CS 571

Operating Systems

Dr. Harry J Foxwell

George Mason University

Fall 2010

Overview

- **Textbook**
  - **Required:** ***Operating System Concepts*** **(8th edition), by Silberschatz, Galvin and Gagne. John Wiley & Sons, 2008.**
    - **equivalent versions okay (w/Java, Int'l Ed)**
  - ***Modern Operating Systems*** **(3rd Edition, 2008), by A. S. Tanenbaum**
- **Grading**
  - **Homework problems: 30%**
  - **Midterm exam: 20%**
  - **Research Paper: 20%**
  - **Final Exam: 30%**
- **Prerequisites**
  - **Computer Architecture**
  - **Working knowledge of C**
    - **C in a Nutshell, P.Prinz, 2006**
    - **Programming in C, S.Kochan, 3rd Ed, 2005**
    - **The Art of Concurrency, Breshears, 2009**
    - **Multicore Application Programming: for Windows, Linux, and Oracle Solaris, D.Gove, 2010**

Computer Architecture Prerequisite

- **Must have completed an undergraduate course on Computer Architecture/Organization (e.g. at GMU: CS 465, ECE 445, INFS 515) or equivalent**

- **Should be familiar with concepts such as:**
  - **Instruction set architectures**
  - **Assembly language (but not used in this course)**
  - **CPU control and datapaths**
  - **Instruction cycle (fetch/decode/execute)**
  - **Register Organization**
  - **Memory Hierarchy**
  - **Basic I/O Operations**

Programming Background

- **Must have completed a course on Data Structures and Program Design (e.g. CS 310, INFS 519) or equivalent**
- **Have *working knowledge* of C**
  - **any platform experience, but we will program on Linux and on Solaris**
- **It is assumed that you are *comfortable (not necessarily fluent/master/wizard)* with programming; the projects will involve some advanced concepts that will build on the fundamentals**
  - **Multi-threaded programming**
  - **Distributed system programming and design**

Operational Information

- CS Office: Engineering Building, Room 4300
- Office Hours:
  - by appointment
- E-mail: hfoxwell@cs.gmu.edu
  - use Subject: CS571: …your subject…

- Teaching Assistant: Changwei Liu, *cliu6@gmu.edu*
- Computer Accounts on `mason.gmu.edu` or `zeus.ite.gmu.edu`
- Class Web Pages:
  - http://cs.gmu.edu/~hfoxwell/ cs571
  - http://c4ilab.gmu.edu/moodle
- E-mails with additional info on lectures and assignments will be sent to students' GMU e-mail addresses
  - please manage your email account so that it does not max quota!
- Slides will be available at class web page and Moodle

Distance Education

- **CS 571 Fall'10 session is delivered to the Internet section (Section DL1) online by Network EducationWare (NEW), developed by Prof. Mark Pullen and his students at GMU.**

- **Students in all sections have accounts on NEW and can play back the lectures and download the PDF slide files at http://c4ilab.gmu.edu/disted**

- **Section DL1 (distance education) students will be given the midterm and final exam on campus, on the same day as Section 001 students**

OS/Programming Environments

- **CS Department Systems**
  - **Computer Accounts on `mason.gmu.edu`, `zeus.ite.gmu.edu`, `hercules.ite.gmu`**
- **Your Systems**
  - **Desktop or Laptop**
  - **Linux (Fedora, Ubuntu, CentOS, …)**
    - http://fedoraproject.org/
    - http://www.ubuntu.com/
    - http://centos.org/
  - **Solaris/OpenSolaris**
    - http://www.oracle.com/technetwork/server-storage/solaris/downloads/index.html
  - **Direct, Multi-boot, or Virtualized**
    - **VirtualBox (http://www.virtualbox.org/)**
    - **Parallels (http://www.parallels.com/products/)**
    - **VMware (http://www.vmware.com/products/workstation/)**
  - **GNU C Compiler (gcc, IDE, vi, emacs, …)**
    - **assignments must compile and run on zeus**

Course Objectives

- Why study operating systems?
  - VMware says "operating systems are dead". Agree?
  - Is anyone innovating in operating systems?
- Understanding the *principles* behind the design of *server and distributed* operating systems
- Observing how these principles are put into *practice* in real operating systems
- Discussing both "solved" and "open" problems and issues in OS design, recent trends
- Gaining hands-on experience in
  - Multithreaded programming
  - Distributed system programming and design

Topics

- Introduction
- Threads and Processes
- Inter-process Communication, Synchronization
- CPU Scheduling
- OS Observability
- Memory Management & Virtual Memory
- File and I/O Systems
- Cache Concepts
- Virtualization
- Distributed Systems

Lecture 1

- **Introduction**
  - **What is an Operating System?**
    - **yesterday? today? future?**
      - **where do OS's run?**
  - **Co-evolution of Computer Systems (processors) and Operating System Concepts &Features**
- **Computer System Structures**
- **Operating System Structures**

What is an Operating System?

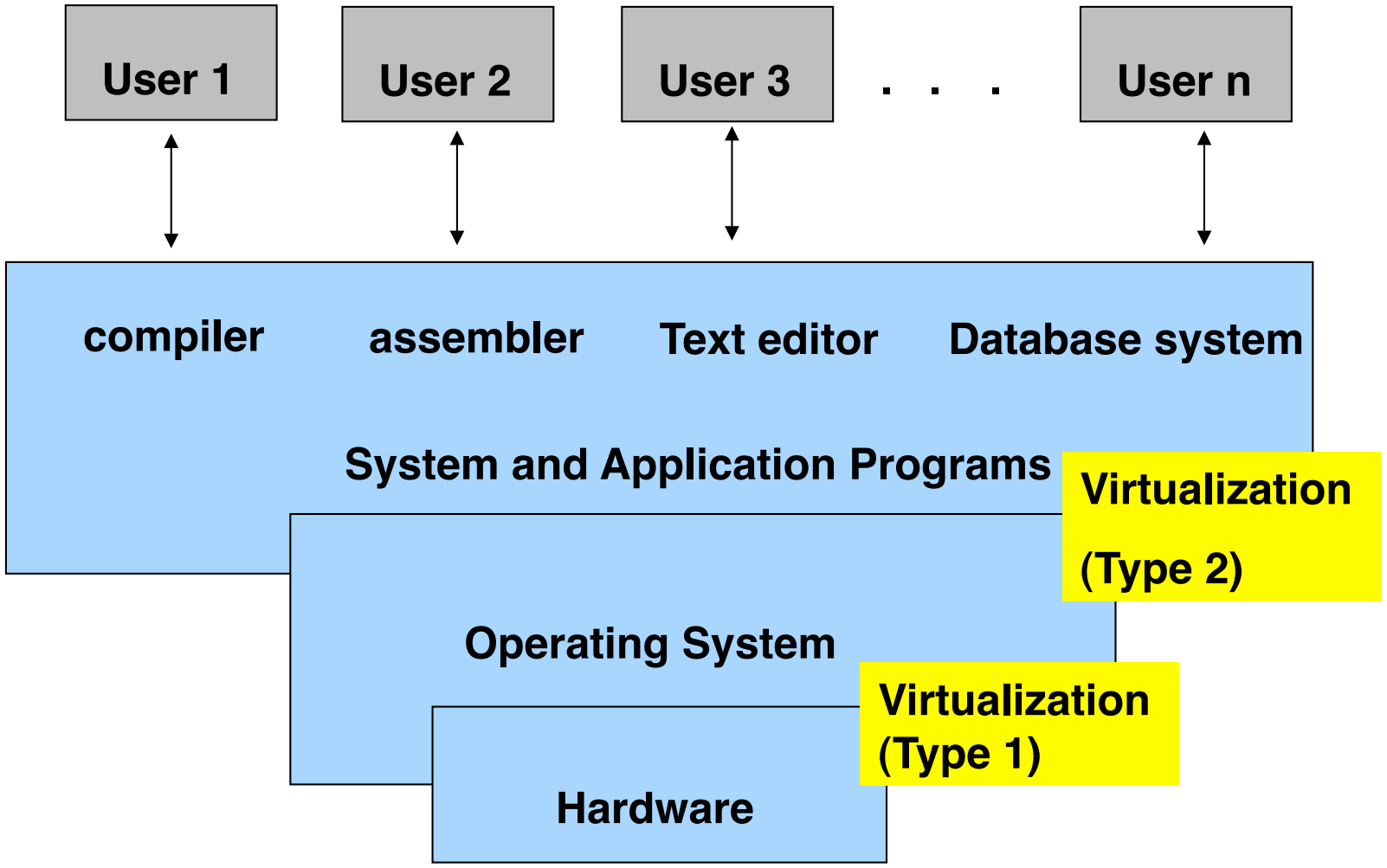
- A program that acts as an *intermediary* between the user of a computer and the computer hardware.
  - generally is in charge of all device access

- Operating system goals
  - Convenience: Make the computer system convenient to use.
  - Efficiency: Manage the computer system resources in an efficient manner
  - Which one has priority?

Computer System Components

1. *Hardware* – provides basic computing resources (CPU, memory, I/O devices).
2. *Operating system* – controls and coordinates the use of the hardware among the various application programs for the various users.
3. *Application programs* – define the ways in which the system resources are used to solve the computing problems of the users (compilers, database systems, video games, business programs).
4. *Users, interfaces (CLI, GUI)*

Computer System Components



Operating System Definitions: System View

- **Resource allocator – manages and allocates *resources*.**
- **Control program – controls and schedules the execution of user programs and operations of I/O devices.**
- **Kernel – the one program running at all times (all else being application programs).**

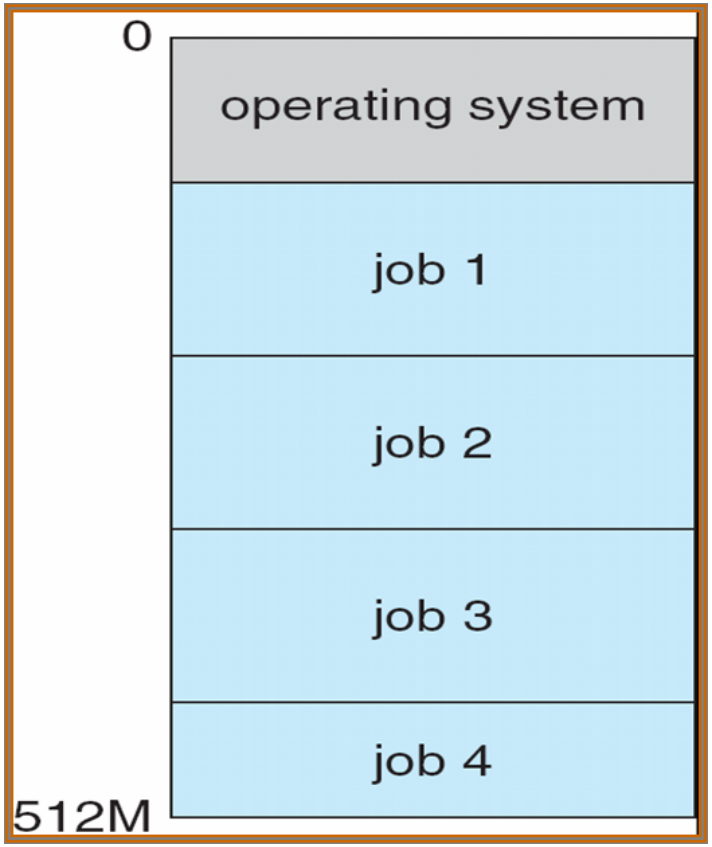
Co-evolution of Computer Systems and Operating System Concepts

- Mainframe Systems
- Batch Systems
- Multi-programmed Systems
- Time-sharing Systems
- Desktop Systems
- Modern Variants
  - Parallel Systems
  - Distributed Systems
  - Real-time and Embedded Systems
  - Handheld Systems

Mainframe Systems

- First computers to tackle many commercial and scientific applications
- Mainframes evolved through *batch, multiprogrammed* and *time-shared systems*
- Early systems were afforded only by major government agencies or universities:
  - physically enormous machines run from a console.
  - the user submitted the job to the human operator in the form of punched cards.
  - The operator collects the output and returns it to the user.
- *Batch systems:* To speed up processing, operators *batched* together jobs with similar needs and ran them through the computer as a group.

Multiprogrammed Systems



Several (and not necessarily similar) jobs are kept in the main memory at the same time, and the CPU is switched to another job when I/O takes place.

OS Features Needed for Multiprogramming

- **Job Scheduling – must choose the processes that will be brought to memory**
  - **problem: how are these choices made?**
- **CPU Scheduling – must choose among several jobs ready to run**
  - **problem: how are these choices made?**
- **Memory Management – must allocate the memory to several jobs**
  - **problem: how are these choices made?**
- **OS/360, developed by IBM to run on its System/360 series, was the first multiprogrammed operating system.**

Time-Sharing Systems: Interactive Computing

- Extension of multiprogrammed systems to allow on-line interaction with users.
- Each user is provided with an on-line session.
- *Response time* for each user should be short.

- A single CPU is multiplexed among several jobs that are kept in memory and on disk…
  - …but what is a CPU today?
- A job swapped in and out of memory to the disk.
- CPU is allocated to another job when I/O takes place.
- All active users must have a *fair* share of the CPU time (e.g. 10 ms for each user). But what is "fair"?

Desktop Systems

- *Personal computers* – computer system originally dedicated to a single user
  - consequences of this approach?
- I/O devices – keyboard, mouse, printers, USB Memory stick, disks, …
- *Objective:* User convenience and responsiveness.
  - Individuals have sole use of computers
  - A single user may not need advanced features of mainframe OS (maximize utilization, protection).
    - what's the assumption here?
- Today, may run several different types of operating systems (Windows, UNIX, BSD, MacOS, Linux)
  - per system, multi-boot, or use V12N

Parallel Systems

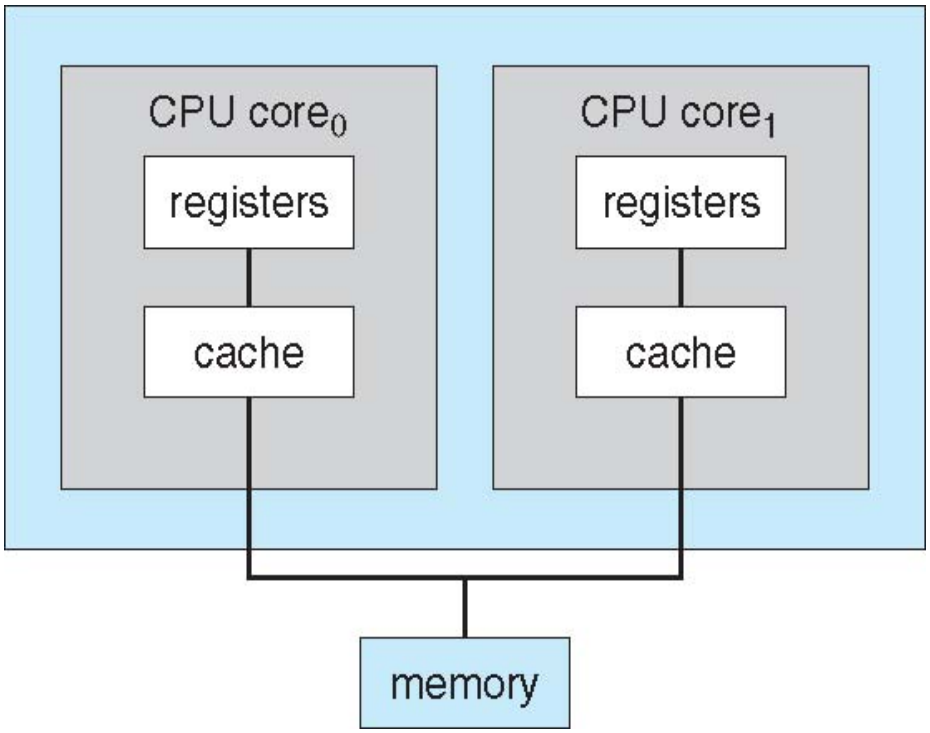
- Multiprocessor systems with more than one CPU in close communication.
- *Tightly coupled system* – processors share memory and a clock; communication usually takes place through the shared memory.
- Advantages of parallel system
  - Increased *throughput*
  - Economy of scale
  - Increased reliability
    - graceful degradation
    - fault-tolerant systems
- *What are the challenges of parallelism?*

Parallel Systems (Cont.)

- ***Symmetric multiprocessing (SMP)***
  - **All processors are *peers***
  - **Kernel routines can execute on any different CPUs, in parallel**
- ***Asymmetric multiprocessing***
  - **Master/slave structure**
  - **The kernel runs on a particular processor**
  - **Other CPUs can execute user programs and OS utilities.**
- **Which is easier to design?**

Parallel Systems (Cont.)

- **Multi-core architectures**
  - **Include multiple computing cores on a single chip**
    - **examples?**
  - **Faster (really?) and more energy-efficient than multiple chips with single cores**



Distributed Systems

- Distribute the computation among several physical processors.
- *Loosely coupled system* – each processor has its own local memory; processors communicate with one another through various communications lines
- Advantages of distributed systems
  - Resource and Load Sharing
  - Reliability, failover
  - Communications

Clustered Systems

- A set of computers that share storage (but not main memory) and that are closely linked via a fast local-area network
- Typically used to provide *high availability* services through proper *middleware*
  - Asymmetric versus symmetric modes
- May be also used to support *high performance computing (HPC), or "grid computing"*
  - *for massive scaling*

Real-Time and Embedded Systems

- A real-time system is used when rigid time requirements have been placed on the operation of a processor or the flow of data.

- An embedded system is a component of a more complex system
  - Control of a nuclear plant
  - Missile guidance
  - Control of home and car appliances (microwave oven, DVD players, car engines, …)

- Real-time systems
  - have well-defined time constraints
    - "predictable latency"
  - may be either *hard* or *soft* real-time.

Real-Time Systems (Cont.)

- **Hard real-time**
  - Critical tasks must be completed on time
  - Secondary storage limited or absent, data stored in short term memory, or read-only memory (ROM)
- **Soft real-time**
  - No absolute timing guarantees (e.g. “best-effort scheduling”)
  - Limited utility in industrial control of robotics
  - Useful in applications (multimedia, virtual reality) requiring advanced operating-system features.

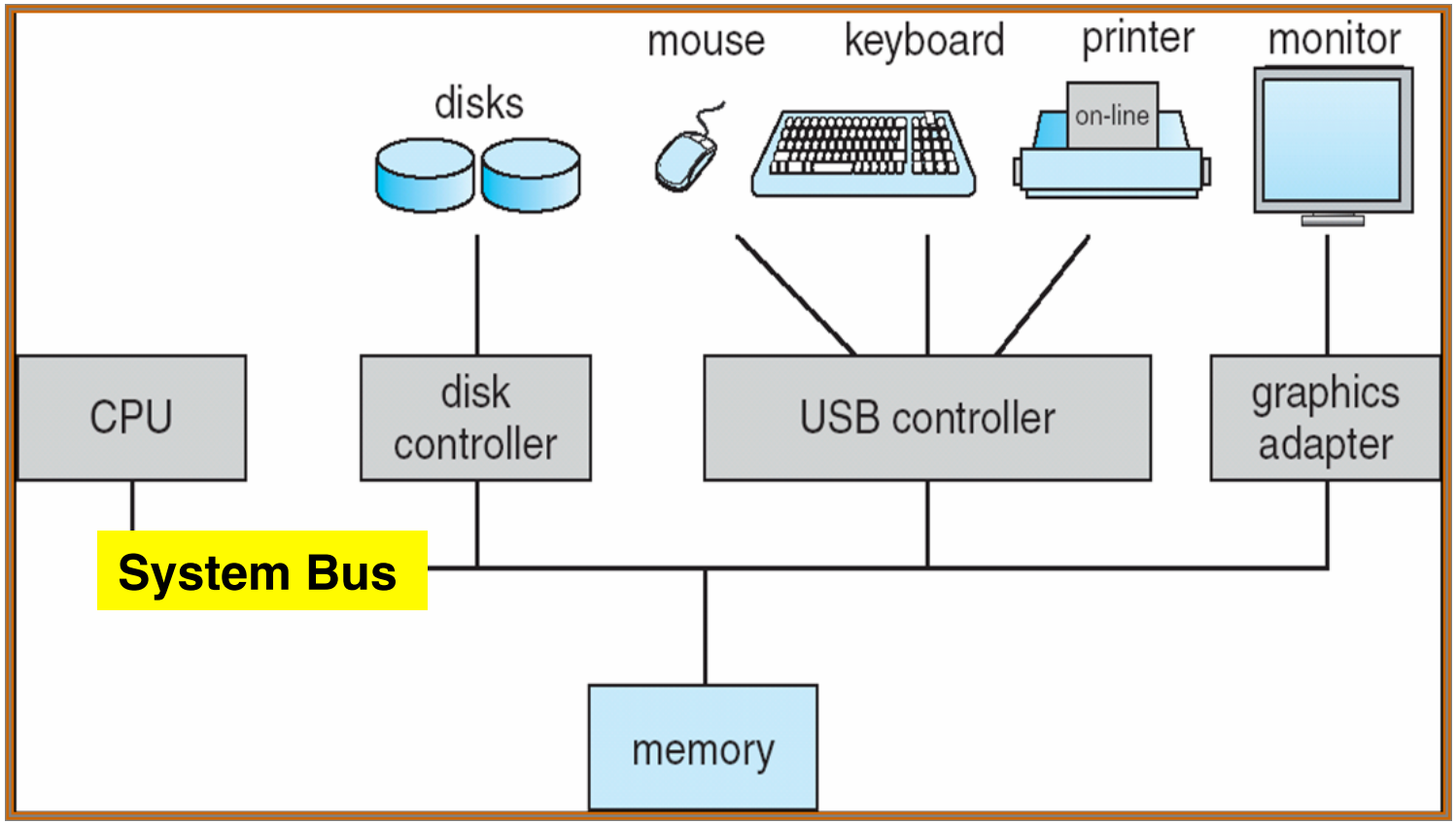
Handheld Systems

- Personal Digital Assistants (PDAs)
  - iPhone, Droid, BlackBerry, …
    - really, computers that can make phone calls
  - iPad, “tablet computers”, netbooks
- Cellular telephones
- Issues
  - Limited memory
  - Limited battery power
  - Slow processors
  - Small display screens
  - but all these are getting better

Computer-System Structures

- Computer System Organization
- Instruction Execution
- Computer System Operation
- Interrupt Processing
- Storage Structure
- Storage Hierarchy

Computer-System Organization



Instruction Execution

- While executing a program, the CPU:
  - fetches the next instruction from memory
  - decodes it to determine its type and operands
  - executes it
- May take multiple clock cycles to execute an instruction
- Examples:
  - LOAD R1, #3
  - LOAD R2, M2
  - STORE M3, R4
  - ADD R1, R2, R3
- Each type of CPU has a specific set of instructions that it can execute *(instruction-set architecture)*
  - *x86, x64, SPARC, Power, MIPS, …*

Instruction Execution

- **Registers**
  - General registers
  - Program Counter (PC): contains the memory address of the next instruction to be fetched.
  - Stack Pointer (SP): points to the top of the current *stack* in memory. The stack contains one frame for each procedure that has been entered but not yet exited.
  - Program Status Word (PSW): contains the condition code bits and various other control bits.

- When *time multiplexing* the CPU, the operating system will often stop the running program to (re) start another one. In these cases, it must save the "state information" (e.g. values of the registers).
  - this is a "context switch", and has performance impact

Computer-System Operation

- I/O devices and CPU can execute concurrently.
- Each device controller has local buffer(s).
- CPU moves data from/to main memory to/from local buffers
  - need to understand how OS handles caching
- I/O is from/to the device to/from local buffer of controller.
- The *device driver* is a special operating system software that interacts with the device controller.
- Typically, the device controller informs CPU that it has finished its operation by causing an *interrupt*.

Interrupts

- Important *events* are signaled to the CPU by an interrupt
- Hardware (e.g. a device controller) may raise an interrupt by way of the system bus
- Software may trigger an interrupt by executing a special instruction called a *system call* (or, *monitor call)*
- Once the interrupt is detected (typically at the end of each instruction), it must be dealt with through interrupt processing sequence

Classes of Interrupts

- I/O Interrupts: Generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions.
- Timer Interrupts: Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.
- Hardware Failure Interrupts: Generated by a failure (e.g. power failure or memory parity error).
- Traps (Software Interrupts): Generated by some condition that occurs as a result of an instruction execution
  - Errors
  - User request for an operating system service

Interrupt Mechanism

- Interrupt transfers control to the interrupt service routine generally through the *interrupt vector* which contains the addresses of all the service routines.
- Interrupt Service Routines (ISRs): Separate segments of code determine what action should be taken for each type of interrupt.
- Once the interrupt has been serviced by the ISR, the control is returned to the interrupted program.
- Need to save the "process state" before ISR takes over.
- Modern operating systems are *interrupt-driven*.
  - too many interrupts can affect performance

Basic Interrupt Processing

1. The interrupt is issued
2. Processor finishes execution of current instruction
3. Processor signals acknowledgement of interrupt
4. Processor pushes PSW and PC onto control stack
5. Processor loads new PC value through the interrupt vector
6. ISR saves remainder of the process state information
7. ISR executes
8. ISR restores process state information
9. Old PSW and PC values are restored from the control stack

- ✓ What if another interrupt occurs during interrupt processing?
- ✓ How can we observe interrupts?

Direct Memory Access Structure

- Used for high-speed I/O devices able to transmit information at (close to) memory speeds.
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention.
- Only one interrupt is generated per block, rather than one interrupt per byte/word.

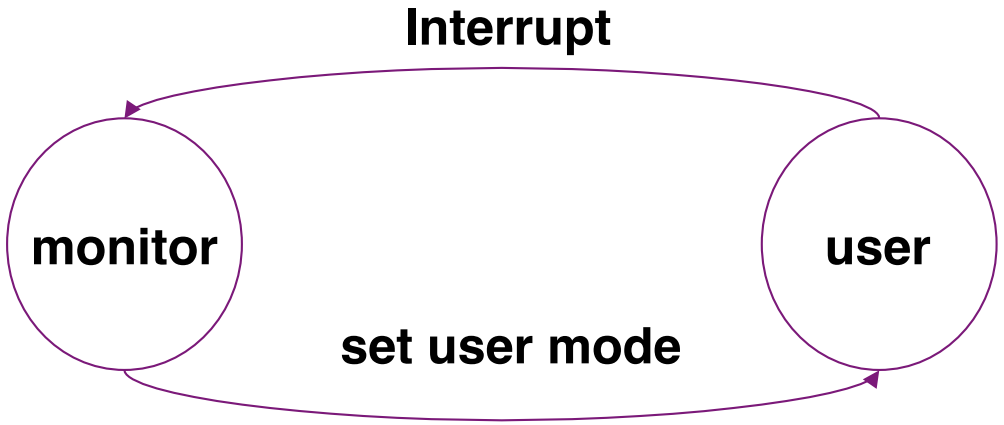
Dual-Mode Operation

- **Operating System must protect *itself* and all other programs (and their data) from any malfunctioning program.**

- **Provide hardware support to differentiate between at least two modes of operations.**
  1. ***User mode*** **– execution done on behalf of a user.**
  2. ***Monitor mode*** **(also *kernel mode* or *system mode*) – execution done on behalf of operating system.**
  3. **system constantly switches between user & kernel mode**
     2. **how can we observe this?**

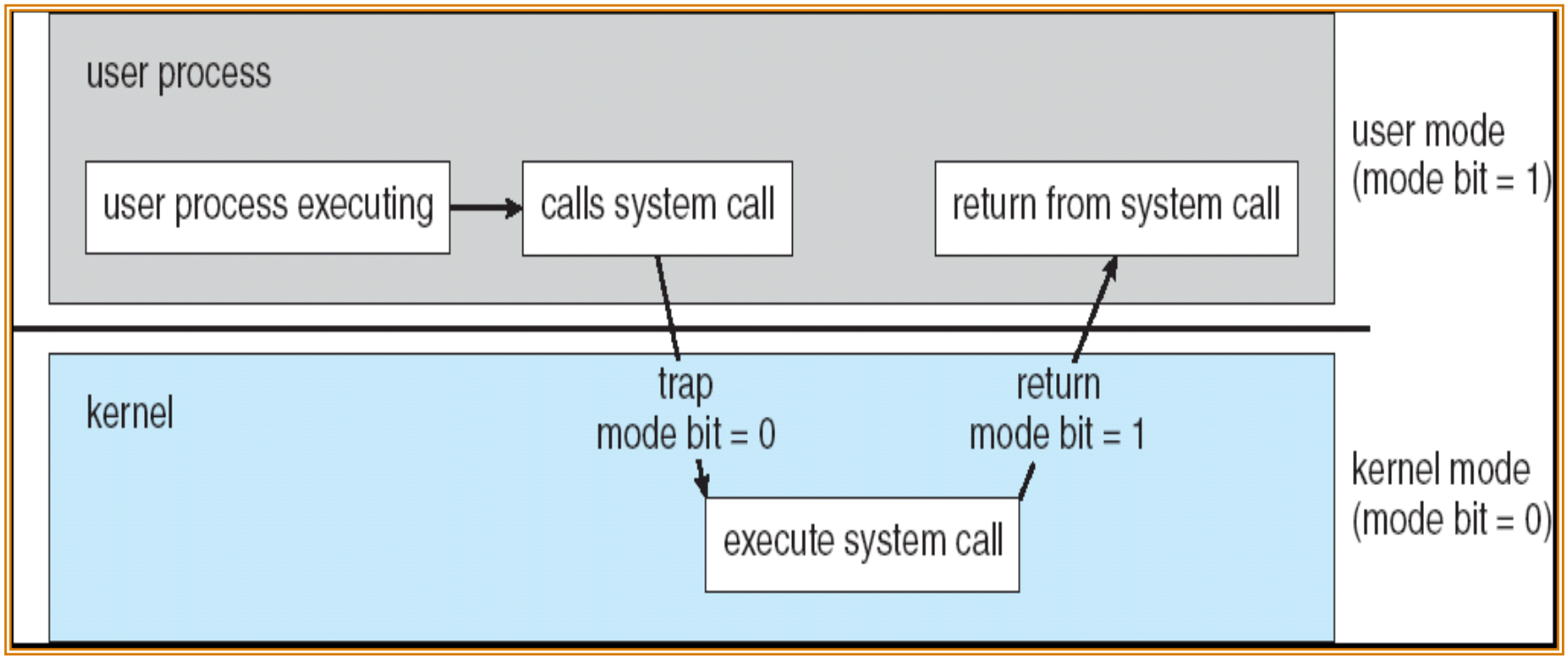
Dual-Mode Operation (Cont.)

- *Mode bit* added to computer hardware to indicate the current mode: monitor (0) or user (1).
- When an interrupt occurs hardware switches to monitor mode.



Privileged instructions can be issued only in monitor mode.

Transition From User to Kernel Mode



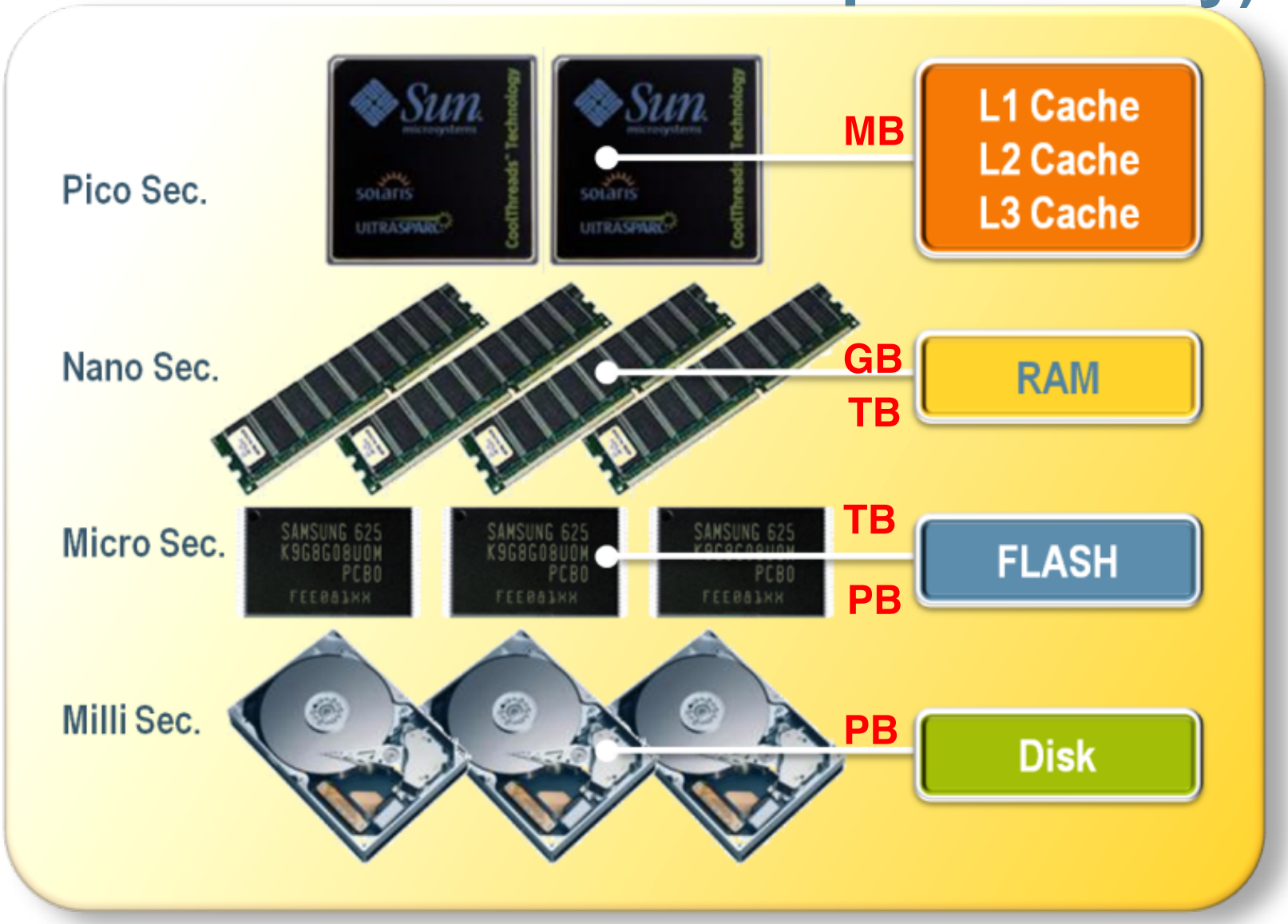
The system call can be executed by a generic `trap` (or, `syscall` instruction).

Storage Structure

- Main memory – only large storage media that the CPU can access directly…but S...L...O...W...
- Secondary storage – extension of main memory that provides large nonvolatile storage capacity.
- Magnetic disks – rigid metal or glass platters covered with magnetic recording material
  - Disk surface is logically divided into *tracks*, which are subdivided into *sectors*.
  - The *disk controller* determines the logical interaction between the device and the computer.
- Flash (Solid State) "Disks" becoming more widely used and cost-competitive with spinning disks

Size Matters!

(But note inverse relationship to latency)

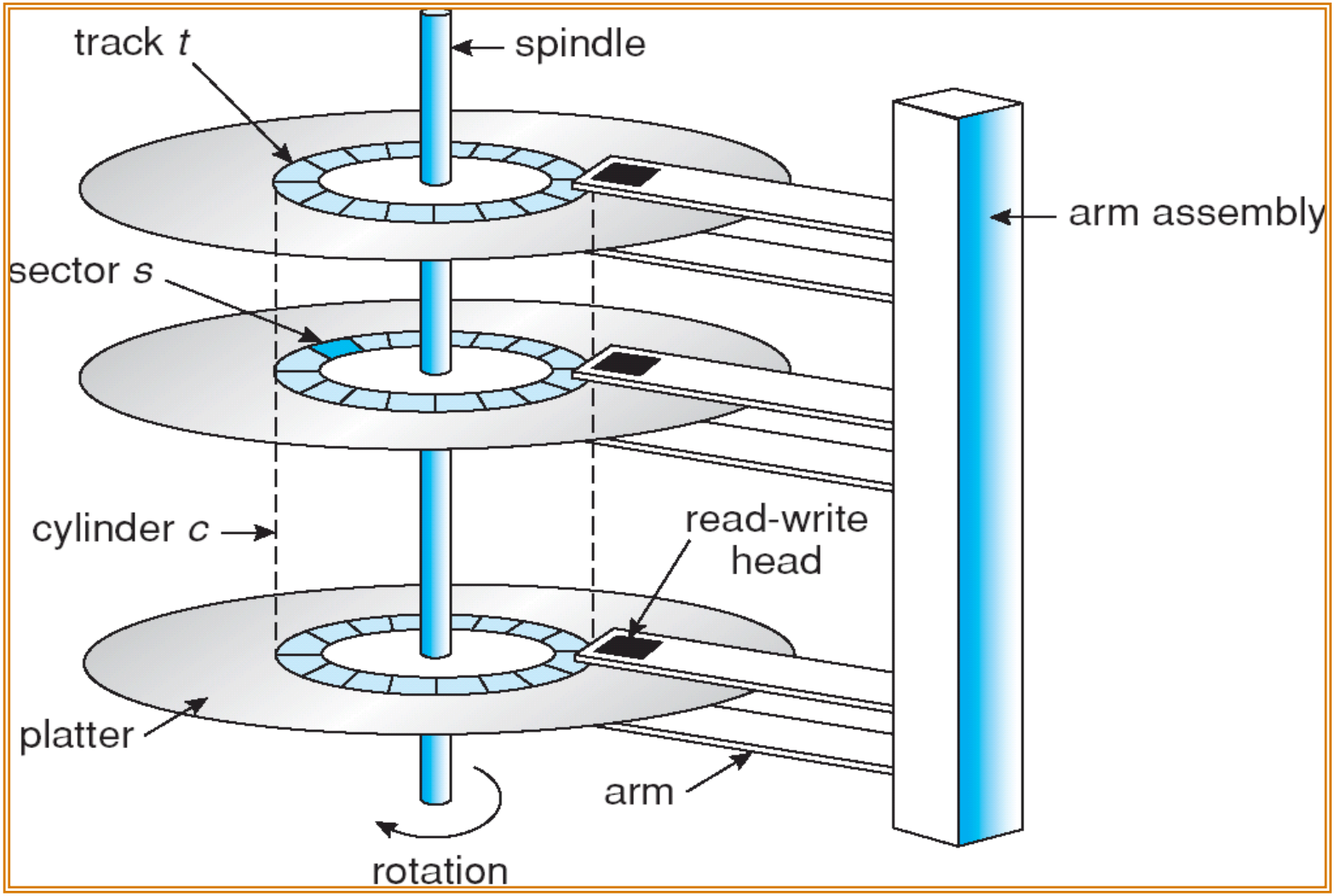


Latency Matters!

Level	Latency	Seconds	Time To Insight		
			Seconds	Hours	Days
NH L3 Cache	52 Clocks	1.62×10^{-8}	1		
Local Memory	150 ns	1.5×10^{-7}	9.23		
32 Core NUMA	3 Hops	5.19×10^{-7}	31		
SSD	125 us	1.25×10^{-4}	7,692	2.14	
Gb Ethernet	1 ms	1×10^{-3}	61,538	17	.71
Disk	3.6 ms	3.6×10^{-3}	221,538	62	2.56

- **In 30 seconds, you can stop someone at the border**
- **In 2 hours, you can intercept someone**
- **In 2 days, actionable intelligence is forfeited**

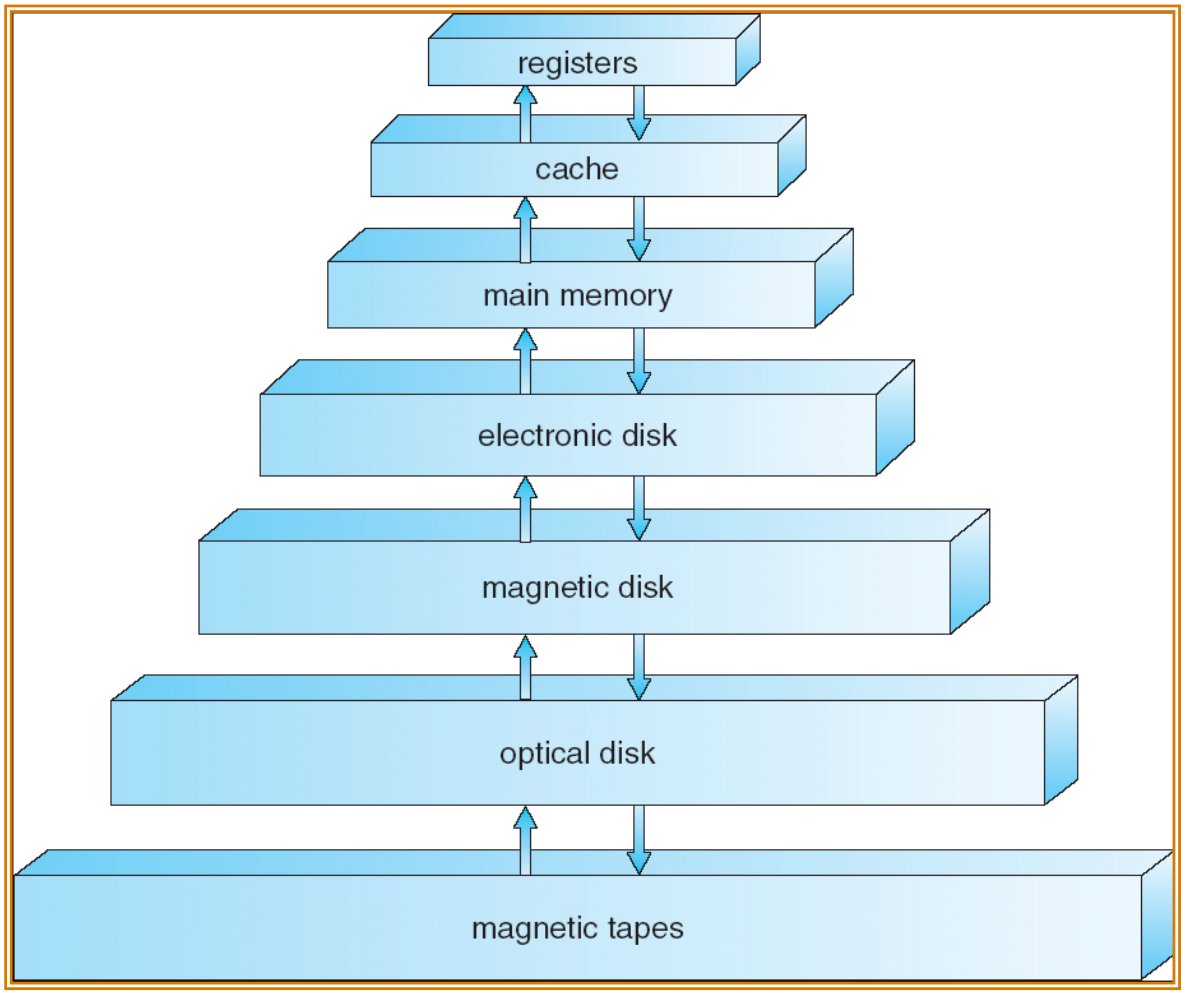
Moving-Head Disk Mechanism



Storage Hierarchy

- Storage systems organized in hierarchy
  - Speed
  - Cost
  - Volatility
- Faster access time, greater cost per bit
- Greater capacity, lower cost per bit
- Greater capacity, slower access speed

Storage-Device Hierarchy



Cache Memory

- Use of high-speed memory to hold recently-accessed data.
- Requires a *cache management* policy.
- *Caching* is a rather general mechanism
  - copying information into faster storage system
  - e.g. main memory as a *cache* for secondary storage, proxy server as a cache for remote web server,…
  - introduces the *consistency* problem.

Operating-System Structures

- System Components
  - Process Management
  - Main Memory Management
  - File System Management
  - Mass Storage Management
  - I/O System Management
  - Protection and Security
- User Operating-System Interface
- System Calls
- System Programs

Process Management

- A *process* is a program in execution
- A process needs certain resources, including CPU time, memory, files, and I/O devices to accomplish its task
- The operating system is responsible for
  - Process creation and deletion
  - Process suspension and resumption
  - Provision of mechanisms for
    - Process synchronization
    - Process communication
    - Deadlock handling

Main Memory Management

- The main memory is
  - a large array of words/bytes, each with its own address
  - a volatile storage device

- The operating system will
  - Keep track of which parts of memory are currently being used and by whom
  - Decide which processes to load when memory space becomes available
  - Allocate and deallocate memory space as needed

File System Management

- A file is a collection of related information defined by its creator
- Commonly, files represent programs (both source and object forms) and data
- The operating system responsibilities
  - File creation and deletion
  - Directory creation and deletion
  - Support of primitives for manipulating files and directories
  - Mapping files onto secondary storage
  - File backup on stable (non-volatile) storage media

Mass-Storage Management

- The *secondary storage* backs up main memory and provides additional storage.
- Most common secondary storage type: disks
- The operating system is responsible for
  - Free space management
  - Storage allocation
  - Disk scheduling
- For long-term archival storage, *tertiary storage* devices may be used
  - Magnetic tape, CD, DVD

I/O System Management

- The Operating System will hide the peculiarities of specific hardware from the user.
- In Unix, the I/O subsystem consists of:
  - A buffering, caching and spooling system
  - A general device-driver interface
  - Drivers for specific hardware devices
- Interrupt handlers and device drivers are crucial in the design of efficient I/O subsystems.

Protection and Security

- *Protection* refers to a mechanism for controlling access of processes, or users to the resources defined by a computer system.
  - authentication
  - authorization
  - Data CIA
    - Confidentiality, Integrity, Availability
- *Security* system must provide defense against both internal and external attacks
  - Worms and viruses
  - Denial of service attacks
  - Buffer overflow attacks
  - Lots of other horrible stuff (See Comp Sec Course!)

User Operating-System Interface

- **Two main approaches**
  - ***Command-line interpreter*** **(a.k.a *command interpreter*, or *shell*)**
  - ***Graphical User Interfaces (GUI)***
- **The shell**
  - **allows users to directly enter commands that are to be performed by the operating system**
  - **is usually a system program (not part of the kernel)**
- **GUI allows a mouse-based window-and-menu system**
  - **WIMP: Windows, Icons, Menus, Pointer**
- **Most general purpose systems allow both**

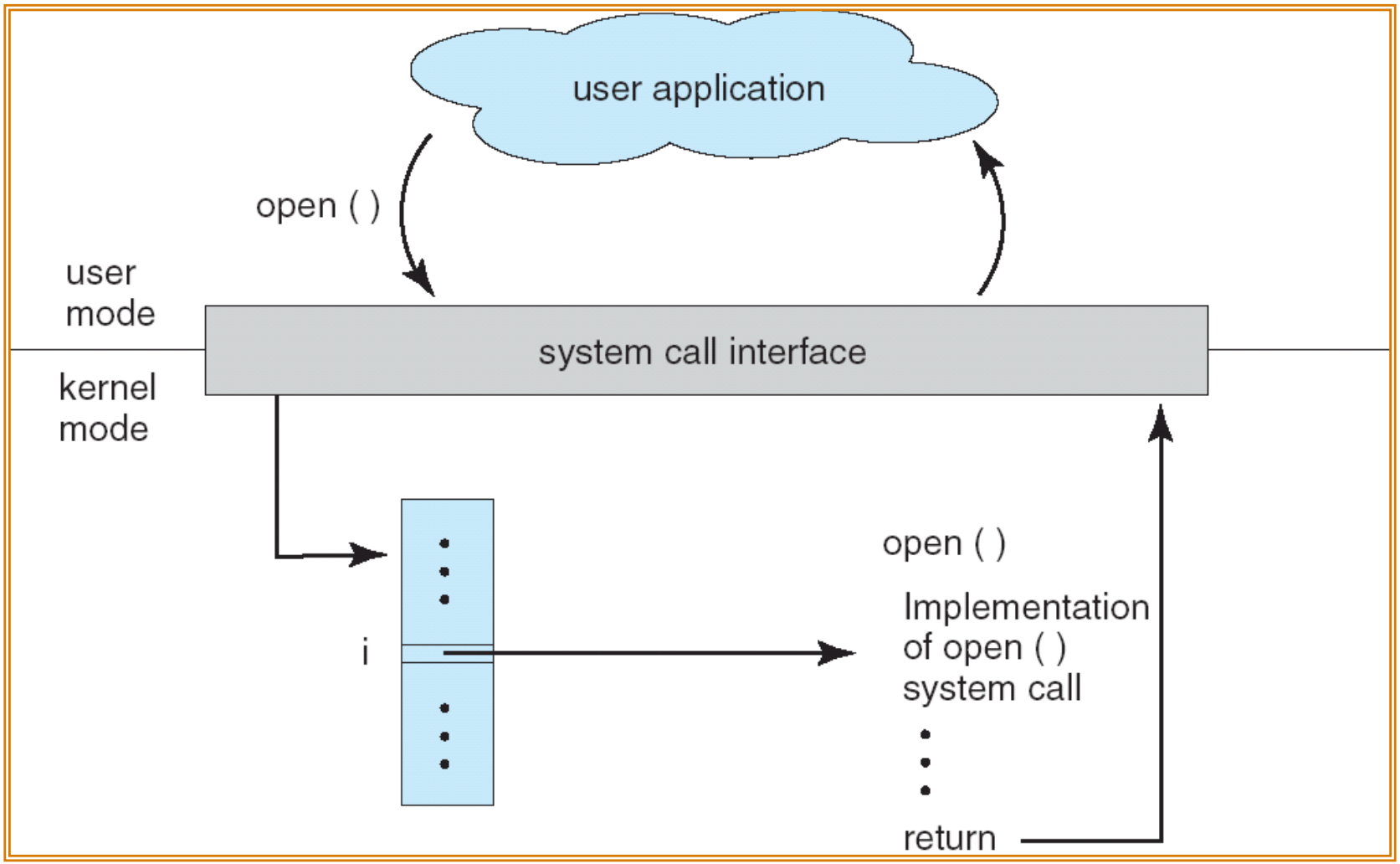
System Calls

- System calls provide the interface between a running program and the operating system.
- Typically, application programmers design programs using an *application programming interface (API).*

- The run-time support system (run-time libraries) provides a *system-call interface*, that intercepts function calls in the API and invokes the necessary system call within the operating system.

- The library routines check the arguments given by the application, build a data structure to convey the arguments to the kernel, and then execute a privileged instruction (`trap` or `syscall`), which triggers the interrupt processing sequence.

Example System-Call Processing



Major System Calls in Unix : Process Management

- pid = fork()
  - Create a child process identical to the parent
- pid = waitpid(pid, &statloc, options)
  - Wait for a child to terminate
- s = execve(name, argv, environp)
  - Replace a process' core image
- exit(status)
  - Terminate process execution and return status
- s = kill (pid, signal)
  - Send a signal to a process

Major System Calls in Unix : File Management

- fd = open (file, how, …)
  - Open a file for reading, writing or both
- s = close (fd)
  - Close an open file
- n = read (fd, buffer, nbytes)
  - Read data from a file into a buffer
- n = write (fd, buffer, nbytes)
  - Write data from a buffer into a file
- position = lseek(fd, offset, whence)
  - Move the file pointer
- s = stat(name, &buf)
  - Get a file's status information

Major System Calls in Unix : Directory and File System Management

- s = mkdir(name, mode)
  - Create a new directory
- s = rmdir (name)
  - Remove an empty directory
- s = link (name1, name2)
  - Create a new directory, name2, pointing to name1
- s = unlink (name)
  - Remove a directory entry
- s = mount (special, name, flag)
  - Mount a file system
- s = umount(special)
  - Unmount a file system

System Programs

- System programs provide a convenient environment for program development.
- They can provide various services
  - Status information
  - File modification
  - Programming language support
  - Program loading and execution
  - Communications

- ***Most users' view of the operating system is defined by system programs, not by the actual system calls.***

Major Questions

- How can we know that an operating system is doing what it is designed to do? correctly? efficiently?
- What tools are available to observe operating system actions? are they “safe” to use?