Reminders

• Assignment 3 due tomorrow night
• Project 3 due tomorrow night
• Midterm take-up Monday afternoon
• Final: next Thursday at 8 AM

• Grading return objectives:
  ‣ HW2 and quizzes: by Monday
  ‣ HW3: before the final
  ‣ Project2, Project3: end of next week
Outline

• Disk structure: physical and logical

• Disk addressing

• Disk scheduling

• Management
Need for Storage

• Memory is:
  ‣ volatile: persistence is required
  ‣ insufficient: large capacity is required
  ‣ not portable: how can we take information with us?

• Long-lasting backup data is needed:
  ‣ scientific applications
  ‣ industry and finance
Mass Storage Application

CERN Particle Collider

- 50,000 data channels
- 200 GB buffering
- Detector proper 40MHz collisions
- Event filtering (1 CPU/event)
- Data storage
- 1MB/event
- ~1TB/s
- ~500Gb/s
- ~0.5GB/s
- ~5PB/year

Oregon Systems Infrastructure Research and Information Security (OSIRIS) Lab
Past & Present in

1956: IBM 305 RAMAC - 5 MB capacity (50 disks, each 24” in diameter)

2008: Seagate Savvio 15K - 73.4 GB capacity, 2.5” diameter
- can read/write complete works of Shakespeare 15 times per second
Storage Hierarchy

expensive and fast

cheap and slow

tertiary storage

secondary storage

main memory

L2 cache

L1 cache

registers
Secondary Storage

• Generally, magnetic disks provide the bulk of secondary storage in systems
  ‣ future alternative: solid-state drives?
    • e.g. MacBook Air
  ‣ MEMS and NEMS (nanotech)
  ‣ holographic storage
    • data read from intersecting laser beams

[Image of holographic storage system]

www.inphase-technologies.com
Inside a Hard Disk

Aluminum (sometimes glass) platters
Deep Inside a Hard Disk

- Bit-cell composed of about 50–100 magnetic grains
- 0 has uniform polarity, 1 has a boundary between magnetizations
- Magnetized in direction of disk head (longitudinal) or perpendicular (more complex, but more density)
- In development: HAMR
  - Heat-assisted (with lasers)
  - Potentially 50 Tb/in²
Disk Operation

• Platters start moving from rest (*spinup time*)
  ‣ lots of mass to start moving

• Heads find the right track (*seek time*)
  ‣ arm powered by actuator motor, accelerates and coasts, slows down and settles on correct track (servo-guided)

• Disk rotates until correct sector found (*rotational latency*)
  ‣ contingent on platter diameter and RPM (Savvio 15K rotates 300 times/second)

• Have to stop the platters (*spindown time*)
Addressing Disks

• Old days: CHS (cylinder-head-sector)
  ‣ supply physical characteristics of the disk to the operating system
  ‣ it specifies exactly where on the physical disk to read and write data

• Nowadays: cylinders not uniform
  ‣ can store more data on outer tracks than inner tracks (zoned bit recording)
  • why?
    ‣ function of constant angular velocity (CAV) vs constant linear velocity (CLV) found in CD-ROM
Logical Block Addressing (LBA)

• OS sees drive as an array of blocks
  ‣ first block LBA = 0, next block LBA = 1 etc.

• disk firmware takes care of managing the physical location of data

• Block: smallest unit of data accessible through the OS
  ‣ can be the size of a sector (512 bytes) up to the size of a page (often 4 KB): defined by kernel
Disk Scheduling

• Why does the OS need to schedule?
  ‣ Improves access time (seek time & rotational latency)
  ‣ even with LBA, assumption is that blocks are written in essentially contiguous order
  ‣ maximizes bandwidth
    • transferred bytes / service + transfer time
• Consider the following request queue
  ‣ min cylinder = 0, max cylinder = 199
    ‣ requests at the following cylinders:
      ‣ 98, 183, 37, 122, 14, 124, 65, 67
    ‣ drive head is at cylinder 53
First-come First-served (FCFS)

- Service the requests in order of arrival
- Head movement of 640 cylinders

queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
Shortest Seek Time First (SSTF)

- Min. seek time from head position (like SJF)
- Head movement of 236 cylinders
SCAN (Elevator) Algorithm

- Arm moves from one end of disk to the other then reverses (like an elevator)
- Head movement of 208 cylinders
C-SCAN Algorithm

- More uniform wait time than SCAN
- Head services requests in one direction then returns to beginning of disk (like circular list)
C-LOOK Algorithm

• Like C-SCAN but only seeks to farthest request in queue

• Returns to lowest request (not start of disk)
Choosing a Disk Scheduling Algo.

- SSTF: increased performance over FCFS
- SCAN, C-SCAN: good for heavy loads
  - less chance of starvation
- C-LOOK: good overall
- File allocation plays a role
  - contiguous allocation limits head movement
- Note: only considering seek time
  - rotational latency also important but hard for OS to know (doesn’t have physical drive characteristics)
  - drive controllers implement some queueing and request coalescing
Drive Controller Scheduling?

• Why not have the drive controller in the disk perform all of the disk scheduling?
• Would be more efficient, but...
• OS knows about constraints that the disk doesn’t
  – demand paging > application I/O
  – write > read if cache is almost full
  – guaranteeing write ordering (e.g. journaling, data flushing)
Linux I/O Schedulers

- **Linus Elevator** (default in 2.4 kernel)
  - merges adjacent requests and sorts request queue
  - can lead to starvation in some cases though: big push to change for 2.6 kernel

- **Deadline I/O Scheduler**
  - merges & sorts request + expiration timer
  - multiple queues to minimize seeks while ensuring request don’t starve

- **Anticipatory I/O Scheduler**
  - waits a few ms after a read request to see if another one is made (high probability); acts like deadline scheduler otherwise
  - loses time if wrong but big win if right
• Complete Fair Queueing (CFQ) I/O Scheduler
  – different than the others: assigns queues based on originating process
  – queues are serviced round-robin, usually picking 4 requests from each queue at a time
  – good for multimedia (e.g., ensuring audio buffers are full)

• When to use which?
  – Linus Elevator: obsolete
  – Deadline: good for lots of seeks, critical workloads
  – Anticipatory: good for servers
  – CFQ: desktops
Disk Management

- Low-level formatting
- Logical formatting
- Booting
- Bad block recovery
- Swap space
Low-Level (Physical) Formatting

• divide disk into sectors for disk controller to read and write
  ‣ sector numbers, error-correcting codes (ECC), other identifying information (e.g., servo control data) written to each sector

• usually only done at factory
  ‣ can restore factory configuration (reinitialize)
High-Level (Logical) Formatting

• Before formatting, OS needs to partition the disk into 1 or more cylinder groups
  ‣ why more than 1? root vs swap partitions, dual boot, etc.

• write a file system onto the disk
  ‣ structures such as file allocation table (FAT - DOS) or inodes (UNIX)

• write the boot block (boot sector)
Boot Process

• Bootstrapping starts from a process in ROM

• Boot loader reads a bootstrap program from the bootblock
  – on PCs: Master boot record (MBR): first sector on disk (446 bytes, then 64 byte partition table)

• Second-stage boot loader: program whose location is pointed to from MBR
  – NTLDR on Windows, LILO/GRUB on Linux
    • choose the partition to boot from to start to OS
Bad Block Recovery

• Most disks have some bad blocks even from the factory

• ECC used (Reed-Solomon encoding on modern disks) to try and recover

• Sector Sparing: drive marks bad block and maps to a spare block the OS doesn’t see

• Sector Slipping: drive remaps blocks in order on disk, skipping over bad one
  – Disk does lots of background tasks
    • Still, Avoid head crashes
Swap-Space Management

• Swap space: used for virtual memory (extension of main memory)

• Often given its own disk partition
  – Can hold process images or memory pages

• Linux and Solaris: page slots within swap files or partitions
  – Only allocate swap page slot when page forced out of memory
  – Swap map indicates how many processes using page
Linux Swap Structures

swap partition or swap file

swap map

1 0 3 0 1

swap area

page slot
Attaching Disks to Networks

• **NAS**: network attached storage - RPCs between host and storage
  – e.g., NFS (what we use), iSCSI

• **SAN**: storage area network
  – multiple connected storage arrays, servers connect directly to SAN

• **Becoming more like each other**
  – e.g., Open Storage Networking proposal (from NetApp) combines elements of each
SCSI vs IDE/ATA

• Originally speed but with serial ATA (SATA) interface speeds have caught up

• SCSI supports more drives on a bus but SATA can be beneficial for small numbers

• Why pay more for SCSI? Disks manufactured differently
  – assumed to be server (enterprise) vs personal
    • often faster (e.g., 15K disks usually only SCSI)
    • SCSI drives better constructed (O-ring sealing, air flow, more rigidity); stronger actuator motors; more reliable
    • ATA cheap though: 1 TB SATA < 73 GB SCSI
Summary

• Storage is critical and getting more so
• Physical characteristics: cylinders (tracks), heads, sectors
• Seek, rotation time
• Scheduling algorithms affect system performance
• Storage management: boot process, swap space
• On your own: look over NAS and SAN figs
  – Recommended: RAID (0, 1, 5 most common)
Final Exam

• Thursday, June 13 at 8 AM

• 2 hours

• Structure: similar to previous exam in terms of question layout, may be similar length or possibly a bit longer

• Questions:
  ‣ some technical details
  ‣ some conceptual questions

• Material: slides, text, class discussion, homeworks
What we’ve covered

• Everything from the first part of the course is fair game
  ‣ but probably de-emphasized a bit - bit twiddling probably not required but have a strong idea of the concepts

• Chapter 6 -- Synchronization
• Chapter 7 -- Deadlocks & Chapter 18 -- Distributed Coordination
• Chapter 8 -- Main Memory (Physical)
• Chapter 9 -- Virtual Memory
• Chapter 10 -- File System Interface & Chapter 14 -- Protection
• Chapter 11 -- File System Implementation
• Chapter 12 -- Storage
• Chapter 21 - Linux
Synchronization

• Problems
• Synchronization Requirements
• Disabling Interrupts
• Busy-wait/Spinlock solutions
  ‣ Related to properties
• Hardware-enabled Solutions
• OS-supported
Requirements for Solution

1. Mutual Exclusion - If process Pi is executing in its critical section, then no other processes can be executing in their critical sections.

2. Progress - If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely.

3. Bounded Waiting - A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.
   - Assume that each process executes at a nonzero speed.
   - No assumption concerning relative speed of the N processes.
Synchronization

• Hardware Enabled Solutions

• OS-supported Solutions
  ‣ Mutex
  ‣ Semaphores
  ‣ Condition Variables

• Apply these to code

• Classic Synchronization Problems
You are given a data-type Semaphore_t.

On a variable of this type, you are allowed

- P(Semaphore_t) -- wait
- V(Semaphore_t) – signal

Intuitive Functionality:

- Logically one could visualize the semaphore as having a counter initially set to 0.
- When you do a P(), you decrement the count, and need to block if the count becomes negative.
- When you do a V(), you increment the count and you wake up 1 process from its blocked queue if not null.
Deadlocks

• Necessary Conditions
• Safe States
• Resource Allocation Graph
• Deadlock Prevention
  ‣ Safe States
• Deadlock Detection
  ‣ Detection Algorithm
  ‣ Recovery
Necessary Conditions for a Deadlock

• Mutual exclusion: The requesting process is delayed until the resource held by another is released.

• Hold and wait: A process must be holding at least 1 resource and must be waiting for 1 or more resources held by others.

• No preemption: Resources cannot be preempted from one and given to another.

• Circular wait: A set (P0, P1, … Pn) of waiting processes must exist such that P0 is waiting for a resource held by P1, P1 is waiting for …. by P2, … Pn is waiting for … held by P0.
5 processes, 3 resource types A (10 instances), B (5 instances), C (7 instances)

<table>
<thead>
<tr>
<th></th>
<th>MaxNeeds</th>
<th>Allocated</th>
<th>StillNeeds</th>
<th>Free</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td>B</td>
<td>C</td>
<td>A</td>
</tr>
<tr>
<td>P0</td>
<td>7</td>
<td>5</td>
<td>3</td>
<td>P0</td>
</tr>
<tr>
<td>P1</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>P1</td>
</tr>
<tr>
<td>P2</td>
<td>9</td>
<td>0</td>
<td>2</td>
<td>P2</td>
</tr>
<tr>
<td>P3</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>P3</td>
</tr>
<tr>
<td>P4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>P4</td>
</tr>
</tbody>
</table>

This state is safe, because there is a reduction sequence \(<P1, P3, P4, P2, P0>\) that can satisfy all the requests.

Exercise: Formally go through each of the steps that update these matrices for the reduction sequence.
Deadlock Detection Example

5 processes, 3 resource types A (7 instances), B (2 instances), C (6 instances)

<table>
<thead>
<tr>
<th>Allocated</th>
<th>Request</th>
<th>Free</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A  B  C</td>
<td>A  B  C</td>
<td>A  B  C</td>
</tr>
<tr>
<td>P0 0 1 0</td>
<td>P0 0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>P1 2 0 0</td>
<td>P1 2 0 2</td>
<td></td>
</tr>
<tr>
<td>P2 3 0 3</td>
<td>P2 0 0 0</td>
<td></td>
</tr>
<tr>
<td>P3 2 1 1</td>
<td>P3 1 0 0</td>
<td></td>
</tr>
<tr>
<td>P4 0 0 2</td>
<td>P4 0 0 2</td>
<td></td>
</tr>
</tbody>
</table>

This state is NOT deadlocked.

By applying algorithm, the sequence <P0, P2, P3, P1, P4> will result in Done[i] being TRUE for all processes.
Distributed Coordination

• Event ordering: happened-before relationship
• If events not related by happened-before then they can execute concurrently
• Lamport clock: counter incremented between any two successive events executed within a process
• Distributed Mutex: If $P_i$ is executing in its critical section, then no other process $P_j$ is executing in its critical section.
• Two-phase commit protocol
• Generating unique timestamps: GUIDs
Main Memory

• Swapping

• Allocation
  ‣ Contiguous, Non-contiguous (paging)
  ‣ Algorithms

• Fragmentation
  ‣ Internal, External

• Page-tables, TLBs
  ‣ virtual-physical translation
  ‣ Page table structure, entries
Memory Allocation

Queue of waiting requests/jobs

Question: How do we perform this allocation?
Demand Paging

• Programs are provided with a virtual address space (say 1 MB).

• Role of the OS to fetch data from either physical memory or disk.
  ‣ Done by a mechanism called (demand) paging.

• Divide the virtual address space into units called “virtual pages” each of which is of a fixed size (usually 4K or 8K).
  ‣ For example, 1M virtual address space has 256 4K pages.

• Divide the physical address space into “physical pages” or “frames”.

Page Tables

Virtual Address

Virtual Page # | Offset in Page
---|---
VP # | PP # | Present
vP₁ | pP₁ |
... |
vPₙ | pPₙ |

Physical Address

Physical Page # | Offset in Page
Virtual Memory

• Page Fault Handling
  ‣ Performance Estimations

• Memory Initialization

• Page Replacement
  ‣ Algorithms
  ‣ Belady’s Anomaly

• Uses of Virtual Memory
  ‣ COW, Shared Pages, Memory-mapped Files

• Thrashing
Page Fault

- If there is a reference to a page, first reference to that page will trap to operating system:
  - page fault

- Operating system looks at another table to decide:
  - Invalid reference -- abort
  - Just not in memory

- Get empty frame
- Swap page into frame
- Reset tables
- Set validation bit = v
- Restart the instruction that caused the page fault
File Systems

• File System Concepts
  ‣ Files, Directories, File Systems
  ‣ Operations and Usage
  ‣ Remote File Systems

• File System Implementation
  ‣ What’s on the disk? How’s it formatted?
  ‣ What’s in memory? How’s it represented?

• File System Usage
  ‣ Get a file
  ‣ Caching
  ‣ Free Space
  ‣ Recovery
File System Mounting
i-node

![Diagram of i-node structure]

- Filename
- Time
- Perm.

Disk Block

Data

Disk Block

Data

Disk Block

Data

Disk Block

Data

Disk Block

Data
Access Control: Mode Bits

• Three classes of users: public, group, owner

• Three types of access permissions:
  ‣ read, write, execute

• Example:

<table>
<thead>
<tr>
<th></th>
<th>Owner</th>
<th>Group</th>
<th>Public</th>
</tr>
</thead>
<tbody>
<tr>
<td>rwx</td>
<td>111</td>
<td>101</td>
<td>101</td>
</tr>
<tr>
<td>Octal</td>
<td>7</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

What if no exec access and only owner can read/write?
Access Control/Authorization

- An access control system determines what rights a particular entity has for a set of objects.
- It answers the question:
  - E.g., do you have the right to read /etc/passwd?
  - Does Alice have the right to view the CIS website?
  - Do students have the right to share project data?
  - Does Prof. Butler have the right to change your grades?

- An Access Control Policy answers these questions.
Mass Storage and I/O

- Disk scheduling algorithms
- Access and transfer time (and their components)
- Real schedulers and their differences
Looking forward...

• What is the future of operating system design and research?
  ‣ Mobile phones (Android is a big research platform right now)
  ‣ Clouds (Amazon AWS and other services)
  ‣ Both cases: distributed operation is becoming ever more critical
    • IPC within system and to other components that comprise logical systems
    • ubiquitous computing and prevalent network connectivity

• SECURITY
Planning your UO career

• If you are interested in what you’ve learned in this class and want to consider learning more about systems concepts, you may also think about the following courses:
  ‣ CIS 432: Networking
  ‣ CIS 433: Computer and Network Security

• Also: seminars and reading groups, research

• Think about your future and what you want
  ‣ Take advantage of resources you have at your disposal while you’re a student
Thanks to...

• Colleagues here at UO, at the Pennsylvania State University, the University of Pennsylvania, Swarthmore College, Georgetown University, and Columbia University
  ‣ provided basis for many course materials and course ideas
  ‣ special thanks: Trent Jaeger and Adam Aviv

• Dave Tian: GTF for this course

• Peter McKay, Sean Haverty, Kevin Garsjo, Joe Pletcher: course oracles

• You: Hope you learned something from it all

• Good luck on the final!