Introduction to Operating Systems

What is an OS (component, functional, service views)? Dual mode operation. System call vs. function call. Coordination between CPU and IO (interrupt-driven design). Computer system and OS structures. Basic concept of process and concurrency. Process vs. program. Memory/storage hierarchy and caching.

OS1: 25/1/2016
Textbook (SGG): Ch. 1.1-1.6,1.8.3,1.13.3

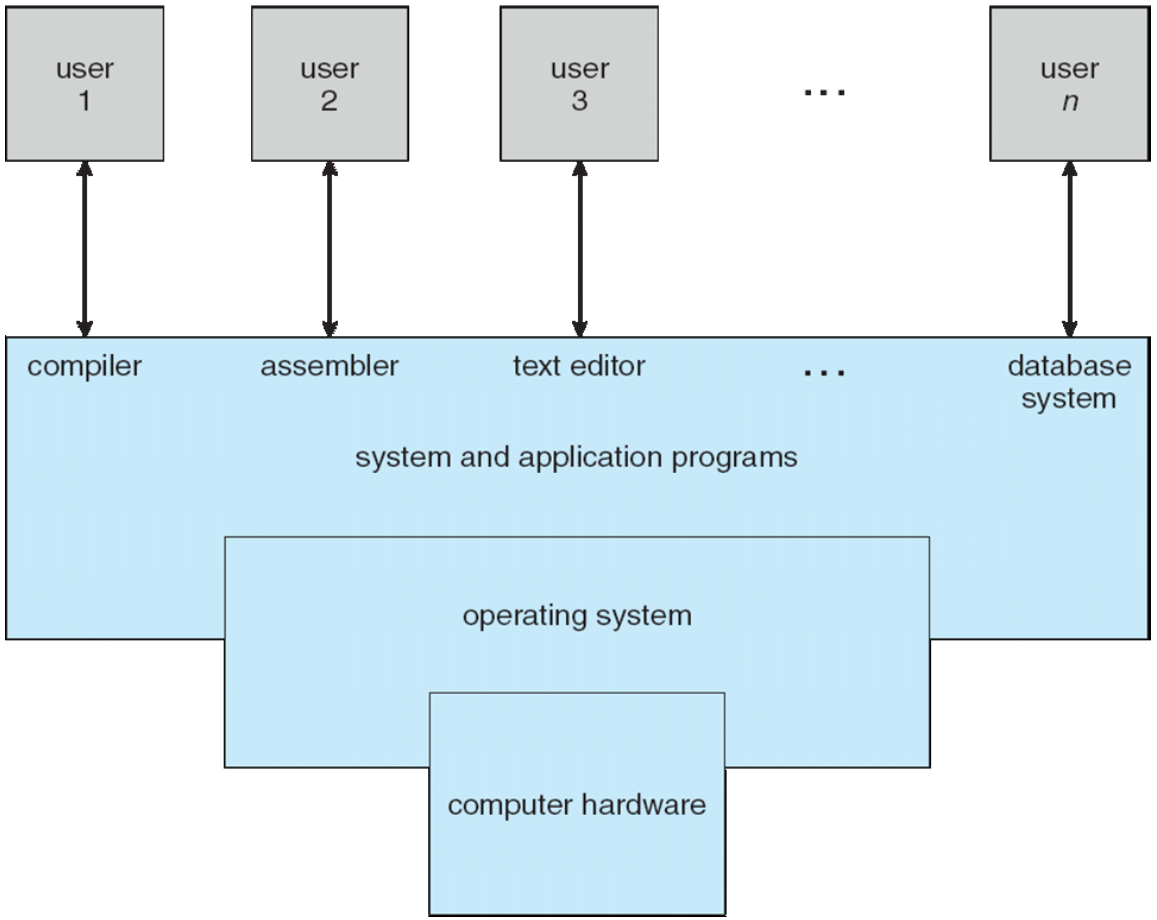
What is an Operating System?

- An OS is just a program – like any other programs that you write.
- But it's a special program.
  - Acts as an intermediary between a user of a computer and the computer hardware
  - Used a lot by all the users.
- Operating system 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

OS is part of computer system

- Computer system can be divided into four components:
  - Hardware – provides basic computing resources
    - CPU, memory, I/O devices
  - Operating system
    - Controls and coordinates use of hardware among various applications and users
  - Application programs – define the ways in which the system resources are used to solve the computing problems of the users
    - Word processors, compilers, web browsers, database systems, video games
  - Users
    - People, machines, other computers

Four Components of a Computer System



Roles of Operating System

- OS is a **resource allocator**
  - Manages all resources
  - 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
- Like *government* of a country?

What's the OS? What's the "kernel"?

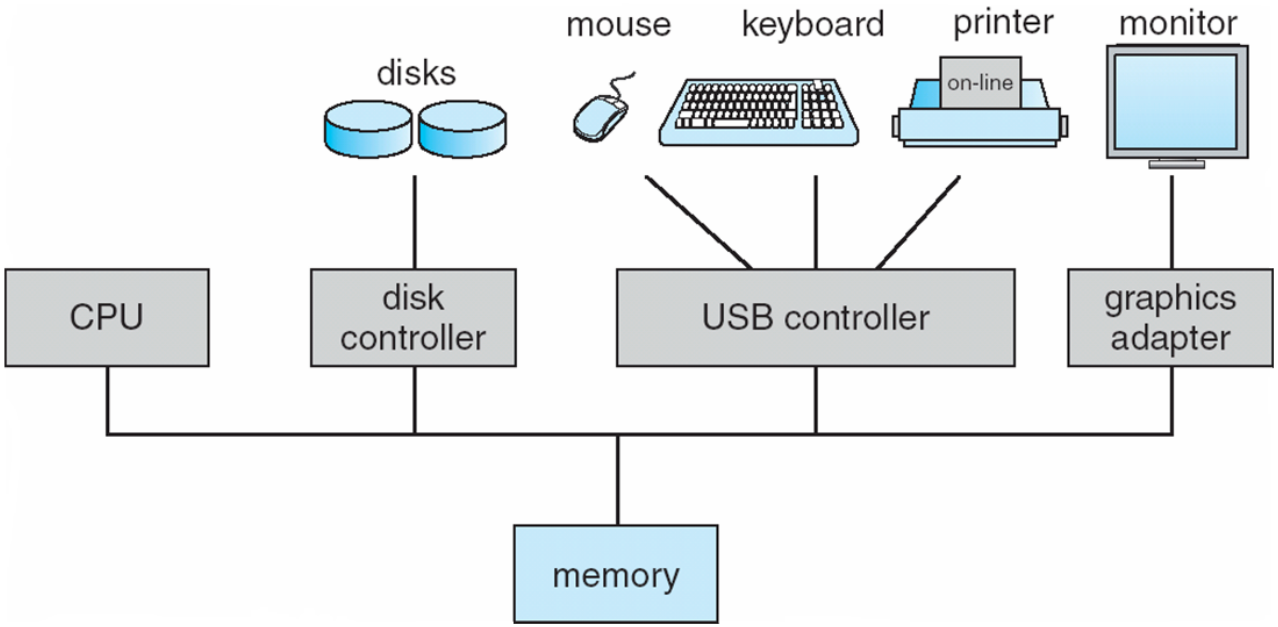
- No universally accepted definition of what constitutes OS.
- Some people think of it as "Everything a vendor ships when you order an operating system" – could be useful first approximation
  - But varies wildly
- More precise definition: "The one program running at all times on the computer" is the **kernel**. Everything else (even shipped by OS vendor) is either a system program or an application program.
- The kernel is *unique*.
  - Although it's just a program that someone wrote, it *runs* with special privileges – it can do what normal user code cannot do.
  - Hardware support required – CPU has (at least) dual mode operation: user (unprivileged) mode vs. kernel (privileged) mode.
  - Why are these privileges critical for kernel to fulfill the OS roles?

How OS starts to run?

- **Bootstrap program** is loaded at power-up or reboot of computer
  - Typically stored in ROM or EPROM, generally known as **firmware**
  - Initializes all aspects of system
  - Loads operating system kernel and starts execution

Computer System Organization

- Computer-system operation:
  - One or more CPUs, device controllers connect through common bus providing access to shared memory.
  - Concurrent execution of CPUs and devices competing for memory cycles.



Computer-System Operation

- CPU is not the only component capable of running code or starting activities. IO devices/controllers can also act autonomously.
- 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.
- I/O is from the device to local buffer of controller.
- CPU/controller needs coordination. E.g., device controller informs CPU that it has finished its operation by causing an *interrupt.*

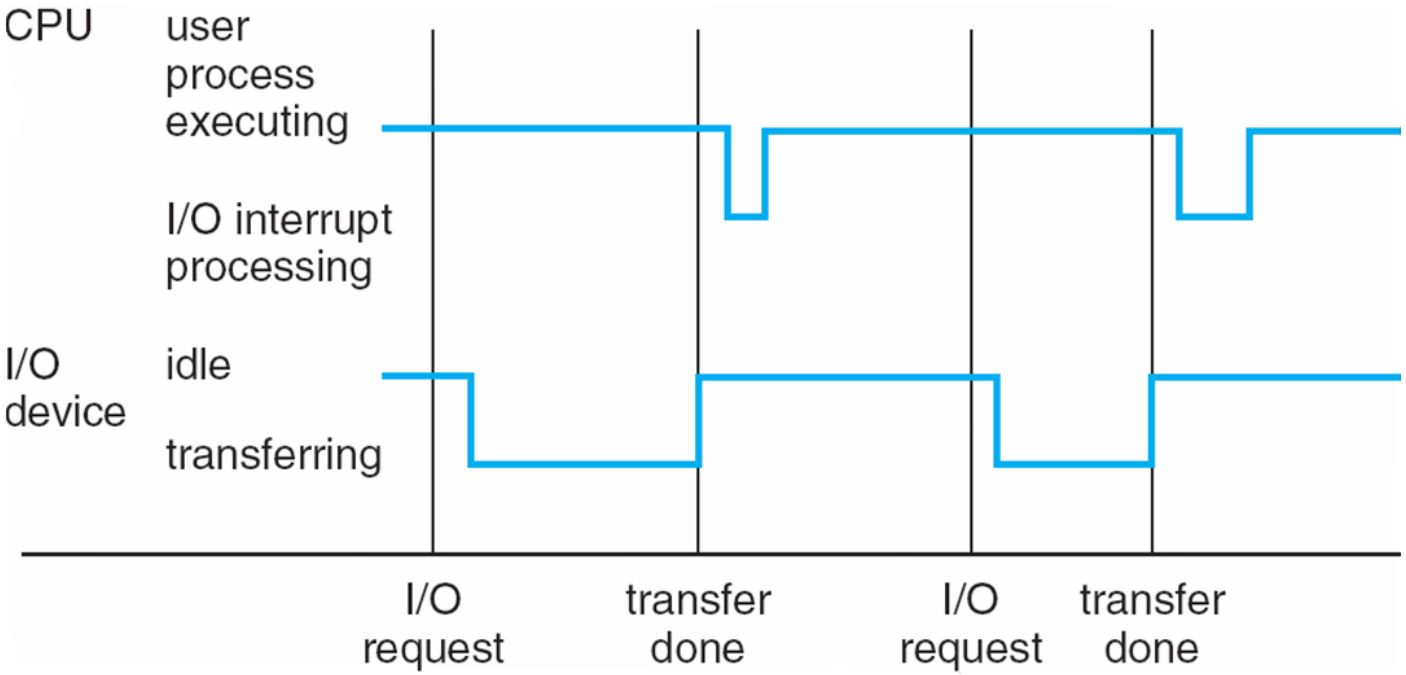
More about Interrupts

- Interrupt transfers control to the interrupt service routine generally, through the **interrupt vector**, which contains the addresses of all the service routines.
- Interrupt architecture must save the address of the interrupted instruction.
- Incoming interrupts are *disabled* while another interrupt (of same or higher priority) is being processed to prevent a *lost interrupt* or *reentrancy* problems.
- A *trap* is a software-generated interrupt caused either by an error or a request by user code. Latter allows a user program to invoke an OS function and run it in kernel mode. Hence, entry points into kernel are carefully controlled – Why? And why is this important?
- Modern operating system is **interrupt driven**.

Interrupt Handling

- The operating system preserves the state of the CPU by storing registers and the program counter.
- Determines which type of interrupt has occurred:
  - **polling**
  - **vectored** interrupt system
- Separate segments of code determine what action should be taken for each type of interrupt.

Interrupt Timeline



Quiz 1.1

- When a computer boots up, does the CPU start executing in user mode or kernel mode?
- A user process has been running in user mode. What two types of events will cause the CPU to change its execution mode from user to kernel?
- Assume that the OS has an interrupt table (or array) called ISR, so that when the CPU is interrupted by a device with vector number *n*, the CPU will execute the interrupt handler whose address is stored in ISR[*n*]. When an interrupt occurs, why is it not possible for a user to cause the CPU to run any (possibly buggy or malicious) code other than the intended handler?

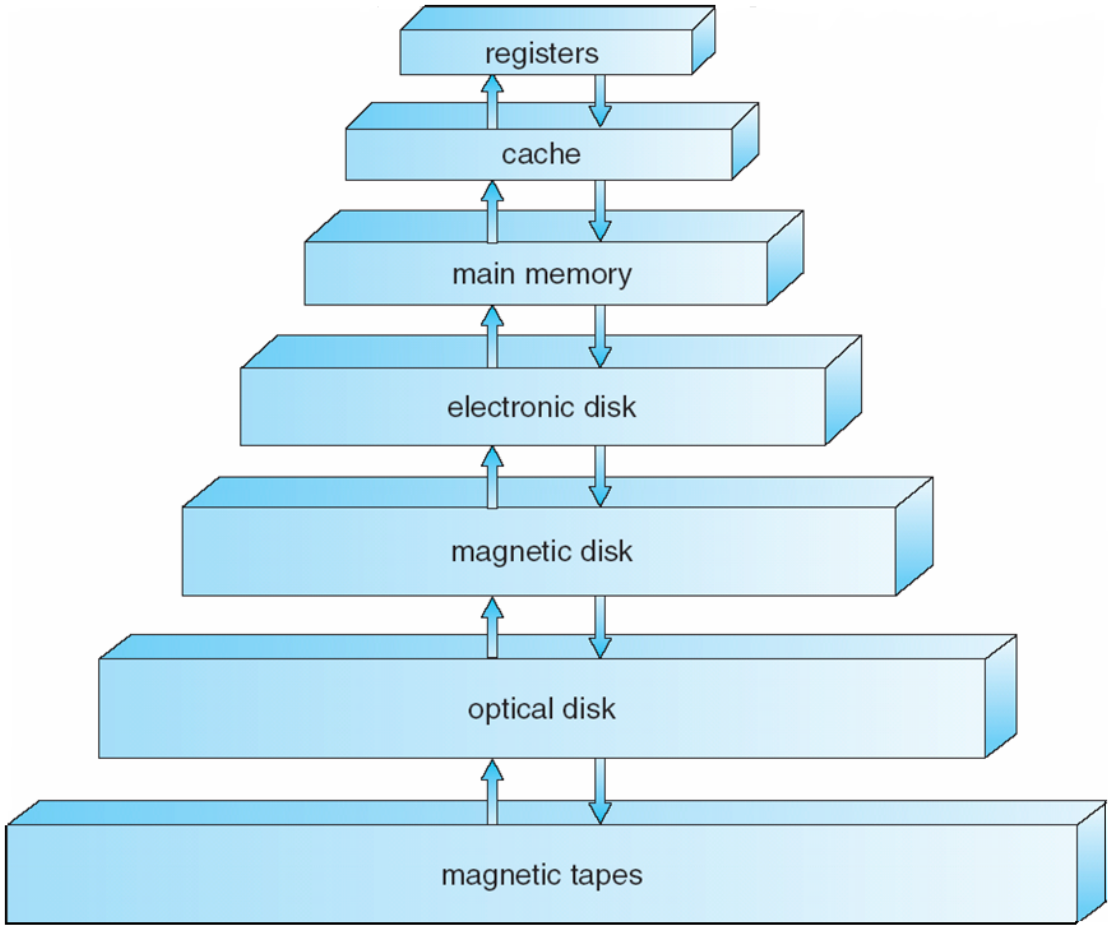
Storage Structure

- Registers – small number of them (expensive), resides in CPU
- Cache – hold much more data than registers, but still pretty small
- Main memory – only large storage media that the CPU can access directly.
- 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.

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

Storage-Device Hierarchy

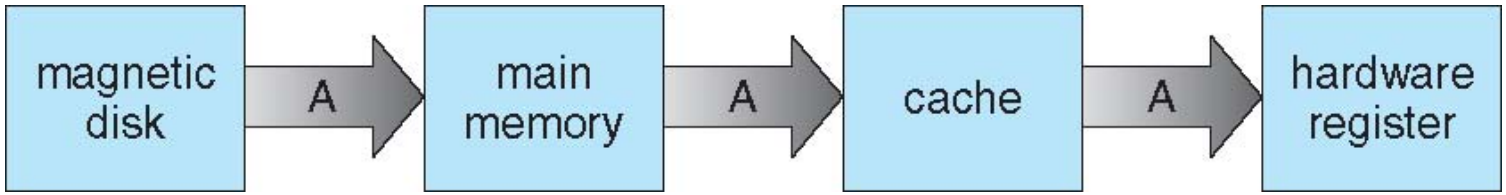


Caching

- 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

Migration of Integer A from Disk to Register

- Multitasking environments must be careful to use most recent value, no matter where it is stored in the storage hierarchy.



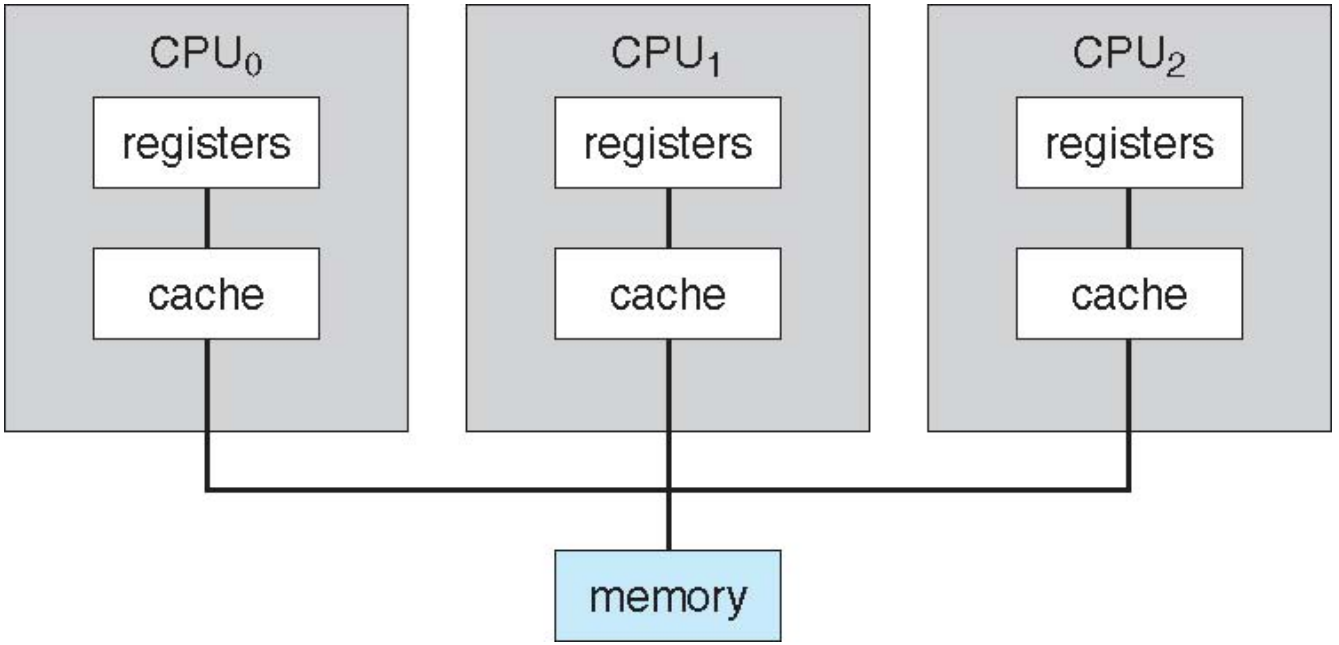
- Multiprocessor environment must provide cache *coherency* in hardware such that all CPUs have the most recent value in their cache.

Performance of Various Levels of Storage

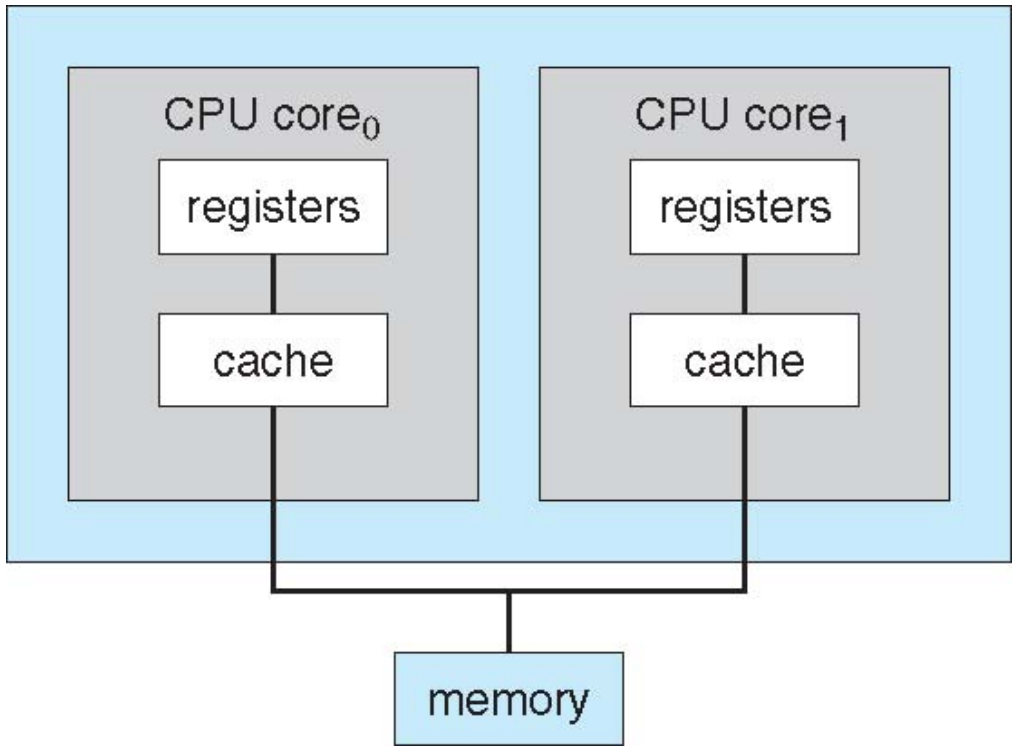
- Movement between levels of storage hierarchy can be explicit or implicit

Level	1	2	3	4
Name	registers	cache	main memory	disk storage
Typical size	< 1 KB	> 16 MB	> 16 GB	> 100 GB
Implementation technology	custom memory with multiple ports, CMOS	on-chip or off-chip CMOS SRAM	CMOS DRAM	magnetic disk
Access time (ns)	0.25 – 0.5	0.5 – 25	80 – 250	5,000.000
Bandwidth (MB/sec)	20,000 – 100,000	5000 – 10,000	1000 – 5000	20 – 150
Managed by	compiler	hardware	operating system	operating system
Backed by	cache	main memory	disk	CD or tape

Symmetric Multiprocessing Architecture



A Dual-Core Design



Clustered Systems

- Like multiprocessor systems, but multiple systems working together
  - Usually sharing storage via a **storage-area network (SAN)**
  - Provides a **high-availability** service which survives failures
    - **Asymmetric clustering** has one machine in hot-standby mode
    - **Symmetric clustering** has multiple nodes running applications, monitoring each other
  - Some clusters are for **high-performance computing (HPC)**
    - Applications must be written to use **parallelization**

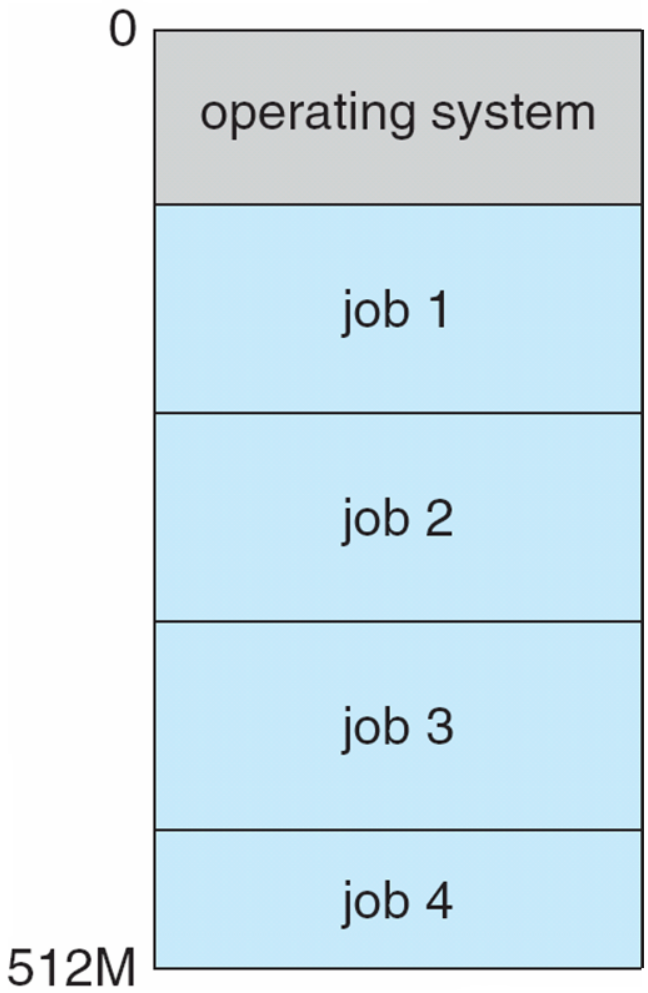
Operating System Structure

- **Multiprogramming** needed for efficiency
  - Single user cannot keep CPU and I/O devices busy at all times
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute
  - A subset of total jobs in system is kept in memory
  - One job selected and run via **job scheduling**
  - When it has to wait (for I/O for example), OS switches to another job

Operating System Structure (Cont.)

- **Timesharing (multitasking)** is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing.
  - **Response time** should be < 1 second
  - Each user has at least one program executing in memory ⇨**process**
    - A process is a *running program* – how are the two different?
  - If several jobs ready to run at the same time ⇨ **CPU scheduling**
  - If processes don't fit in memory, **swapping** moves them in and out to run
  - **Virtual memory** allows execution of processes not completely in memory

Memory Layout for Multiprogrammed System



More on dual-mode operation

- Software error or request creates **exception** or **trap** (*software* interrupt)
  - Division by zero, request for operating system service
- Other (buggy) user process problems include infinite loop, processes modifying each other or OS
- **Dual-mode** operation allows OS to protect itself and other system components from (possibly buggy) user processes
  - **User mode** and (**privileged**) **kernel mode**
  - **Mode bit** provided by hardware
    - Provides ability to distinguish when system is running user code or kernel code
    - Some instructions only executable in kernel mode (examples?)
    - Some parts of memory inaccessible from user mode (examples?)
    - System call changes mode to kernel, return from call (via return-from-trap or RETT instruction) resets it to user
    - Hardware interrupts (e.g., by IO devices) of user process also change processing to kernel mode

Process Management

- A process is a program in execution. It is a unit of work within the system. Program is a *passive entity*, process is an *active entity*.
- Process needs resources to accomplish its task
  - CPU, memory, I/O, files
  - Initialization data
- Process termination requires reclaim of any reusable resources.
- Single-threaded process has one **program counter** specifying location of next instruction to execute.
  - Process executes instructions sequentially, one at a time, until completion.
- Multi-threaded process has one program counter per thread.
- Typically system has many processes (some user, some operating system) running concurrently on one or more CPUs.
  - Concurrency by multiplexing the CPUs among the processes / threads.

Process Management Activities

The operating system is responsible for the following activities in connection with process management:

- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication
- Providing mechanisms for deadlock handling

Activity #1.1: Process and Concurrency

- Take a look at your computer. Do you have multiple programs making progress at the same time?
- Concurrently running programs are possible even if you have only one CPU. How can that happen? Is it real, or is it an illusion?
- If each process runs on its own *virtual CPU*, what needs to be (i) saved when I switch out a process; (ii) restored when I switch in a process? Where do I save to or restore from?
- If a process keeps computing (never gives up the CPU voluntarily), what do you think drives the switch to another process?
- Should disabling interrupt be a privileged instruction, i.e., should we allow it to run in user mode, kernel mode, or both? Why?

Activity #1.2: Functional view of OS

- A computer is smart like a person is smart.
- What makes a person smart? (Think basic biological systems.)
- If I compare a computer to a person, what functional components should a computer system have?
- Roughly speaking, there is an OS subsystem responsible for managing each of the above computing system components. Can you then name some of the major OS subsystems?

Other Important OS Subsystems

- Memory management
- Storage management (including file systems)
- Mass storage management
- IO management
- Plus …

Protection and Security

- **Protection** – any mechanism for controlling access of processes or users to resources defined by the OS.
- **Security** – defense of the system against internal and external attacks.
  - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service
- Systems generally first distinguish among users, to determine who can do what
  - User identities (**user IDs**, security IDs) include name and associated number, one per user
  - User ID then associated with all files, processes of that user to determine access control
  - Group identifier (**group ID**) allows set of users to be defined and controls managed, then also associated with each process, file
  - **Privilege escalation** allows user to change to effective ID with more rights