>56743-61910</td>
<td>0-028</td>
</tr>
<tr>
<td>16</td>
<td>61911-66976</td>
<td>0-027</td>
</tr>
<tr>
<td>17</td>
<td>66977-70933</td>
<td>0-026</td>
</tr>
<tr>
<td>18</td>
<td>70934-76999</td>
<td>0-025</td>
</tr>
<tr>
<td>19</td>
<td>76999-83061</td>
<td>0-024</td>
</tr>
<tr>
<td>20</td>
<td>83062-88096</td>
<td>0-023</td>
</tr>
<tr>
<td>21</td>
<td>88097-93137</td>
<td>0-022</td>
</tr>
<tr>
<td>22</td>
<td>93138-98172</td>
<td>0-021</td>
</tr>
<tr>
<td>23</td>
<td>98173-103209</td>
<td>0-020</td>
</tr>
<tr>
<td>24</td>
<td>103210-108246</td>
<td>0-019</td>
</tr>
<tr>
<td>25</td>
<td>108247-113280</td>
<td>0-018</td>
</tr>
<tr>
<td>26</td>
<td>113281-118315</td>
<td>0-017</td>
</tr>
</tbody>
</table>

Optimizing Disk Accesses

- Typically, **seek time** is the dominant cost
  - despite improvements in areal density
    - 60% a year since 1988, 100% a year more recently
  - 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 the last few lectures
- Multiple requests
  - can schedule these requests to optimize disk performance
    - requests must be **concurrent** (either from multiple processes, or multiple non-blocking requests from a single process)
    - requests can be buffered in a queue
    - 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

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

Using Multiple Disks

- Store data across multiple disks, each with its own physical channel
  - mirroring improves reliability
  - striping, where contiguous "sub-blocks" are stored on different disks, improves transfer time
    - To access a block, all of its sub-blocks are accessed in parallel
    - Does not change seek time or latency

RAID: Redundant Array of Inexpensive Disks
- The elementary approach (only 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