Operating Systems: Introduction

Neerja Mhaskar

Department of Computing and Software, McMaster University, Canada

Acknowledgements: Material based on the textbook Operating Systems Concepts (Chapter 1)

What is an Operating System?

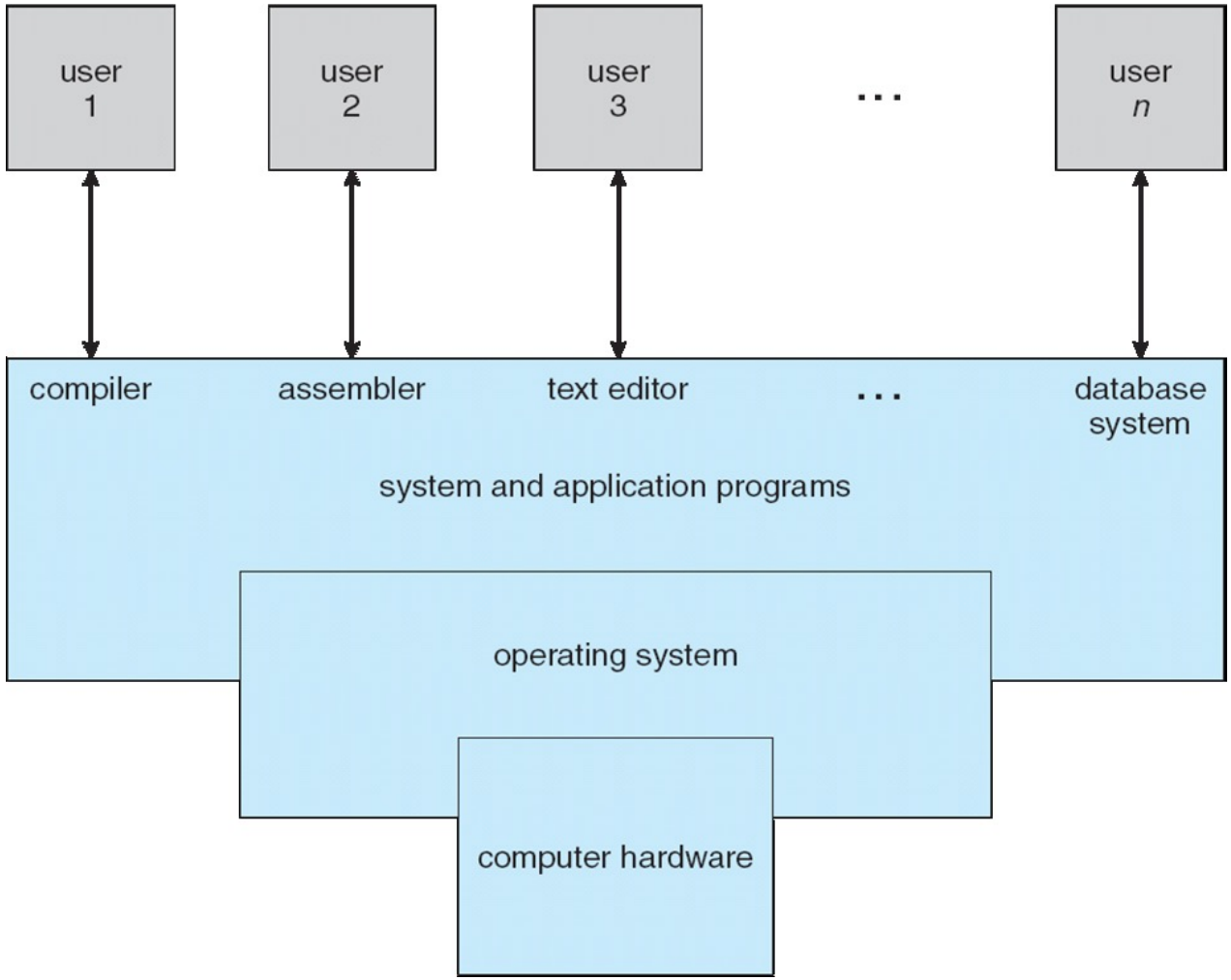
- A program that acts as an intermediary between a user of a computer and the computer hardware
  - They vary dramatically in accomplishing this. For example:
    - Mainframes – primary function is to optimize CPU (central processing units) utilization
    - Smart phones – Ease of use
    - PC – Everything in between

- **Operating system goals:**
- *Efficient use:* Ensure efficient use of a computer's resources.
- *User convenience:* Provide convenient methods of using a computer system.
- *Non interference:* Prevent interference in the activities of its users

Computer System Structure

- Computer system can be divided into **four component** :
  - **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

Abstract view of the components of a computer system



What Operating Systems Do?

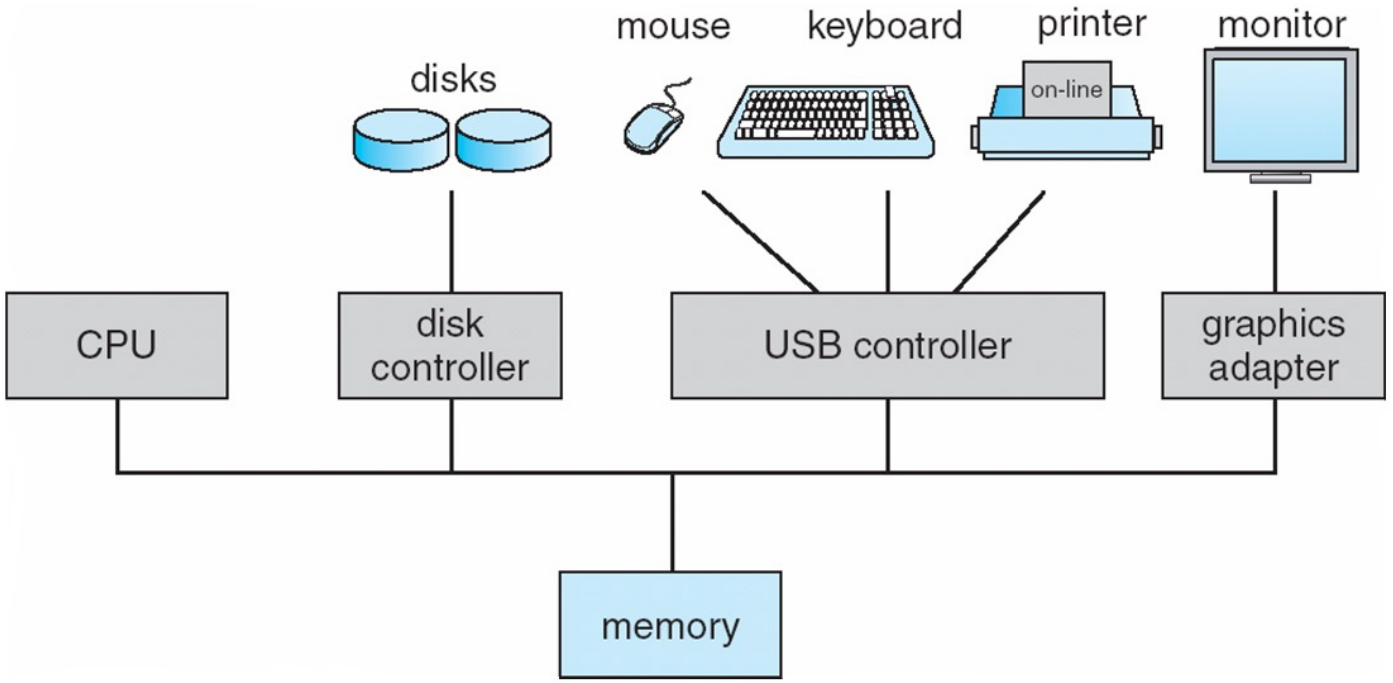
- OS is a **resource allocator**
  - Manages all resources (such as memory, CPU time, I/O devices)
  - Decides between conflicting requests for efficient and fair resource use
- OS is a **control program**
  - Controls execution of programs to prevent errors and improper use of the computer

What is an OS?

- Primarily an OS consists of the
  - **Kernel**
    - Part that stays in main memory (and is the one program always running on the computer)
    - Controls the execution of all other programs
      - Other programs (system or user) interact with it through *system calls* (are routines mostly written in a high-level language (C or C++). However, lower-level task written in assembly.)

Computer System Organization

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



I/O Structure

- A general-purpose computer system consists of CPUs and multiple device controllers that are connected through a common bus.
- Each **device controller** is in charge of a specific type of device(s). Depending on the controller, more than one device may be attached.
- A device controller maintains some local buffer storage and a set of special-purpose registers.
- The device controller is responsible for moving the data between the peripheral devices that it controls and its local buffer storage.
- Typically, in OS there is a **device driver** for each device controller.
- This device driver (a piece of software) provides the operating system with a uniform interface to the device.

I/O Structure

- To start an I/O operation,
  - The device driver loads the appropriate registers within the device controller.
  - The device controller, in turn, examines the contents of these registers to determine what action to take (such as “read a character from the keyboard”).
  - The controller starts the transfer of data from the device to its local buffer.
  - Once the transfer of data is complete, the device controller sends an interrupt signal to CPU.
  - CPU transfers control to ISR which handles this request.

Computer System Operation - Computer Startup

- When a computer system is switched on or rebooted,
  - it automatically loads a program (**bootstrap program**) stored on a reserved part of an I/O device typically a disk (ROM or EEPROM, generally known as **firmware**), and starts executing the program.
- The bootstrap program then
  - Loads the Operating System kernel
  - and system programs (called system program or system daemons)
- The kernel and system programs run all the time on the computer to provide services.
- After the system is fully booted it waits for an **event** to occur.

Computer System Operation - Interrupts

- An operating system is **event driven** and events occur by interrupts. Therefore, OS is **interrupt driven**.
- Interrupt requires the operating system to stop and figure out what to do next.
- **Interrupt:** A mechanism that enables a device/software to notify the CPU that it needs attention.
- An interrupt is caused by a
  - signal to CPU from a device attached to a computer via system bus (hardware), or
  - from an executing program within the computer through system calls.
- A **trap** or **exception** is a software-generated interrupt caused either by an error or a user request.

Interrupt Handling

- When an interrupt occurs:
  - CPU stops executing current task
  - The operating system preserves the state of the CPU by storing registers and the program counter
  - Transfers execution to a fixed location, which has the starting address of **Interrupt Service Routine (ISR)**
  - ISR handles the interrupt, after which
  - Interrupted process resumes its execution.
- Note: Separate segments of code determine what action should be taken for each type of interrupt

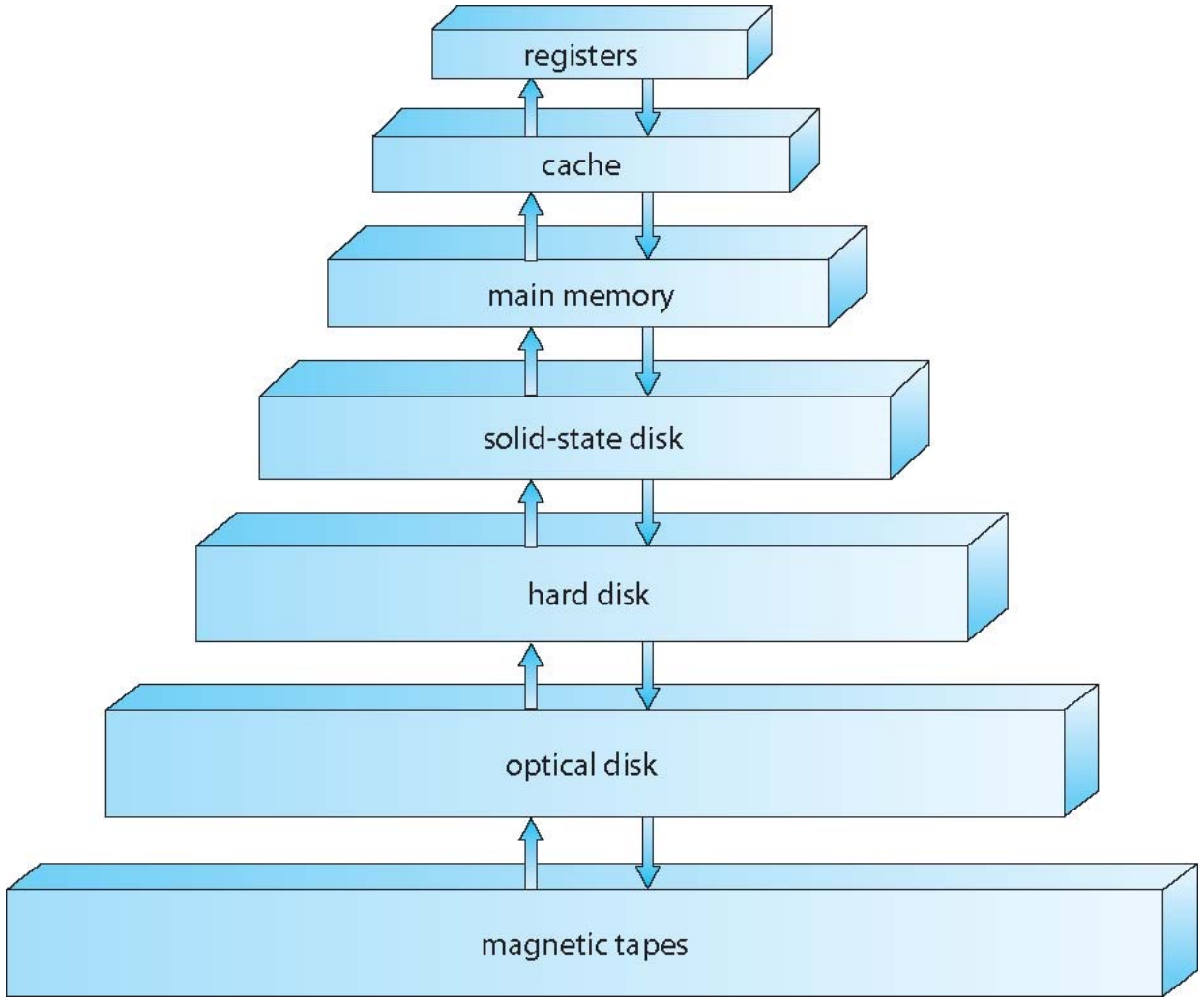
Interrupt Handling

- Implementing ISR as a routine is slow and therefore inefficient.
- Since only predefined interrupts exist, a table of pointers to the various interrupt routines is used instead
- This table of pointers (addresses of the various interrupt routines) is stored in low memory.
- This table or the array of addresses is called the **Interrupt vector**.
- Windows and Linux dispatch interrupts in this manner.

Storage Structure

- CPU can access only main memory (**RAM – Random access memory**).
  - For programs to run, they must be stored in main memory.
  - Main memory is *volatile* (lose data with loss of power)
  - Need secondary storage (e.g., Hard disk drives, CDs, magnetic tapes etc.) to store data permanently and in large quantities.
- **All forms of memory provide an array of bytes**, and each byte as its own address.

Storage-Device Hierarchy



Storage systems organized in hierarchy w.r.t speed, cost and volatility

Storage Definitions and Notation Review

The basic unit of computer storage is the **bit**. A bit can contain one of two values, 0 and 1. All other storage in a computer is based on collections of bits. Given enough bits, it is amazing how many things a computer can represent: numbers, letters, images, movies, sounds, documents, and programs, to name a few. A **byte** is 8 bits, and on most computers, it is the smallest convenient chunk of storage. For example, most computers don't have an instruction to move a bit but do have one to move a byte. A less common term is **word**, which is a given computer architecture's native unit of data. A word is made up of one or more bytes. For example, a computer that has 64-bit registers and 64-bit memory addressing typically has 64-bit (8-byte) words. A computer executes many operations in its native word size rather than a byte at a time.

Computer storage, along with most computer throughput, is generally measured and manipulated in bytes and collections of bytes.
A **kilobyte**, or **KB**, is 1,024 bytes
a **megabyte**, or **MB**, is $1{,}024^2$ bytes
a **gigabyte**, or **GB**, is $1{,}024^3$ bytes
a **terabyte**, or **TB**, is $1{,}024^4$ bytes
a **petabyte**, or **PB**, is $1{,}024^5$ bytes

Computer manufacturers often round off these numbers and say that a megabyte is 1 million bytes, and a gigabyte is 1 billion bytes. Networking measurements are an exception to this general rule; they are given in bits (because networks move data a bit at a time).

Caching

- **Cache** is a faster storage system than main memory and is located very close to CPU or the CPU itself.
- When CPU needs a particular piece of information it first checks in cache, if it finds it, it is called a '**Hit**'.
- If the information is not in the cache it called a '**Miss**'
- CPU then gets information from Main memory and places a copy of it in the Cache.
- Careful selection of cache size and replacement policy greatly increases the system's performance.
- Other caches implemented into the hardware
  - Ex: Instruction Cache – stores the instruction expected to be implemented next – This saves many CPU cycles (otherwise, it needs to go fetch the instruction from main memory every time its needed.)

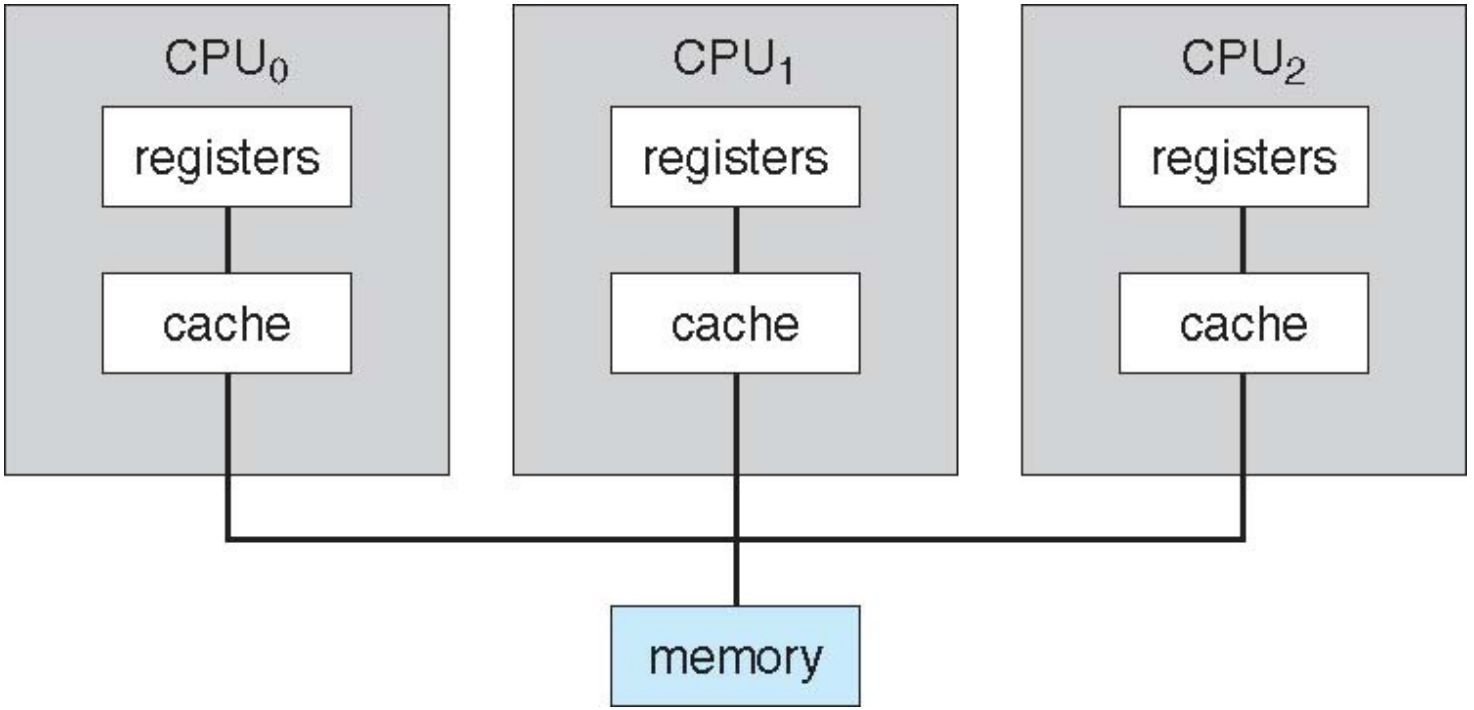
Computer-System Architecture

- A **single processor** system has
  - One general purpose CPU
  - Possibly more than one special purpose CPUs (cannot process user request)
- **Multiprocessors** (parallel systems, multi-core systems)
  - More than one general purpose CPU's (sometimes share memory, bus and peripherals)
  - Advantages include: Increased throughput, Economy of scale, and Increased reliability

Computer-System Architecture

- Two types of Multi-processing:
  - **Asymmetric Multiprocessing:**
    - Boss processor controls system
    - Each processor is assigned a special task.
    - Boss process schedules/allocates processes
  - **Symmetric Multiprocessing (SMP)**
  - Each processor performs all tasks.
  - All Modern systems (Linux, Windows, MAC) provide support for SMP

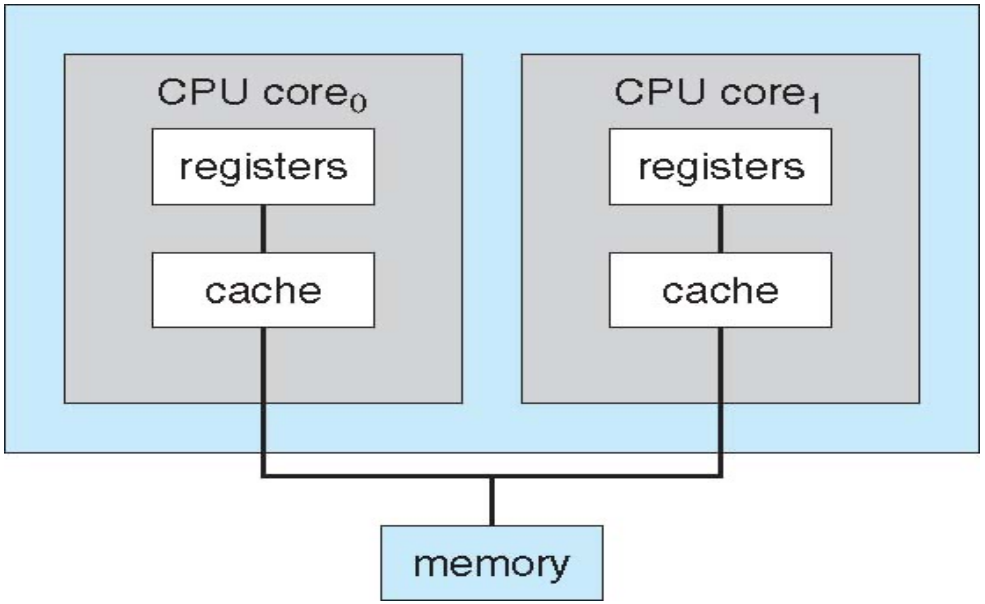
Symmetric Multiprocessing Architecture



Multi-core Design

- Multiprocessor systems can be Multi-chip and/or **multicore** (More than one core on a single chip)
- Multicore systems are efficient as on chip communication is faster and consumes less power

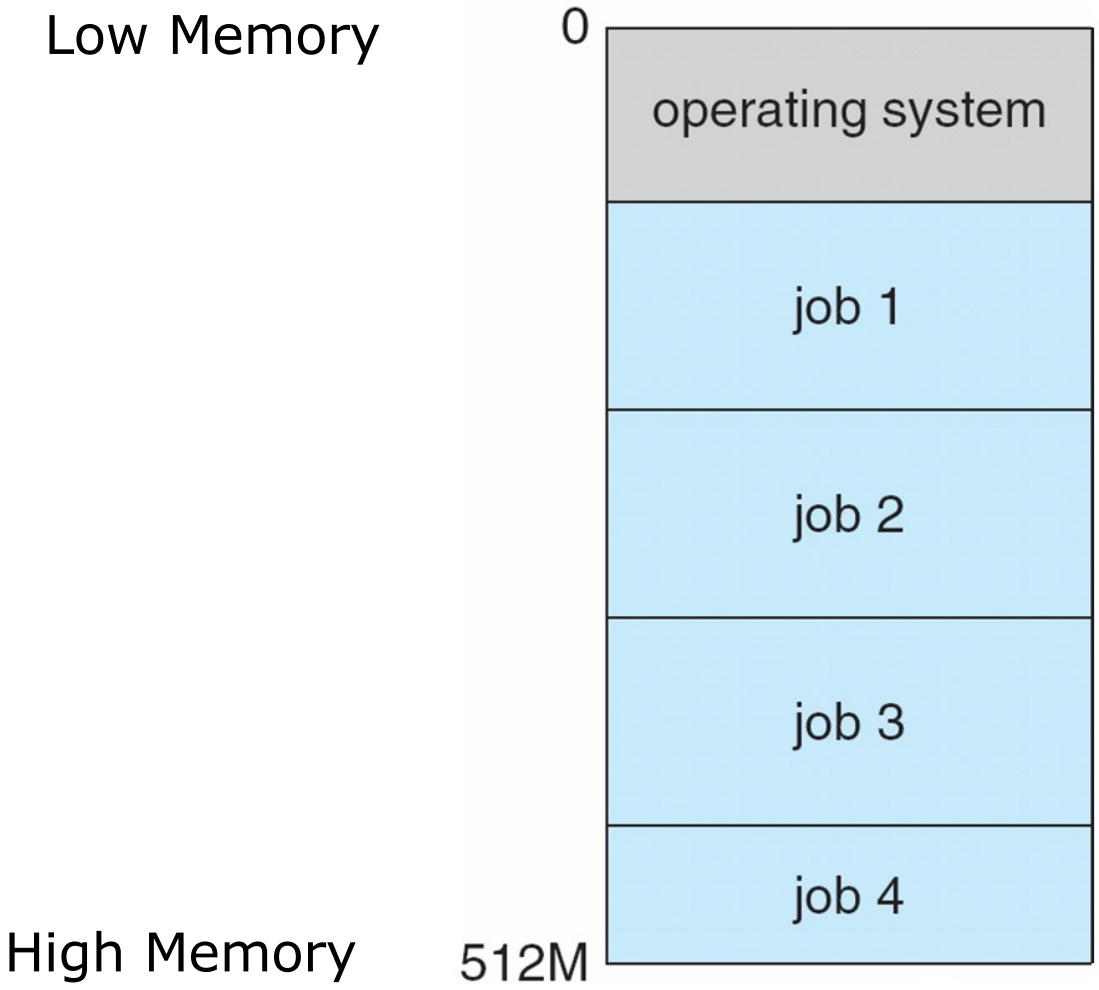
Example of a dual core design:



Operating System Structure

- An operating system provides the environment within which programs are executed
- **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

Memory Layout for Multiprogrammed System



Operating System Structure

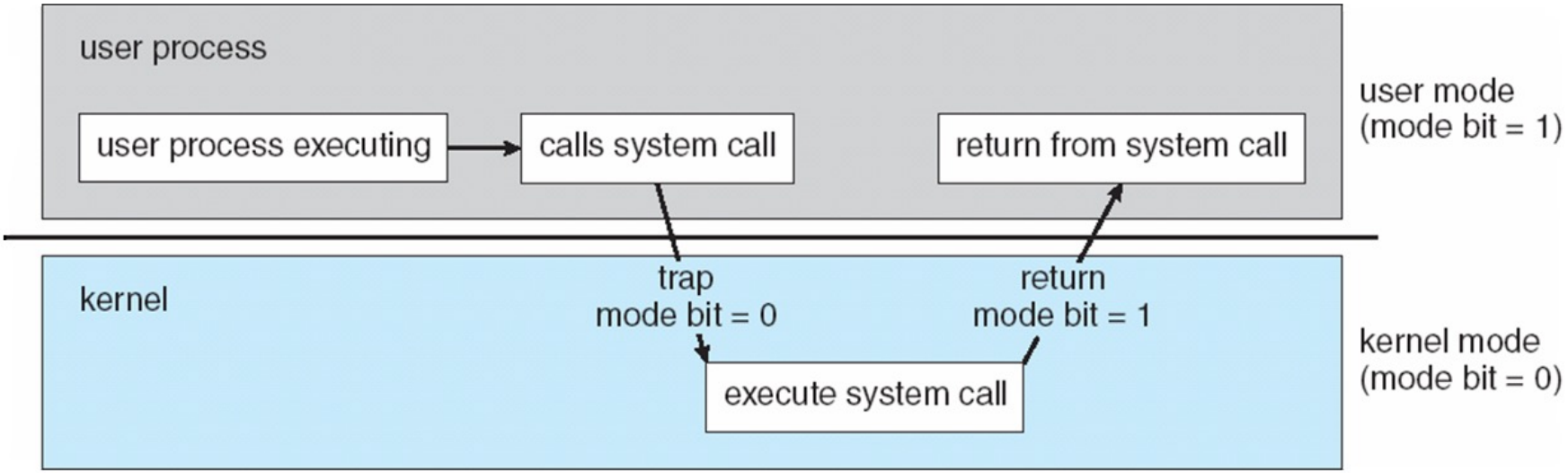
- **Timesharing** (**multitasking**) is logical extension of multiprogramming
  - CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing
  - **Response time** should be < 1 second
  - Each user has at least one program executing in memory ⇨**process**
  - If several jobs ready to run at the same time ⇨ **CPU scheduling**

Operating-System Operations

- **Dual-mode** operation allows OS to protect itself and other system components
  - **User mode** and **kernel mode**
  - **Mode bit** provided by hardware
    - Provides ability to distinguish when system is running user code or kernel code
    - Kernel mode bit = 0 and User mode bit = 1
    - Some instructions designated as **privileged**, only executable in kernel mode
    - System call changes mode to kernel, return from call resets it to user
- Increasingly CPUs support multi-mode operations
  - i.e. **virtual machine manager (VMM)** mode for guest **VMs**

Operating-System Operations

Transition from User to Kernel Mode



Operating-System Operations - Timer

Timer: is a hardware component that can be set to interrupt the computer after a specified period.

- Timer helps to prevent infinite loop / process hogging resources
- Timer is set to interrupt the computer after some time period.
- To set the time period the operating system maintains a counter and sets its value (privileged instruction), which is decremented by the physical clock.
- When counter reaches zero generate an interrupt
- Set up before scheduling process to regain control or terminate program that exceeds allotted time

Key functionality of an OS

- Process Management
- Memory Management
- Storage and File management
- Protection and Security

Process

- Process is a program in execution
- Program is ***passive*** entity stored on disk (**executable file**), process is ***active***
  - Program becomes process when executable file loaded into memory

Process representation in Linux

- Processes in Linux are referred to as **tasks**.
- Represented by the C structure `task_struct`

```
pid t_pid; /* process identifier */
long state; /* state of the process */
unsigned int time_slice /* scheduling information */
struct task_struct *parent; /* this process's parent */
struct list_head children; /* this process's children */
struct files_struct *files; /* list of open files */
struct mm_struct *mm; /* address space of this process */
```

Kernel Data Structures

- Many similar to standard programming data structures
  - Linked lists (single, double and circular)
  - Binary search tree
  - Hash Tables
- Bitmap - string of *n* binary digits representing the status of *n* items
- Linux data structures defined in ***include*** files `<linux/list.h>`, `<linux/kfifo.h>`, `<linux/rbtree.h>`

Computing Environments - Virtualization

- **Virtualization** is a technology that creates abstraction of the computer hardware, thereby creating an illusion that all its operating systems are running on it own private computer.
  - **VMM** (virtual machine Manager) provides virtualization services
- Virtualization belongs to the *Emulation* class of software.
  - **Emulation** is a methodology used when complied program's source CPU type different from target CPU type.
  - This same concept can be extended to allow an entire operating system written for one platform to run on another.
  - Disadvantage: Is lot slower.

Computing Environments - Virtualization