Topics: Early systems and OS overview
Skim Chapters 1-2 of SGG
Read Chapter 1 of USP

CS 3733 Operating Systems

Instructor: Dr. Turgay Korkmaz

Department Computer Science

The University of Texas at San Antonio

Office:	**NPB 3.330**
Phone:	**(210) 458-7346**
Fax:	**(210) 458-4437**
e-mail:	**korkmaz@cs.utsa.edu**
web:	**www.cs.utsa.edu/~korkmaz**

These slides are prepared based on the materials provided by Drs. Robbins, Zhu, and Liu. Thanks to all.

Lecture Outline

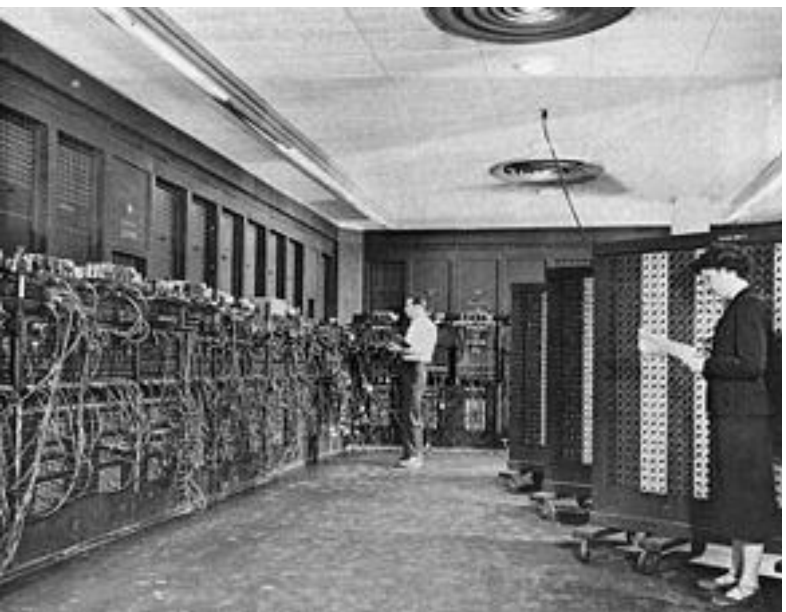
- Evolution of Computer Systems and OS Concepts
- Operating System: what is it?
- Different types/variations of Systems/OS
  - Parallel/distributed/real-time/embedded OS etc.
- OS as a resource manager
  - How does OS provide service? – interrupt/system calls
- OS Structures and basic components
  - Process/memory/IO device managers

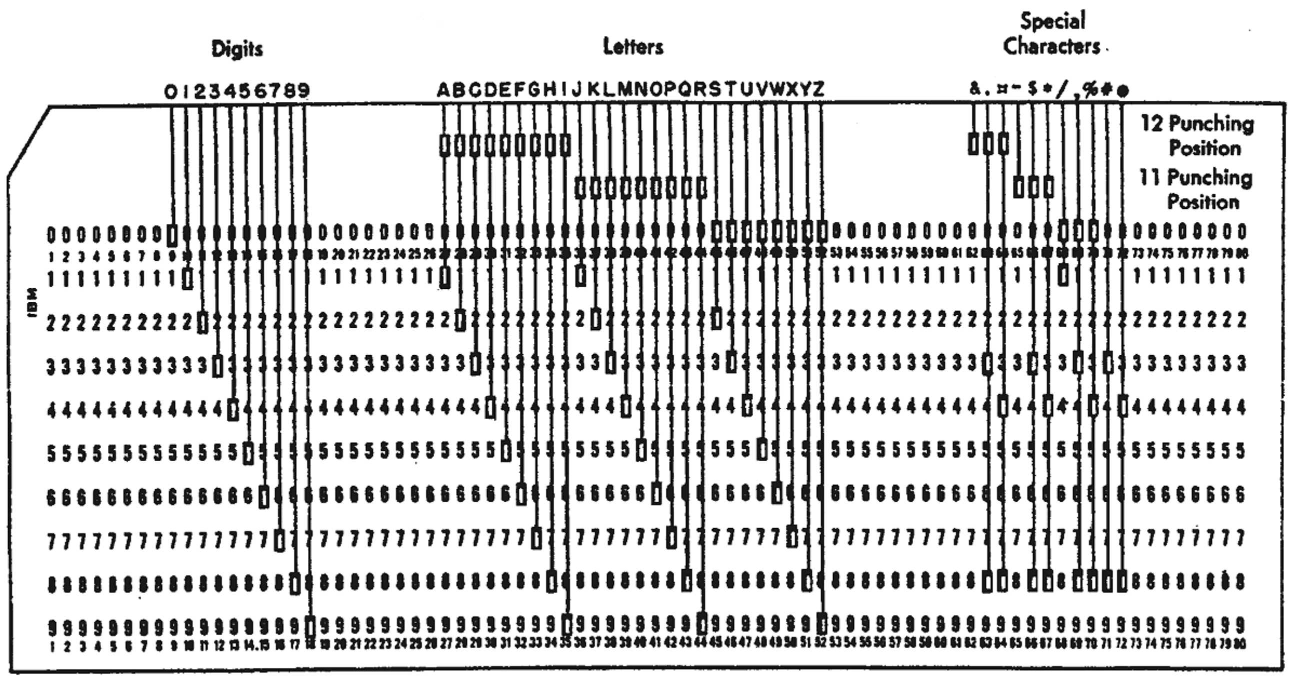
Lecture Outline

- Evolution of Computer Systems and OS Concepts
- Operating System: what is it?
- OS as a resource manager
  - How does OS provide service? – interrupt/system calls
- OS Structures and basic components
  - Process/memory/IO device managers
- Different types/variations of Systems/OS
  - Parallel/distributed/real-time/embedded OS etc.

First Computer: ENIAC in 1940s

- Big: 27 tons, 680 ft^2, and use 150kW
- Slow: tens instructions/s
- Limited functions: addition / multiplication
- Hard to use: Button switch or Punch card I/O





First Computer: ENIAC (cont.)

- Got problem with the program?! → you are in trouble ☺
- Process one job at a time
  - Human is slow
  - CPU time is precious
- Batch systems
  - Read in more jobs
  - Process one by one
  - I/O devices are still slow?

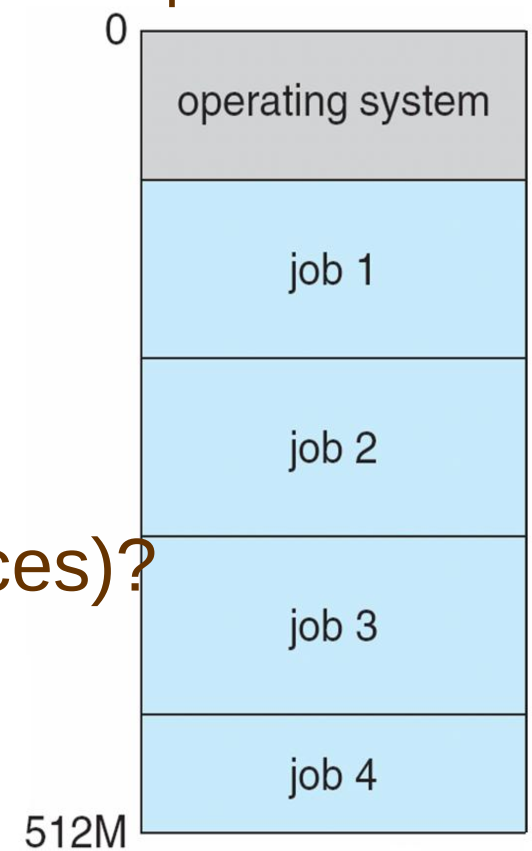
Debug your program!

***

Multiprogrammed Systems

A single program cannot keep busy for CPU and IO, thus wasting resources

- Several jobs run "*concurrently*"
  - ➢ Job: computing → input → computing → … → output
  - ➢ *Take turns* to use CPU and I/O devices



- But ***which*** job uses ***what*** and ***when***?
  - ➢ Need a manager/supervisor → OS
- ***How*** to use the hardware (e.g., I/O devices)?
  - ➢ Resource manager/interface → OS

Time Sharing Systems

- Extension to multi-programmed systems
- Multiple **interactive** users
  - Allow on-line interaction with users;
  - ***Response time*** for each user should be short
- CPU is multiplexed among several jobs of several users that are kept in memory
  - CPU is allocated to jobs in **Round-Robin** manner
  - All active users must have a ***fair*** share of the CPU time: e.g. with 100 ms time quantum
- Example systems: IBM 704 and 7090

Justification for Time Sharing Systems

Table 1.1. Typical times for components of a computer system. One nanosecond (ns) is 10^{-9} seconds, one microsecond (μs) is 10^{-6} seconds, and one millisecond (ms) is 10^{-3} seconds.

item	time		scaled time in human terms (2 billion times slower)	
processor cycle	0.5 ns	(2 GHz)	1	second
cache access	1 ns	(1 GHz)	2	seconds
memory access	15 ns		30	seconds
context switch	5,000 ns	(5 μs)	167	minutes
disk access	7,000,000 ns	(7 ms)	162	days
quantum	100,000,000 ns	(100 ms)	6.3	years

From USP, Robbins

Desktop Systems: 1980s

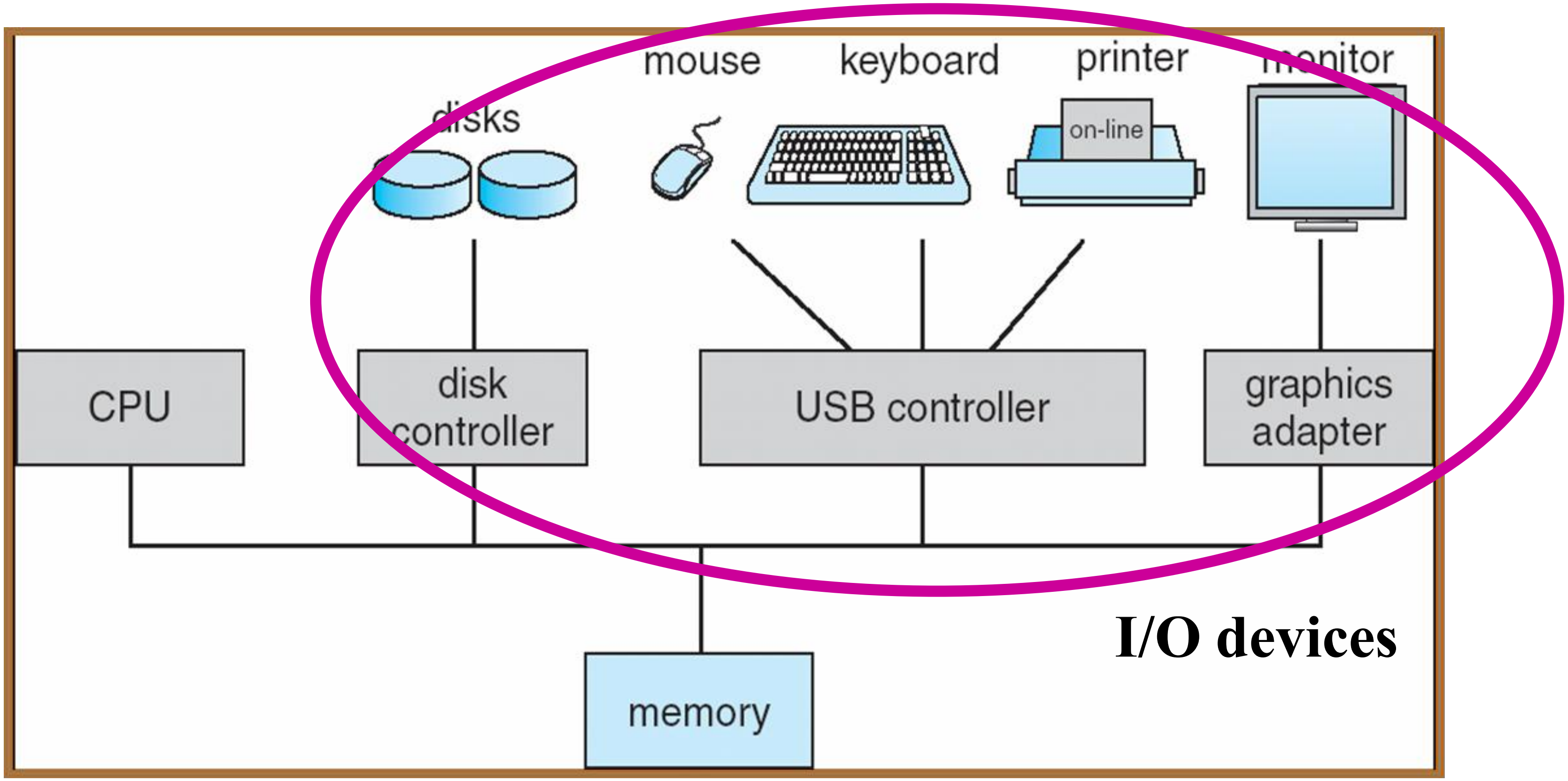
- *Personal computers* **dedicated to a single user**;
- *Objective:* User convenience and responsiveness.
  - Individuals have sole use of computers
  - A single user may not need advanced features of mainframe OS (maximize utilization, protection).
- I/O devices – display, keyboard, mouse and printers
- Today, may run several different types of operating systems (Windows, MacOS, Linux)

Parallel High-Performance Systems

- Goals:
  - Increased performance/throughput
  - Increased reliability: fault tolerance
- Multiprocessor systems: more than one CPUs
  - *Tightly coupled system* – processors share memory, bus, IO, and a clock; communication usually takes place through the shared memory
- *Symmetric multiprocessing (SMP) vs. asymmetric*
  - SMP: each processor runs an identical copy of the operating system; all processors are peers
  - Asymmetric: master-slave

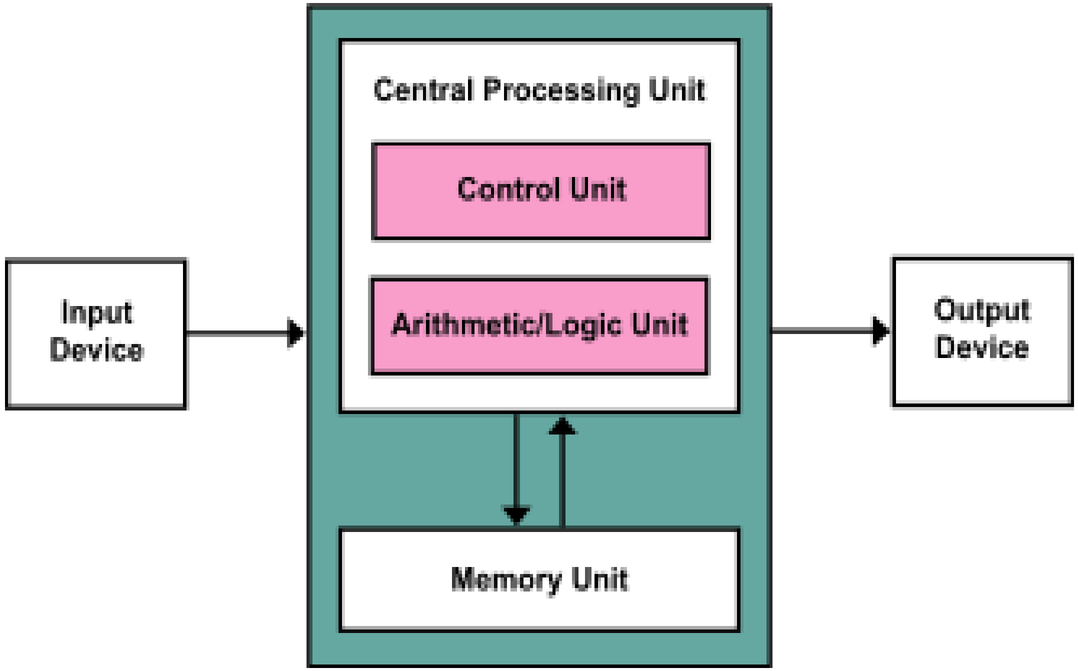
*

Modern Computer Hardware Organization



von Neumann Architecture: store and computation

Von Neumann Architecture



Before, computers have fixed uses

This describes a design architecture:

- A processing unit containing an ALU and processor registers
- A control unit containing an instruction register or program counter
- **A memory to store both data and instructions (Stored-program computer)**
- External mass storage, and input and output mechanisms.

Lecture Outline

- Evolution of Computer Systems and OS Concepts
- Operating System: what is it?
- OS as a resource manager
  - How does OS provide service? – interrupt/system calls
- OS Structures and basic components
  - Process/memory/IO device managers
- Different types/variations of Systems/OS
  - Parallel/distributed/real-time/embedded OS etc.

What Operating Systems do?

+

OS (Windows) *$200*

+

Applications *$$$$*

$300

Do I have to?!

Yes!
Otherwise, a set of silicon circuits doing nothing good for you !

Operating system (OS) goals:

- Execute user programs and make solving user problems easier
- Make the computer system convenient to use
- Use the computer hardware in an efficient manner

What is an Operating System (OS)?

*

User View

User interface, ease of use vs. performance *(windows vs. command line; terminal vs. minicomputer; networked workstations; handheld; embedded)*

System View

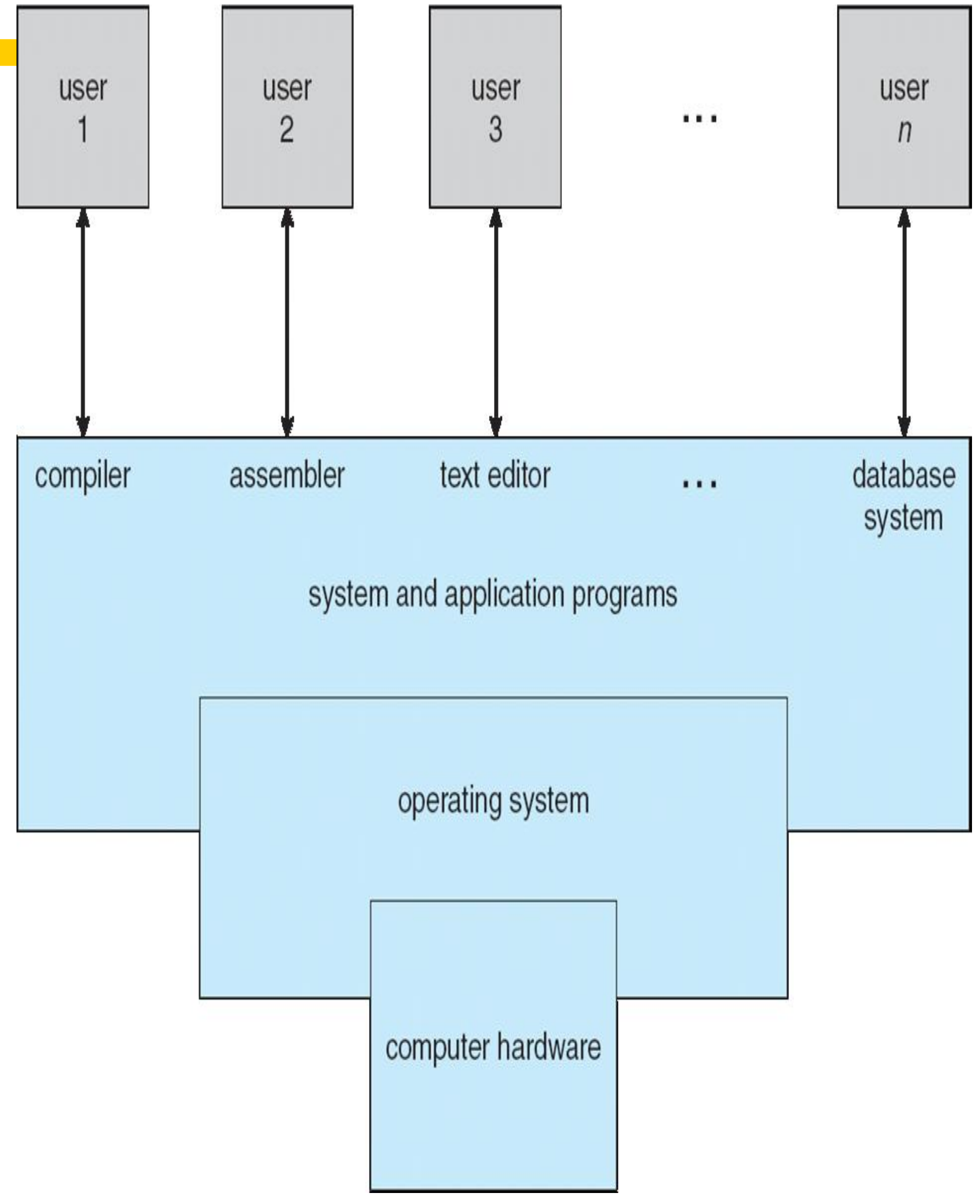
OS is a **resource allocator**

Manages all resources *(CPU, memory, file-storage, I/O devices etc.)*

Decides between conflicting requests for efficient and fair resource use

OS is a **control program**

Controls execution of programs to prevent errors and improper use of the computer



Operating System Definition(s)

Based on the discussion so far, can you define Operating System?

- Here is a good approximation

  "Everything a vendor ships when you order an operating system"

- "The one program running at all times on the computer" is the **kernel**. Everything else is either a system program (ships with the operating system) or an application program
- So we may say: OS is a program (or programs) that
  - **manages** computer hardware and resources,
  - **provides** user friendly interface for applications and users
- No universally accepted or completely adequate definition
- Instead of a definition, we should focus on the **tasks** that are necessary to accomplish the OS goals:
  - Execute user programs and make solving user problems easier (user/programmer view)
  - Make the computer system convenient to use… (user view)
  - Use the computer hardware in an efficient manner… (system view)

HOW?

- The amazing aspect of Operating Systems is how **varied** they are in performing their tasks
  - Different needs, capabilities, etc
  - Ease of use vs. performance vs. cost

Retrieval practice…

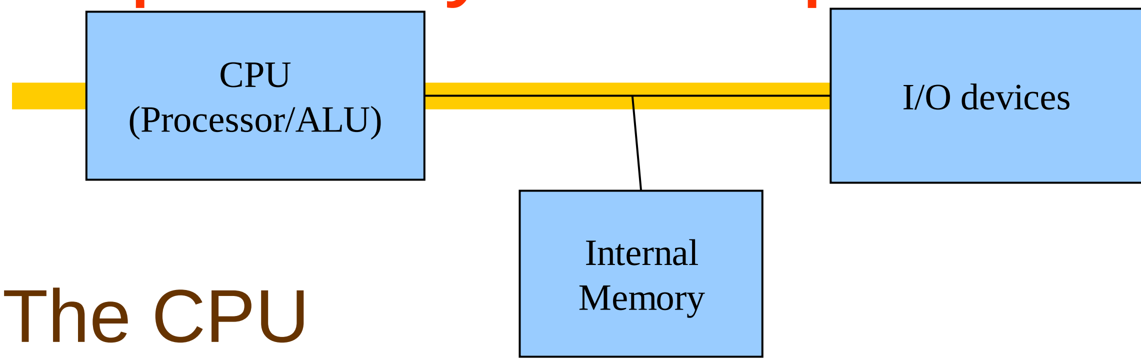
- What is Operating System (OS)?
  - A program (kernel) that runs at all times and acts as an ***intermediary*** between the user/programs and hardware.
    - **manages** computer hardware and resources,
    - **provides** user friendly interface for applications and users
- What are the Goals of Operating System
  - **Convenience**: Make the computer convenient to use for general user and programmers..
  - **Efficiency**: Manage system resources in an efficient manner

Lecture Outline

- Evolution of Computer Systems and OS Concepts
- Operating System: what is it?
- OS as a resource manager
  - How does OS provide service? – interrupt/system calls
- OS Structures and basic components
  - Process/memory/IO device managers
- Different types/variations of Systems/OS
  - Parallel/distributed/real-time/embedded OS etc.

Computer-System Operation

**



Computer Startup: **bootstrap program** is loaded at power-up or reboot: Typically stored in ROM or EPROM, generally known as **firmware.** It initializes all aspects of system and loads operating system **kernel** and starts execution

- The CPU
  - load instructions from main memory,
  - load/store data from/to memory system
  - controls I/O devices to perform certain tasks

User process or kernel process

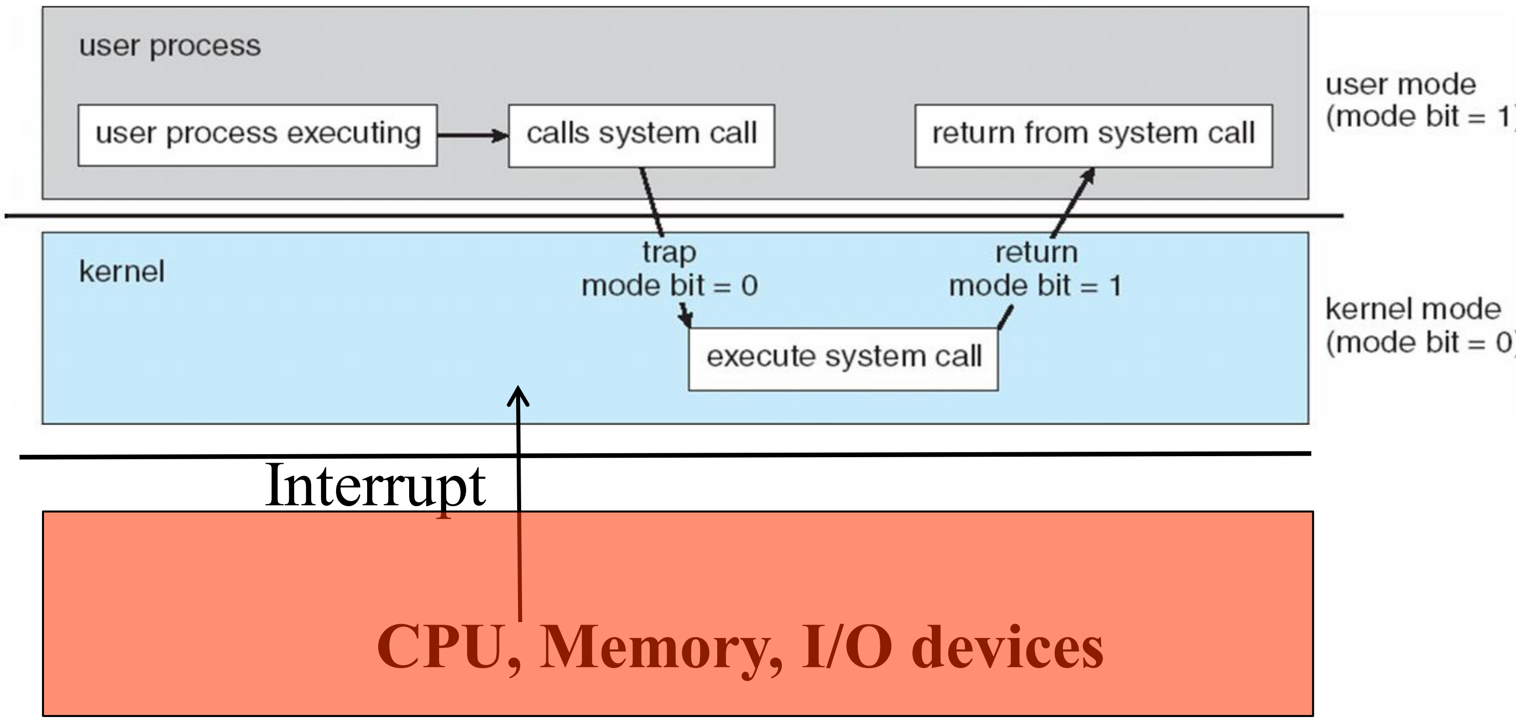
- I/O devices and the CPU can execute concurrently
  - Each device controller is in charge of a particular device type
  - Each device controller has a local buffer
  - CPU moves data from/to main memory to/from local buffers
- Device controller informs CPU that it has finished its operation by causing an ***interrupt***

***

Computer-System Operation (cont'd)

system calls

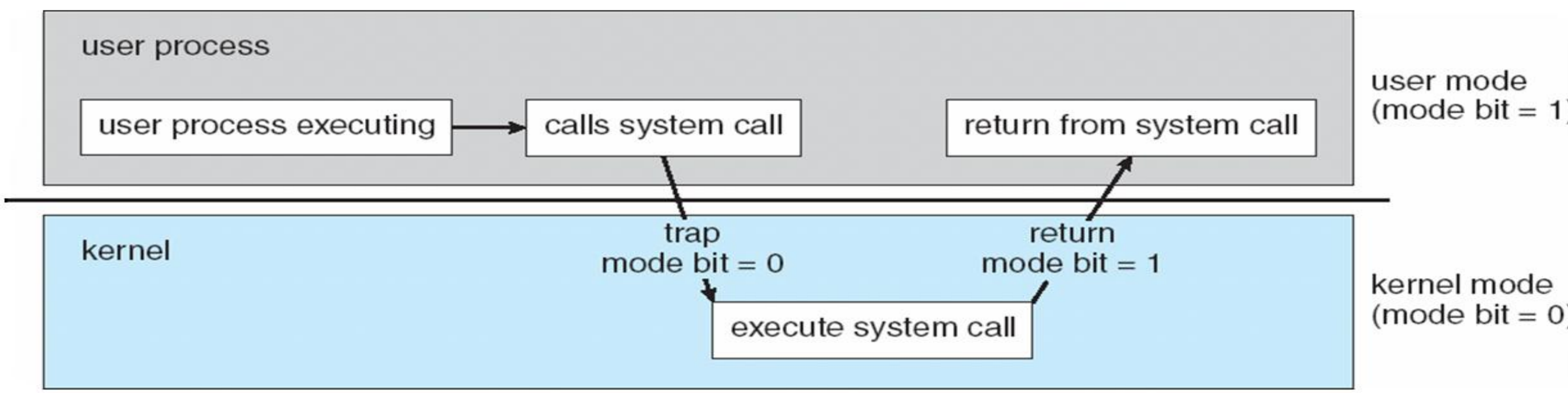
Interrupt



***

OS Interface: APIs and System Calls

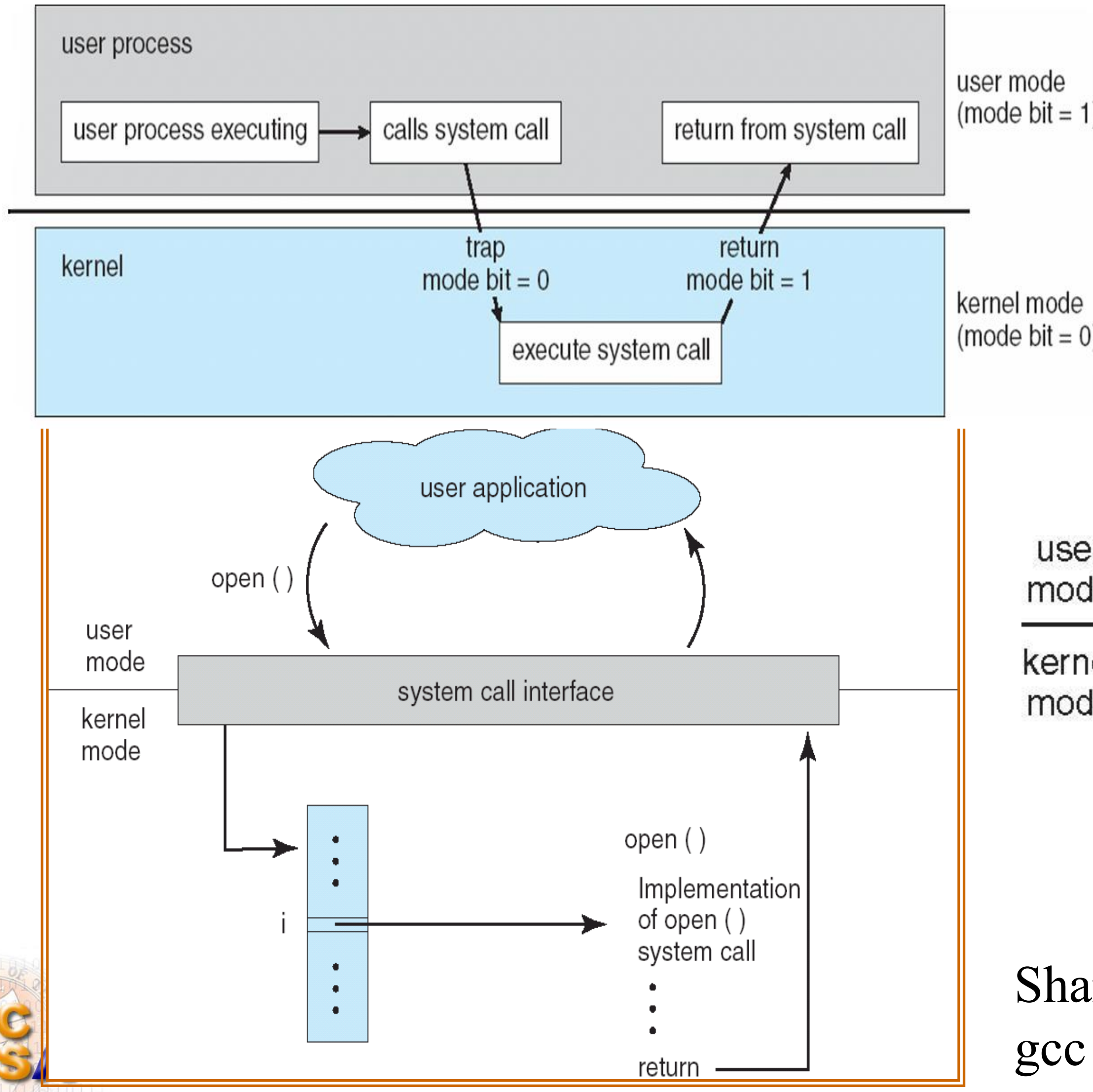
- For application programmers
- ***Application programming interface (API)***
  - The run-time support system (run-time libraries) provides a *system-call interface*, that intercepts function calls in the API and invokes the necessary system call within the operating system
- **System calls** provide the interface between a running program and the operating system.
  - Generally available in routines written in C and C++
  - Certain low-level tasks may have to be written using assembly language

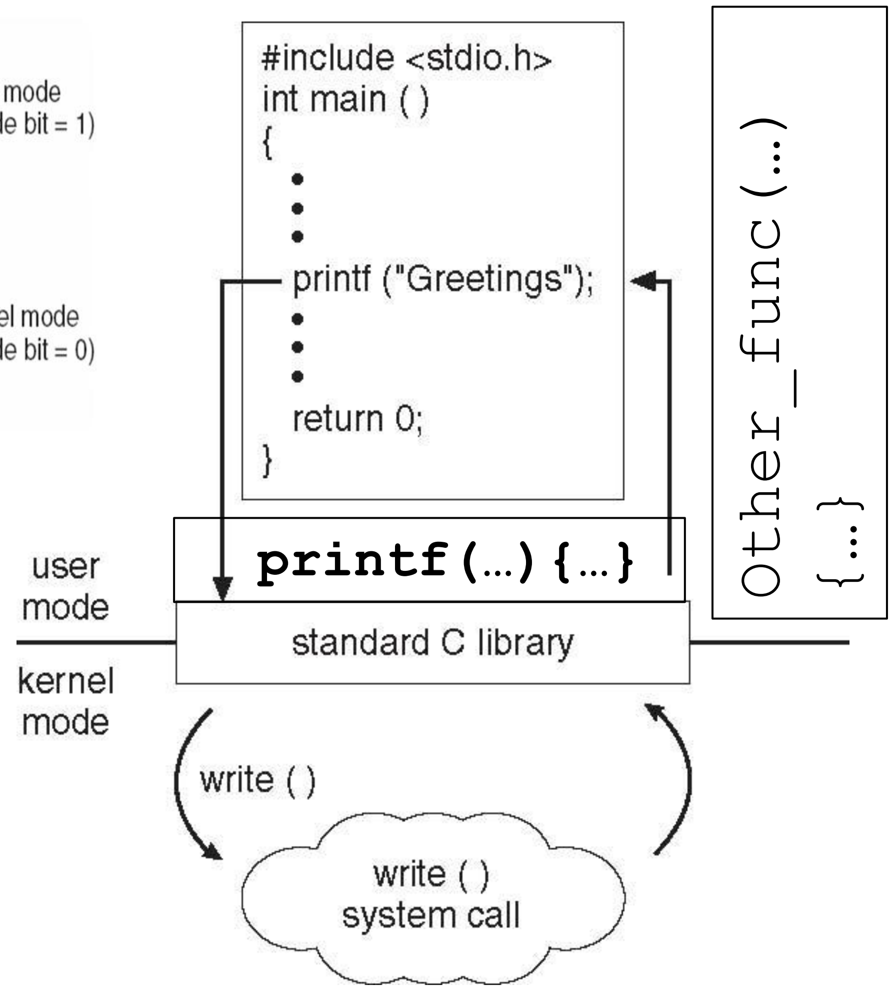


***

System Call vs. User/Library Function Call

write(…) vs. printf(….)



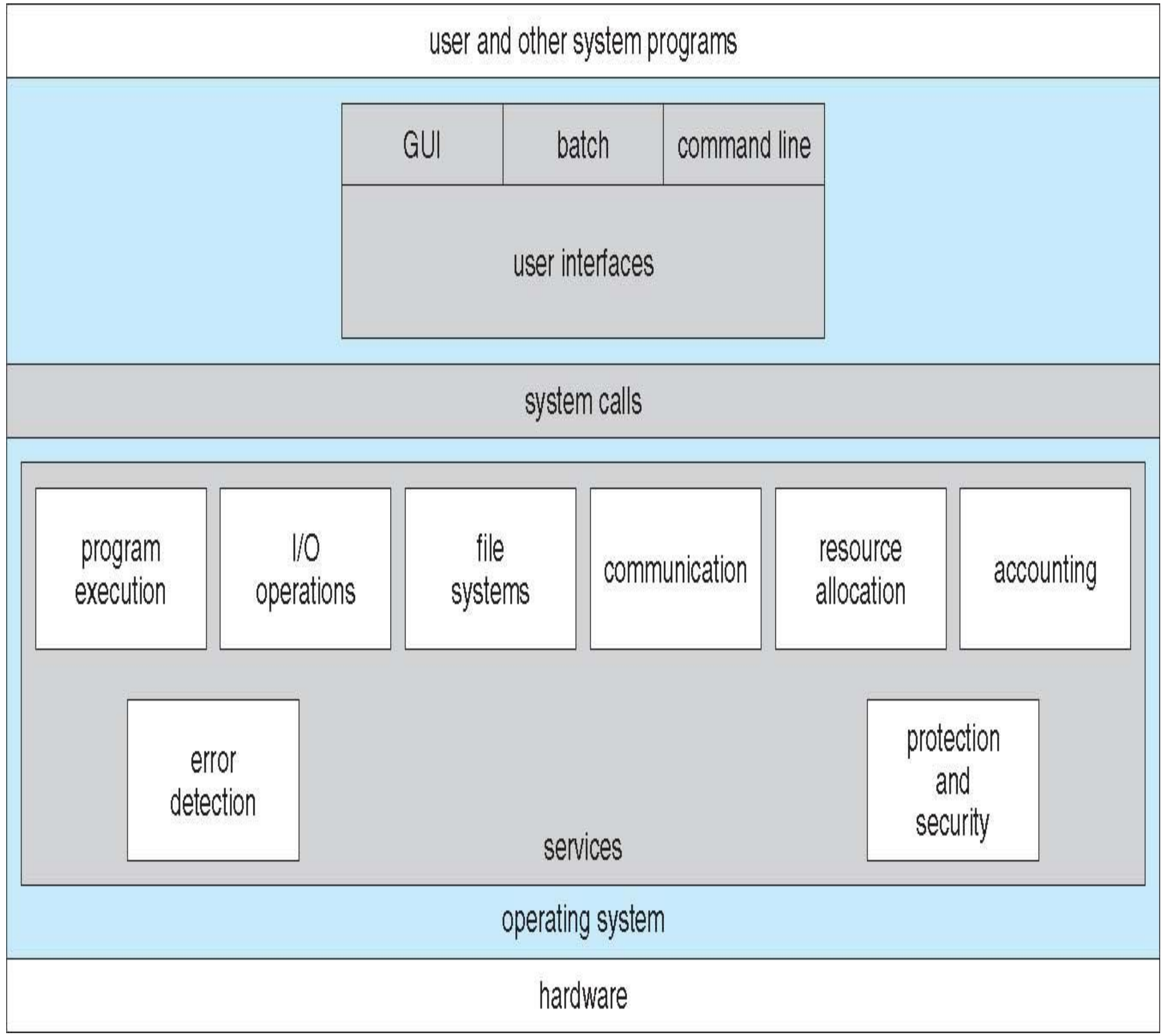


Shared library
gcc hello.c
gcc –static hello.c

How about making printf a system call? Why (not)?

Types of System Calls

- Process control
- File management
- I/O management
- Communications
- Accounting
- Protection



Examples: Major System Calls in Unix

Process management

Call	Description
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

File management

Call	Description
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

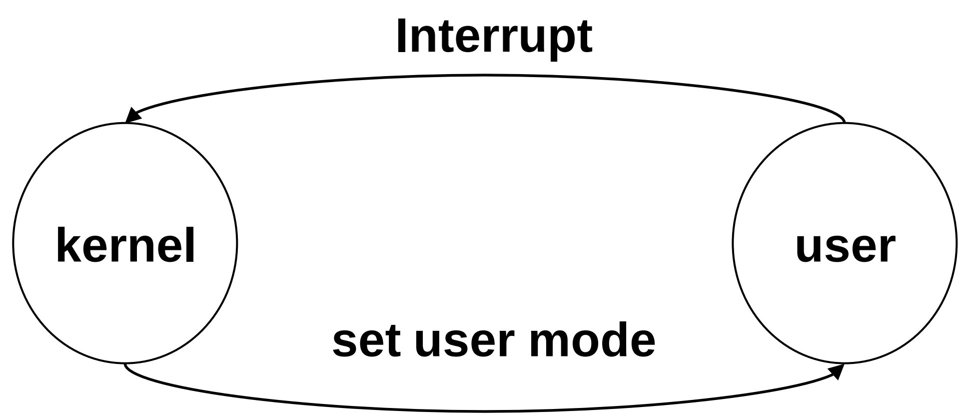
Examples of Windows and Unix System Calls

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

Operation Modes

- Hardware support (**mode bit**) to differentiate between at least two modes of operations
  - ***User mode*** – execution done on behalf of a user
  - ***Kernel mode*** (also *monitor mode* or *system mode or privileged mode*) – executing on behalf of operating system
- E.g., **interrupts**: → switches to monitor mode.

Privileged instructions **can be issued only in kernel mode.**



***

System call control flow - Linux

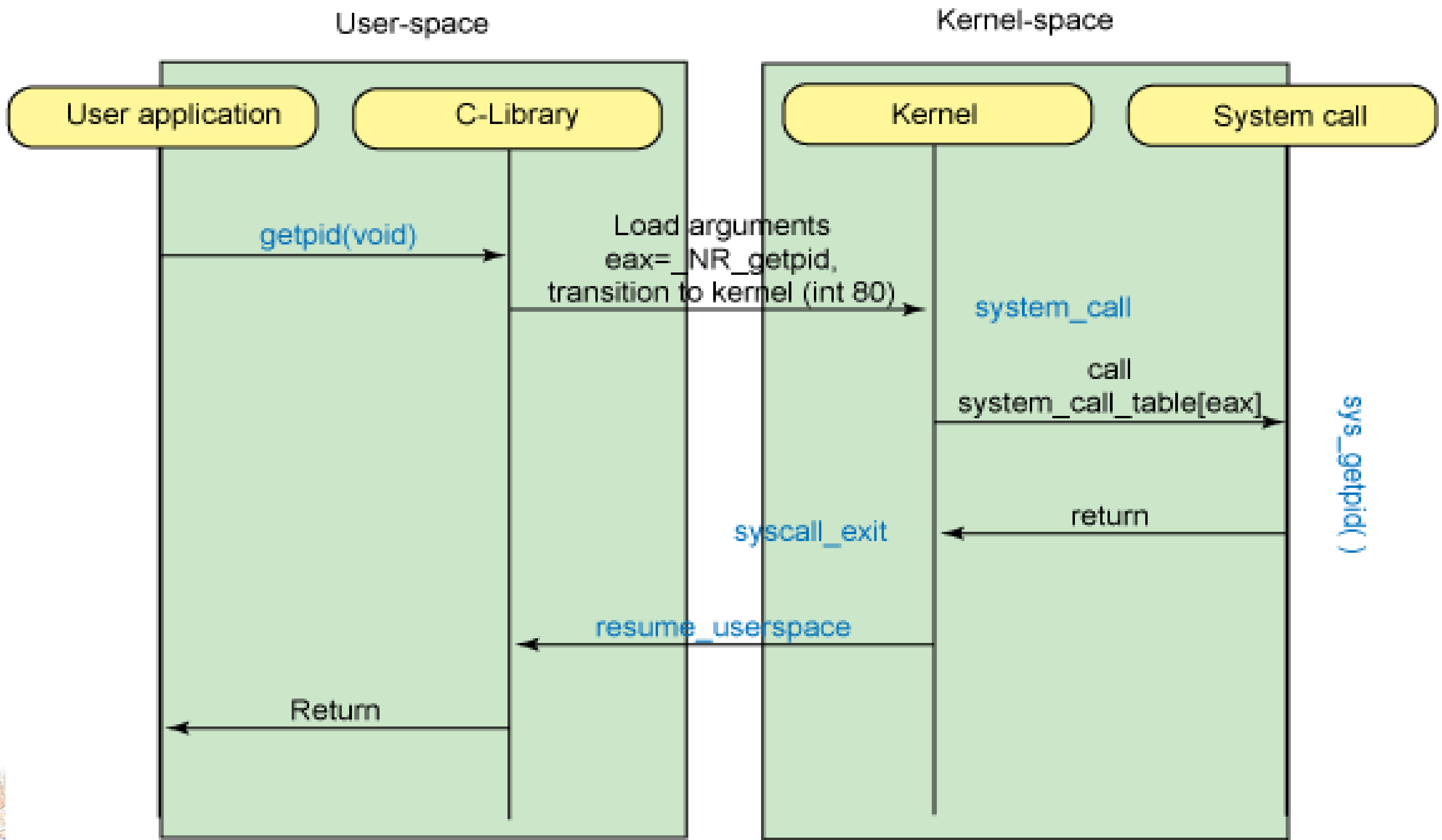
- User application calls a user-level library routine (`gettimeofday()`, `read()`, `exec()`, etc.)
- Invokes system call through stub, which specifies the system call number. From `unistd.h`:

```
#define __NR_getpid 172
__SYSCALL(__NR_getpid, sys_getpid)
```

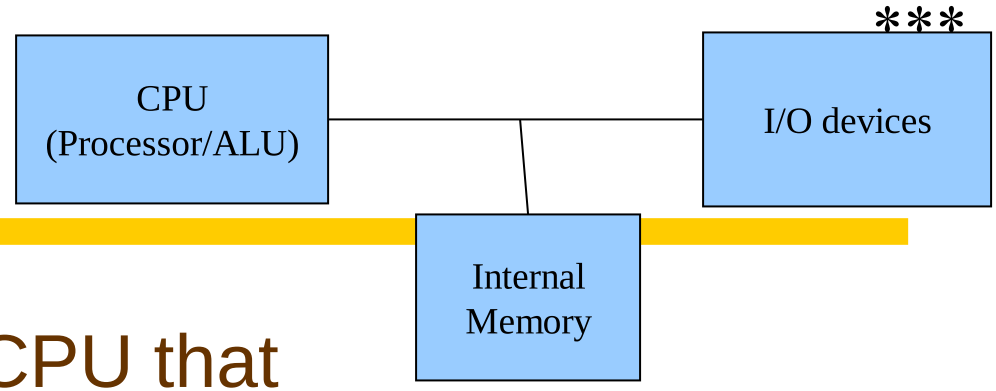
- This generally causes a **software interrupt**, trapping to kernel
- Kernel looks up **system call number** in syscall table, calls appropriate function
- Function executes and returns to interrupt handler, which returns the result to the user space process

***

System call control flow - Linux



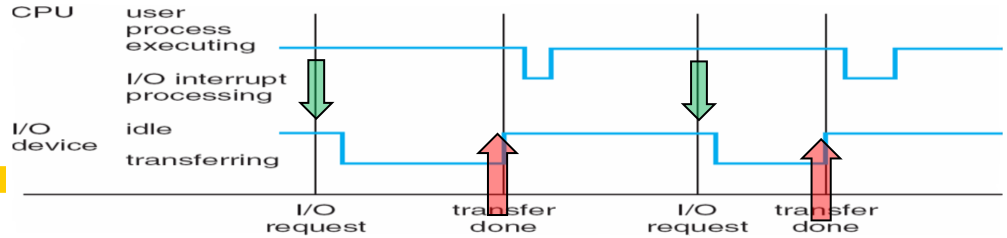
Interrupt Mechanisms



- Device controller informs CPU that it has finished its operation by causing an ***interrupt***
- Save the current "process state"
- Interrupt transfers control to the interrupt service routine (ISR) generally through ***interrupt vector*** containing the addresses of all the service routines.
- **ISR**: 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.

***

Interrupt Handling



1. The interrupt is issued
2. Processor finishes execution of current instruction
3. Processor signals acknowledgement of interrupt
4. Processor pushes PSW(*Program Status Word)* and PC to 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

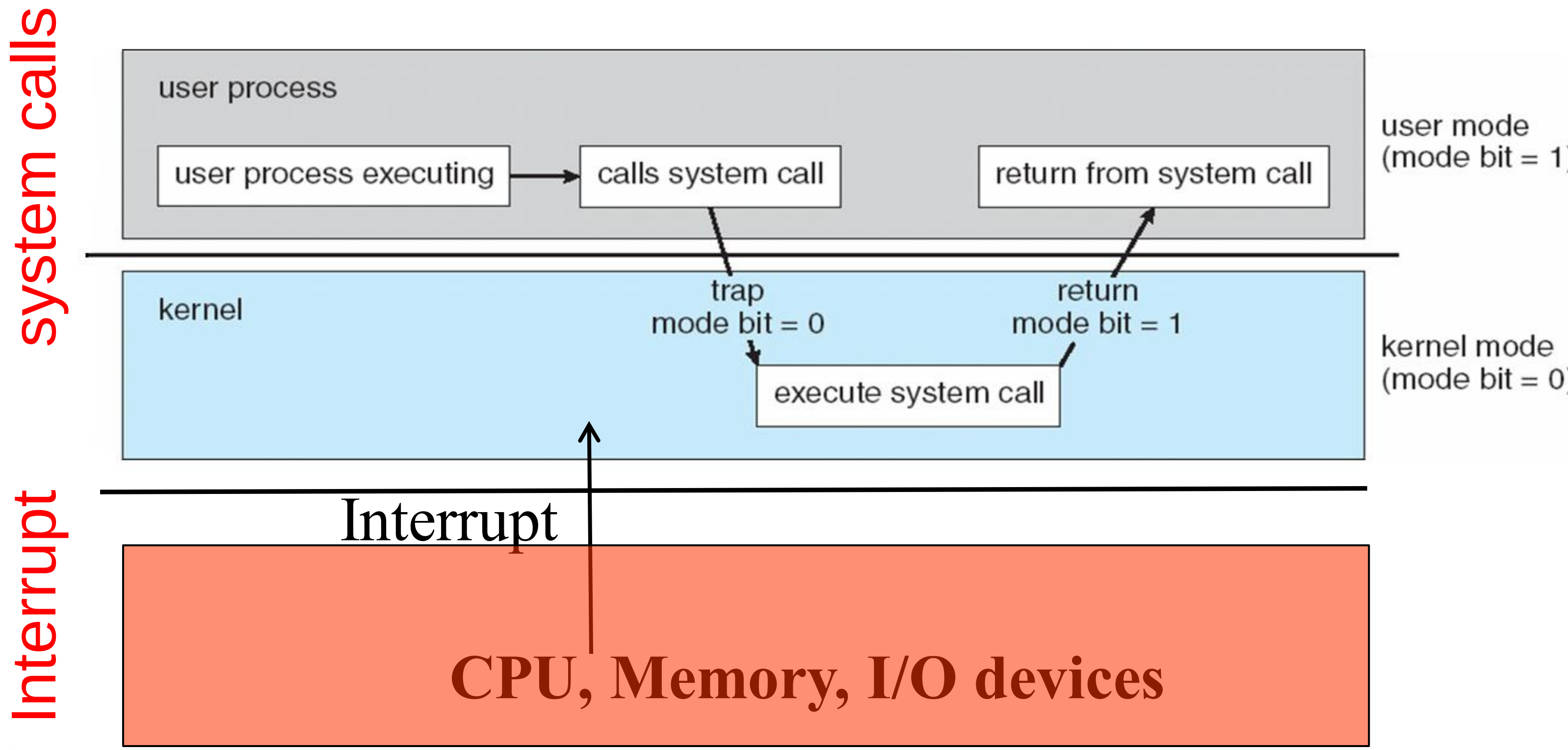
What if another interrupt occurs during interrupt processing?

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, like scheduling
- **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
  - User request for an operating system service (e.g., system calls)
  - Runtime errors

Retrieval Practice

What are the two key mechanisms to interact with the kernel, and how do they work?



Retrieval Practice

What are the two key differences between system calls and user/library function calls?

e.g., write vs. printf

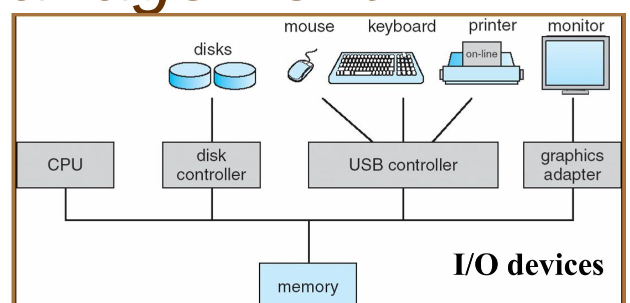
Lecture Outline

- Evolution of Computer Systems and OS Concepts
- Operating System: what is it?
- OS as a resource manager
  - How does OS provide service? – interrupt/system calls
- OS Structures and basic components
  - Process/memory/IO device managers
- Different types/variations of Systems/OS
  - Parallel/distributed/real-time/embedded OS etc.

Components in Operating System

- ***Process/thread Management***
  - ***CPU (processors): most precious resource***
- ***Memory Management***
  - ***Main memory***
- **File Management → data /program**
- Secondary-Storage Management → disk
- I/O System Management → I/O devices
- *Protection and Security → access management*

I/O



***

Process Management

!!! MORE LATER !!!

- A ***process*** is a *program* in execution (active),
  - Dynamic concept, represented by ***process control block***
- A process needs resources: execution environment
  - including CPU time, registers, memory, files, and I/O devices to accomplish its task
- OS provides mechanism to
  - Create/delete processes
  - Run/Suspend/resume processes (scheduling/signal)
  - Process communication and synchronization
  - Deadlock handling

Main Memory Management

!!! MORE LATER !!!

***

- The main memory is
  - a large array of words/bytes, each with its own address
  - a **volatile** storage device: content lost when power off
- 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 becomes available
  - Allocate and de-allocate memory space as needed

***

File Management

!!! MORE LATER !!!

- A file is a collection of related information (logic unit)
  - Format is defined by its creator.
- Represent programs (source/object forms) and data
- 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

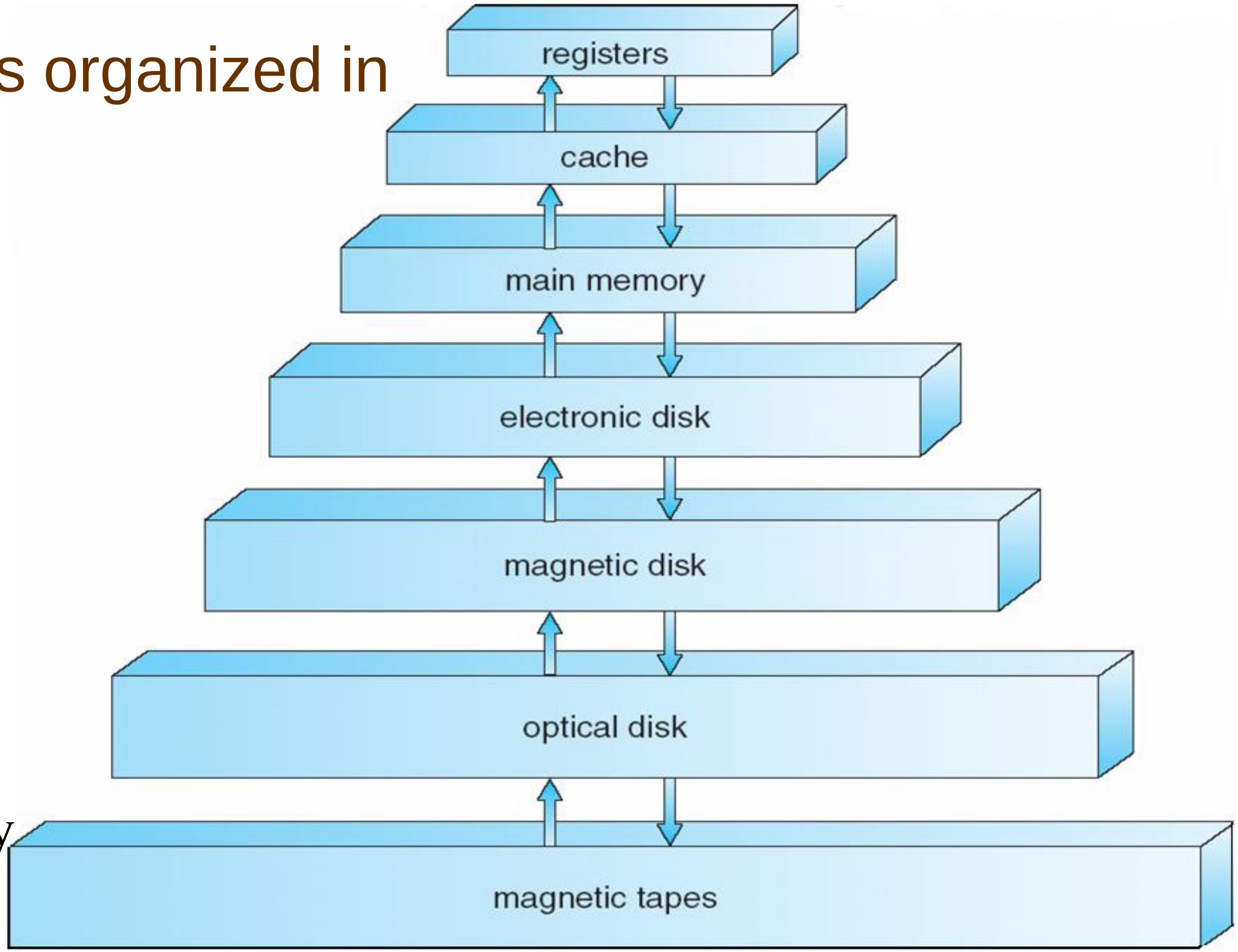
Secondary-Storage Management

- 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

Storage Hierarchy

- Storage systems organized in hierarchy
  - Speed
  - Cost
  - Volatility

Caching – copying information into faster storage system; main memory can be viewed as a last *cache* for secondary storage



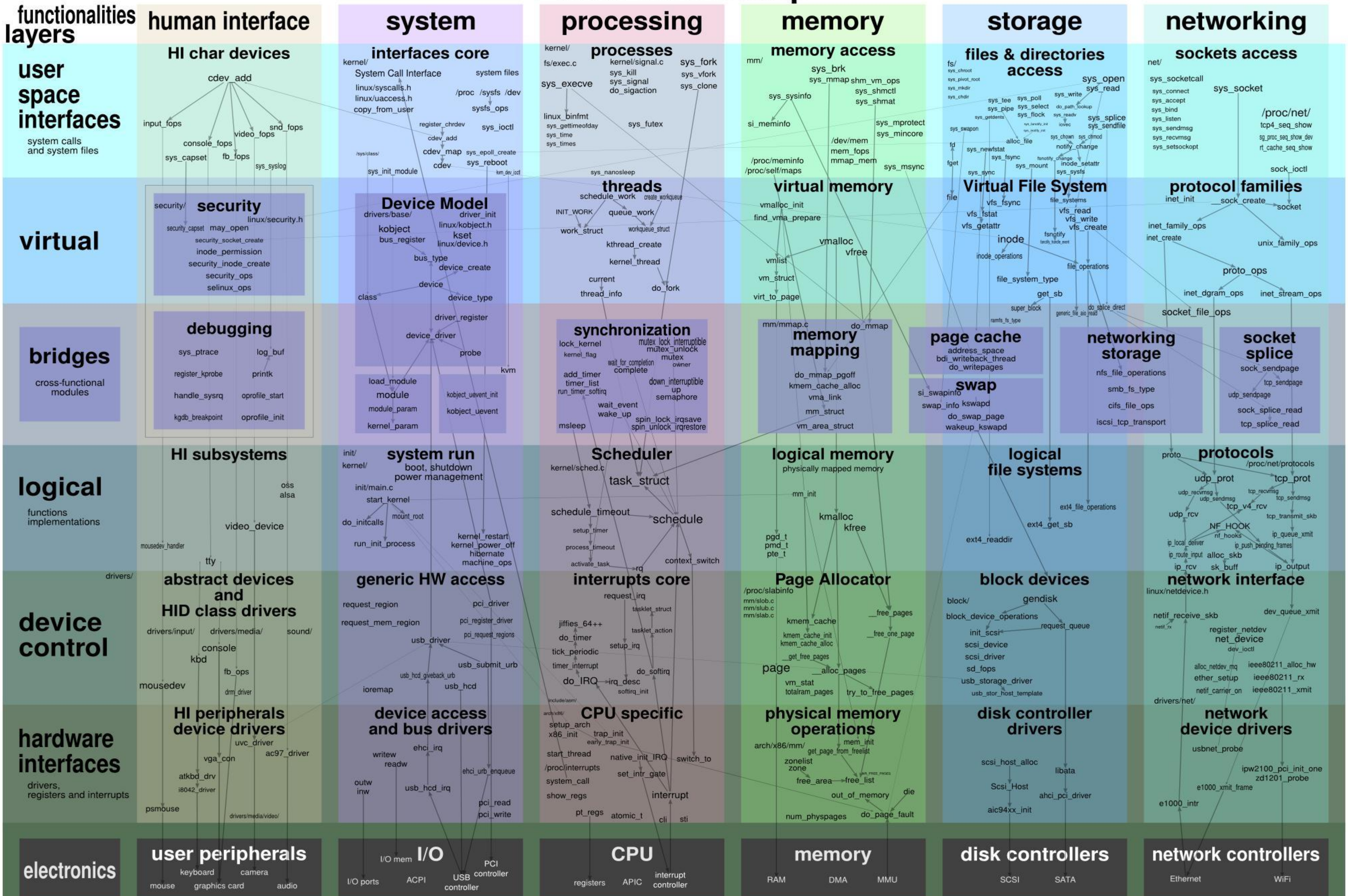
Caching

- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- Information in use copied from slower to faster storage temporarily
- Faster storage (cache) checked first to determine if information is there
  - If it is, information used directly from the cache (fast)
  - If not, data copied to cache and used there
- Cache smaller than storage being cached
  - Cache management important design problem
  - Cache size and replacement policy

I/O System Management

- The Operating System will hide the peculiarities of specific I/O 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

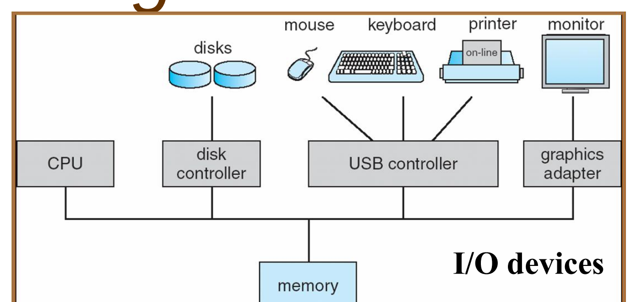
Linux kernel map



Retrieval exercise: What are the Key Components/ modules in Operating System?

- ***Process/thread Management***
  - ***CPU (processors): most precious resource***
- ***Memory Management***
  - ***Main memory***
- **File Management → data /program**
- Secondary-Storage Management → disk
- I/O System Management → I/O devices
- *Protection and Security → access management*

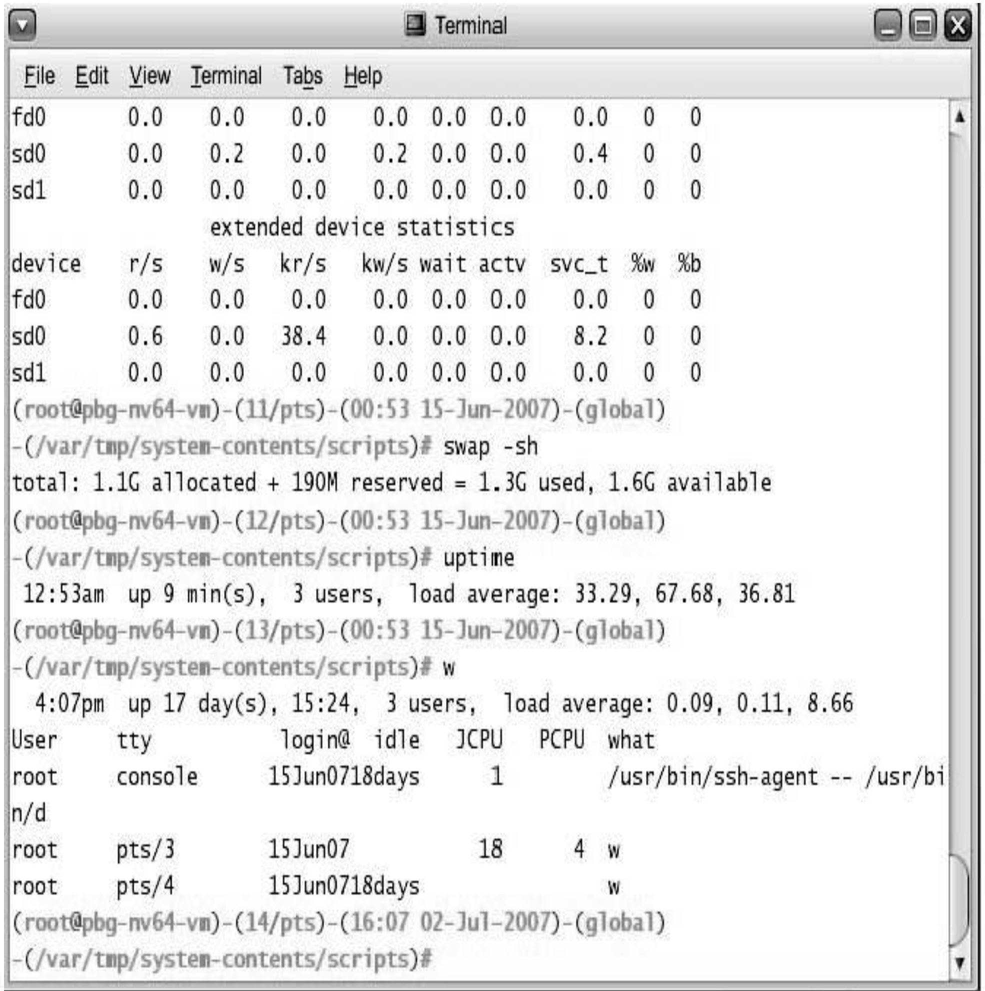
I/O



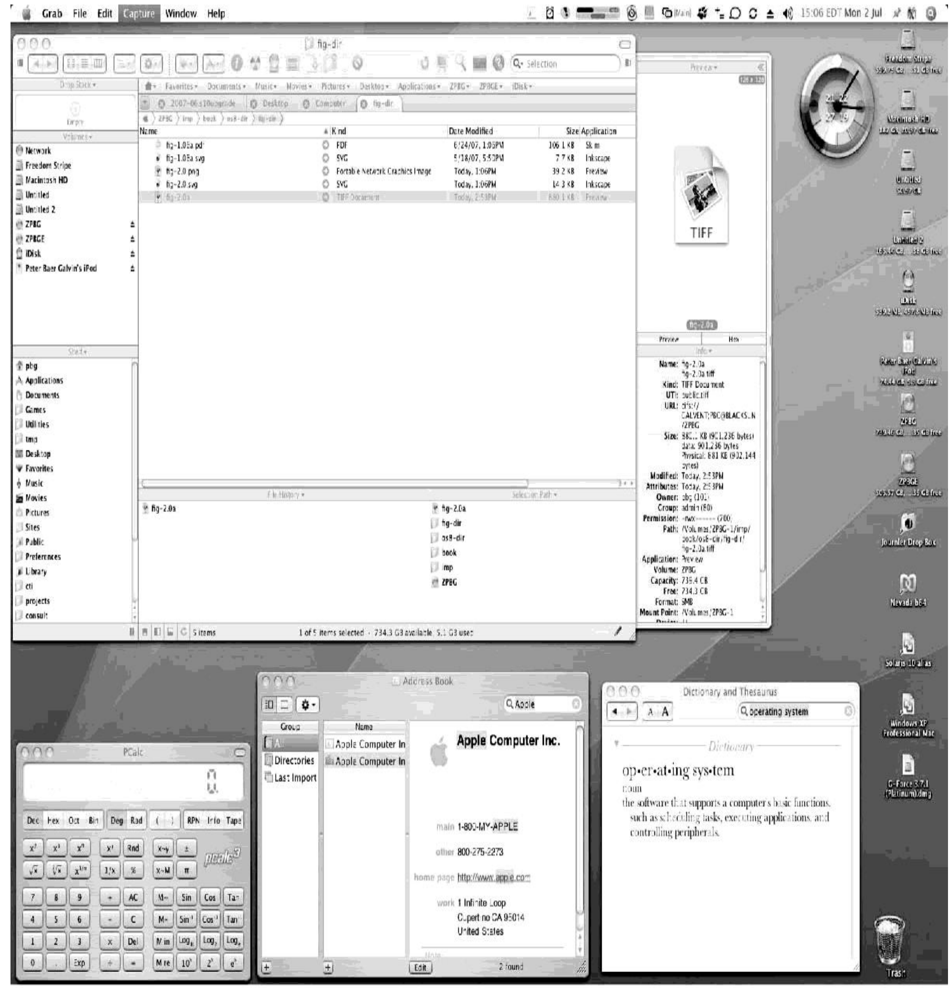
OS Interface: Shell and GUI

- For both ***programmers*** and ***end-users***
- Two main approaches for both
  - *Command-line 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: click-and-play
- Some systems allow both (e.g. X-Windows in Unix)

OS Interfaces



Command Line



The Mac OS X GUI

System Programs

- System programs provide a convenient environment for program development and execution. They can be divided into:
  - File manipulation
  - Status information
  - File modification
  - Programming language support
  - Program loading and execution
  - Communications
  - Application programs
- Most users' view of the operating system is defined by system programs, not the actual system calls

System Programs

- Provide a convenient environment for program development and execution
  - Some of them are simply user interfaces to system calls; others are considerably more complex
- File management - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
- Status information
  - Some ask the system for info - date, time, amount of available memory, disk space, number of users
  - Others provide detailed performance, logging, and debugging information
  - Typically, these programs format and print the output to the terminal or other output devices
  - Some systems implement a registry - used to store and retrieve configuration information

System Programs (Cont.)

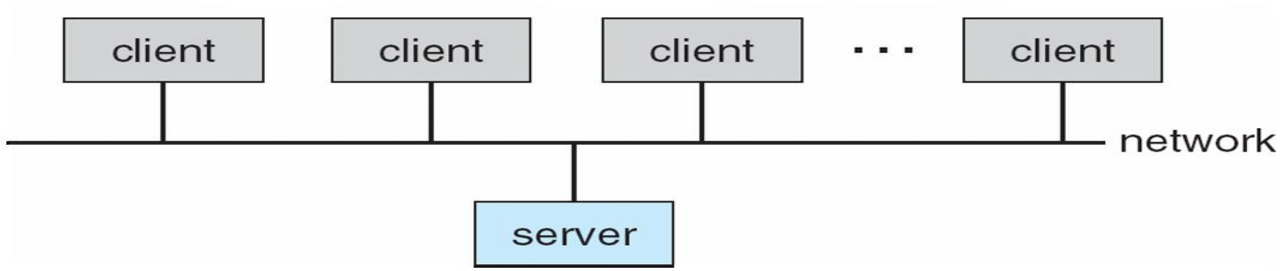
- File modification
  - Text editors to create and modify files
  - Special commands to search contents of files or perform transformations of the text
- Programming-language support - Compilers, assemblers, debuggers and interpreters sometimes provided
- Program loading and execution- Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
- Communications - Provide the mechanism for creating virtual connections among processes, users, and computer systems
  - Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

Lecture Outline

- Evolution of Computer Systems and OS Concepts
- Operating System: what is it?
- OS as a resource manager
  - How does OS provide service? – interrupt/system calls
- OS Structures and basic components
  - Process/memory/IO device managers
- Different types/variations of Systems/OS
  - Parallel/distributed/real-time/embedded OS etc.

Distributed Systems

- *Loosely coupled system* – each processor has its own local memory; processors communicate with one another through various **communications** lines
- Advantages of distributed systems.
  - **Resource Sharing**
  - Computation speed up – load sharing
  - Reliability
  - Communications



Peer-to-Peer Computing Systems

- Another model of distributed system
- P2P does not distinguish clients and servers
  - Instead all nodes are considered peers
  - May each act as client, server or both
  - Node must join P2P network
    - Registers its service with central lookup service on network, or
    - Broadcast request for service and respond to requests for service via **discovery protocol**
  - Examples include *Napster* and *Gnutella*

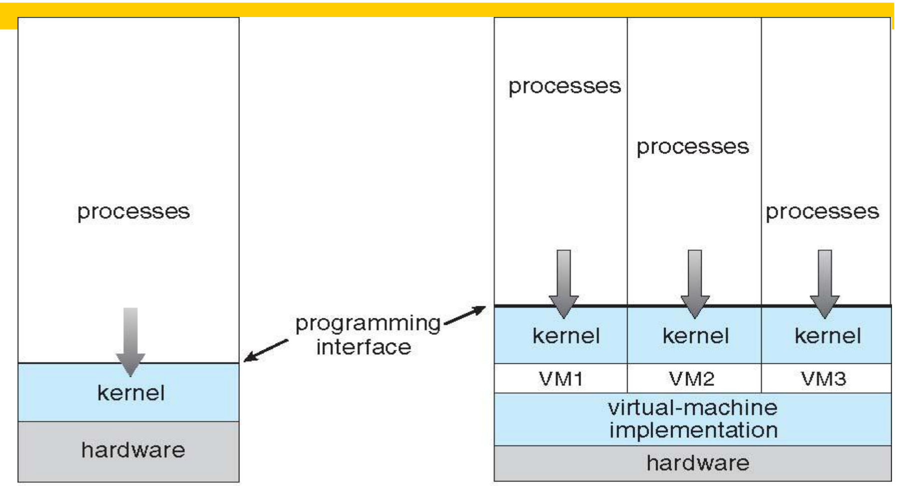
Special Purpose Systems

- A **real-time** system is used when **strict time requirements** have been placed on the operation of a processor or the flow of data
  - *Hard* real-time: critical tasks <u>must</u> be completed on time
  - *Soft* real-time: no absolute timing guarantees (e.g. "best-effort scheduling"); multimedia applications;
- An **embedded** system is a component of a more complex system
  - Control of a nuclear plant or Missile guidance
  - Control of home and car appliances (microwave oven, DVD players, car engines, …)
- Example: VxWorks and eCos; Android and iOS

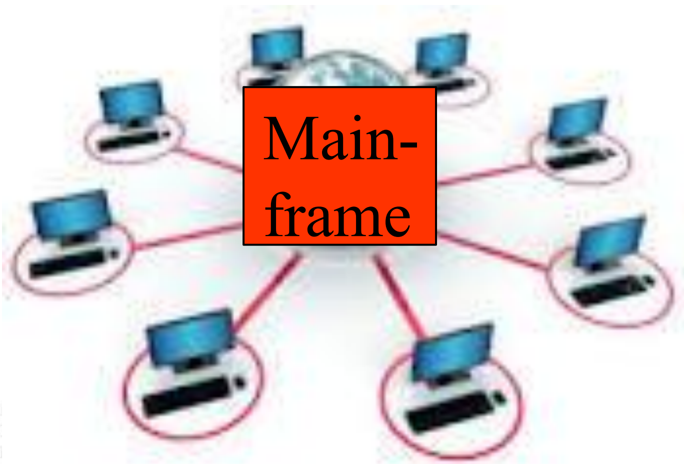
***

Virtualization and Cloud Computing

- **Virtualization**: Run an OS as Application in another OS
- Emulation, Interpretation
- **Cloud Computing**
  - SaaS, PaaS, IaaS, XaaS…

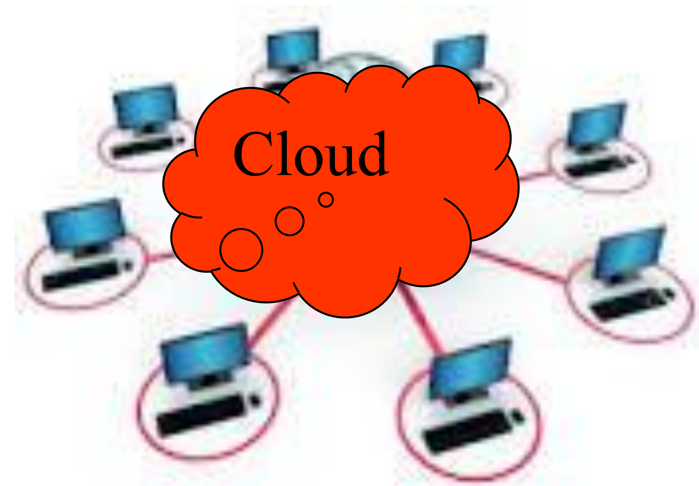


(a) (b)



Then…

Along the way…



Now…