

Course Number: EECIE17-S3303

Course Name: Operating System

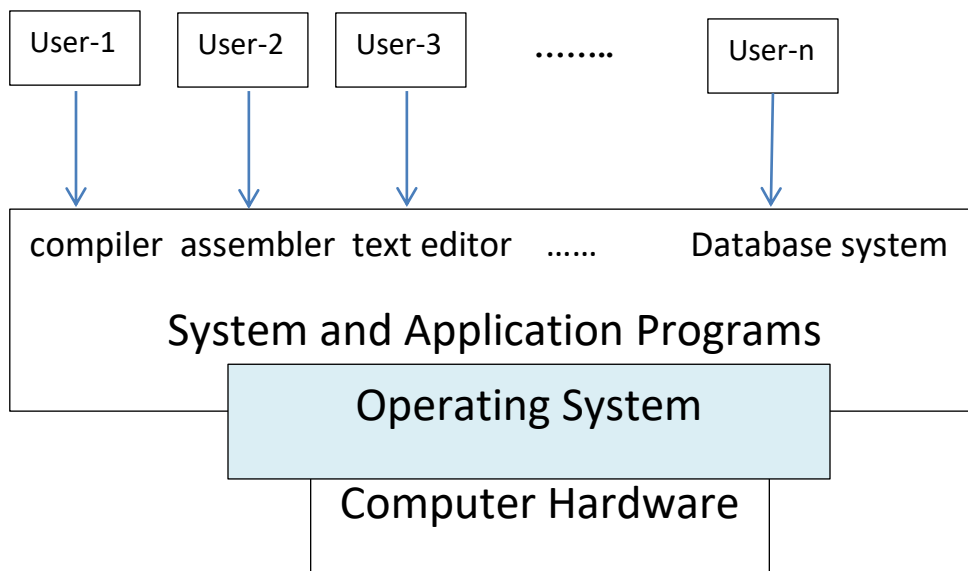
Credit Hours: (4,2,0,0) (Units, Theory, Tutorial, Practical)

Lecturer: Dr. Sahar Abdulaziz Altalib

Chapter 1 - Introduction and Overview

Computer System Components

- Hardware
 - Provides basic computing resources (CPU, memory, I/O devices).
- Operating System
 - Controls and coordinates the use of hardware among application programs.
- Application Programs
 - Solve computing problems of users (compilers, database systems, video games, business programs such as banking software).
- Users
 - People, machines, other computers



Introduction

What is an Operating System (OS)?

- An OS is a program that acts an intermediary between the user of a computer and computer hardware.
- Major cost of general purpose computing is software.
 - OS simplifies and manages the complexity of running application programs efficiently.

Goals of an Operating System

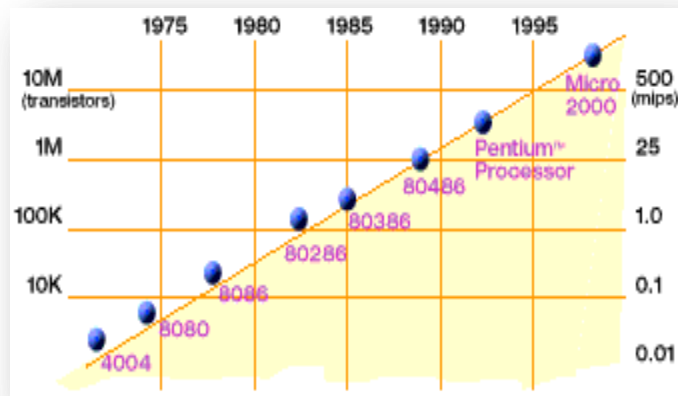
- Simplify the execution of user programs and make solving user problems easier.
- Use computer hardware efficiently.
 - Allow sharing of hardware and software resources.
- Make application software portable and versatile.
- Provide isolation, security and protection among user programs.
- Improve overall system reliability
 - Error confinement, fault tolerance, reconfiguration.

Why should I study Operating Systems?

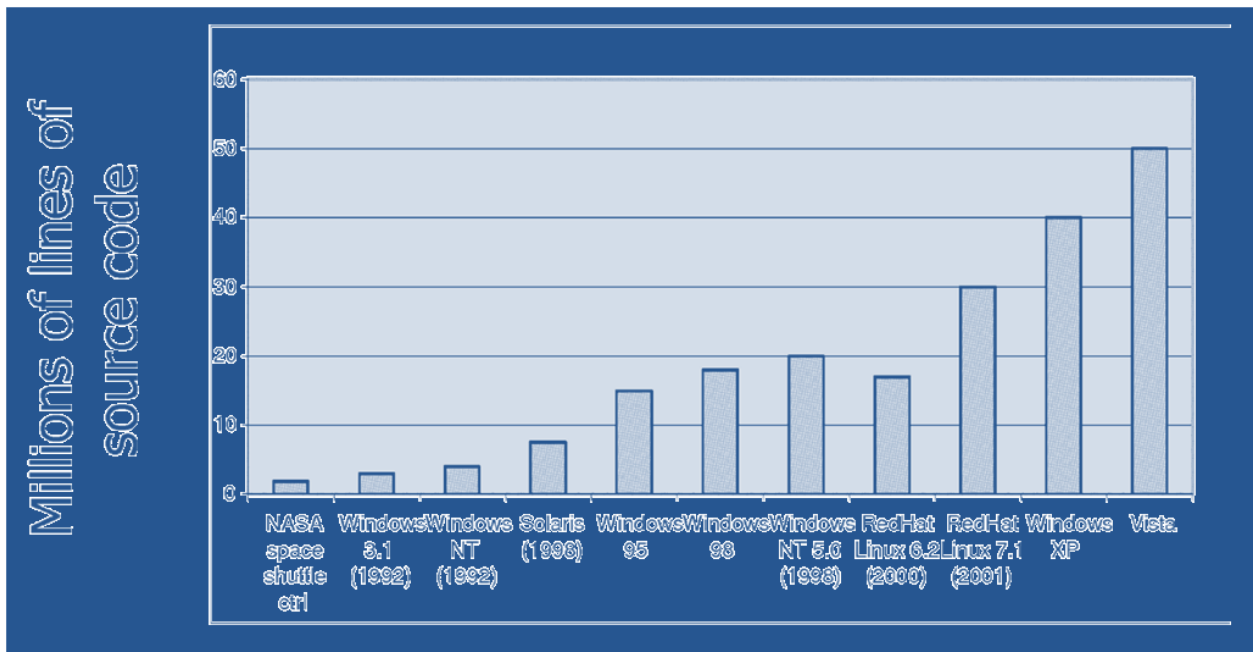
- Need to understand interaction between the hardware and applications
 - New applications, new hardware.
 - Inherent aspect of society today
- Need to understand basic principles in the design of computer systems
 - efficient resource management, security, flexibility
- Increasing need for specialized operating systems
 - e.g. embedded operating systems for devices - cell phones, sensors and controllers
 - real-time operating systems - aircraft control, multimedia services

Hardware Complexity Increases

Moore's Law: 2X transistors/Chip Every 1.5 years



Software Complexity Increases



y-axis: 0-10-20-30-40-50-60 Millions of lines of source code

x-axis: NASA space shuttle control, Windows 3.1 (1992), Windows NT (1992), Solaris (1998), Windows 95, Windows 98, Windows NT 5.0 (1998), RedHat Linux 6.2 (2000), RedHat Linux 7.1 (2001), Windows XP, Vista

Operating System Views

Resource allocator

- To allocate resources (software and hardware) of the computer system and manage them efficiently.

Control program

- Controls execution of user programs and operation of I/O devices.

Kernel

- The program that executes forever (everything else is an application with respect to the kernel).

Operating System Spectrum

Monitors and Small Kernels

- special purpose and embedded systems, real-time systems

Batch and multiprogramming

Timesharing

- workstations, servers, minicomputers, timeframes

Transaction systems

Personal Computing Systems

Mobile Platforms, devices (of all sizes)

Early Systems - Bare Machine (1950s)

Hardware – *expensive*; Human – *cheap*

Structure

- Large machines run from console
- Single user system
 - Programmer/User as operator
- Paper tape or punched cards

Early software

- Assemblers, compilers, linkers, loaders, device drivers, libraries of common subroutines.

Secure execution

Inefficient use of expensive resources

- Low CPU utilization, high setup time.

Simple Batch Systems (1960's)

Reduce setup time by batching jobs with similar requirements.

Add a card reader, Hire an operator

- User is NOT the operator
- Automatic job sequencing
 - Forms a rudimentary (being in the earliest stages of development) OS.
- Resident Monitor
 - Holds initial control, control transfers to job and then back to monitor.
- Problem
 - Need to distinguish job from job and data from program.

Supervisor/Operator Control

Secure monitor that controls job processing

- Special cards indicate what to do.
- User program prevented from performing I/O

Separate user from computer

- User submits card deck
- cards put on tape
- tape processed by operator
- output written to tape
- tape printed on printer

Problems

- Long turnaround time - up to 2 DAYS!!!
- Low CPU utilization
 - I/O and CPU could not overlap; slow mechanical devices.

Batch Systems – Issues

Solutions to speed up I/O:

Offline Processing

- Load jobs into memory from tapes, card reading and line printing are done offline.

Spooling (putting jobs in a buffer, a special area in memory or on a disk where a device can access them when it is ready)

- Use disk (random access device) as large storage for reading as many input files as possible and storing output files until output devices are ready to accept them.
- Allows overlap - I/O of one job with computation of another.
- Introduces notion of a job pool that allows OS choose next job to run so as to increase CPU utilization.

Batch Systems - I/O completion

How do we know that I/O is complete? Polling:

- Device sets a flag when it is busy.
- Program tests the flag in a loop waiting for completion of I/O.

Interrupts:

- On completion of I/O, device forces CPU to jump to a specific instruction address that contains the interrupt service routine.
- After the interrupt has been processed, CPU returns to code it was executing prior to servicing the interrupt.

Multiprogramming

Use interrupts to run multiple programs simultaneously

- When a program performs I/O, instead of polling, execute another program till interrupt is received.

Requires secure memory, I/O for each program.

Requires intervention if program loops indefinitely.

Requires CPU scheduling to choose the next job to run.

Timesharing

Hardware – *getting cheaper*; Human – *getting expensive*

Programs queued for execution in FIFO order.

Like multiprogramming, but timer device interrupts after a quantum (time slice).

- Interrupted program is returned to end of FIFO
- Next program is taken from head of FIFO

Control card interpreter replaced by command language interpreter (CLI).

Interactive (action/response)

- When OS finishes execution of one command, it seeks the next control statement from user.

File systems

- Online file system is required for users to access data and code.

Virtual memory

- Job is swapped in and out of memory to disk.

Personal Computing Systems

Hardware – *cheap*; Human – *expensive*

Single user systems, portable.

I/O devices - keyboards, mice, display screens, small printers.

Laptops and palmtops, Smart cards, Wireless devices.

Single user systems may not need advanced CPU utilization or protection features.

Advantages:

- user convenience, responsiveness, ubiquitous

Parallel Systems

- Multiprocessor systems with more than one CPU in close communication.
- Improved Throughput, economical, increased reliability.
- Kinds:
 - Vector and pipelined
 - Symmetric and asymmetric multiprocessing
 - Distributed memory vs. shared memory
- Programming models:

- Tightly coupled vs. loosely coupled, message-based vs. shared variable

Distributed Systems

Hardware – *very cheap*; Human – *very expensive*

- Distribute computation among many processors.
- Loosely coupled -
 - no shared memory, various communication lines
- client/server architectures
- Advantages:
 - resource sharing
 - computation speed-up
 - reliability
 - communication - e.g. email
- Applications - digital libraries, digital multimedia

Real-time systems

- Correct system function depends on timeliness
- Feedback/control loops
- Sensors and actuators
- Hard real-time systems -
 - Failure if response time too long.
 - Secondary storage is limited
- Soft real-time systems -
 - Less accurate if response time is too long.
 - Useful in applications such as multimedia, virtual reality.

Chapter 2 – Computer System Structure

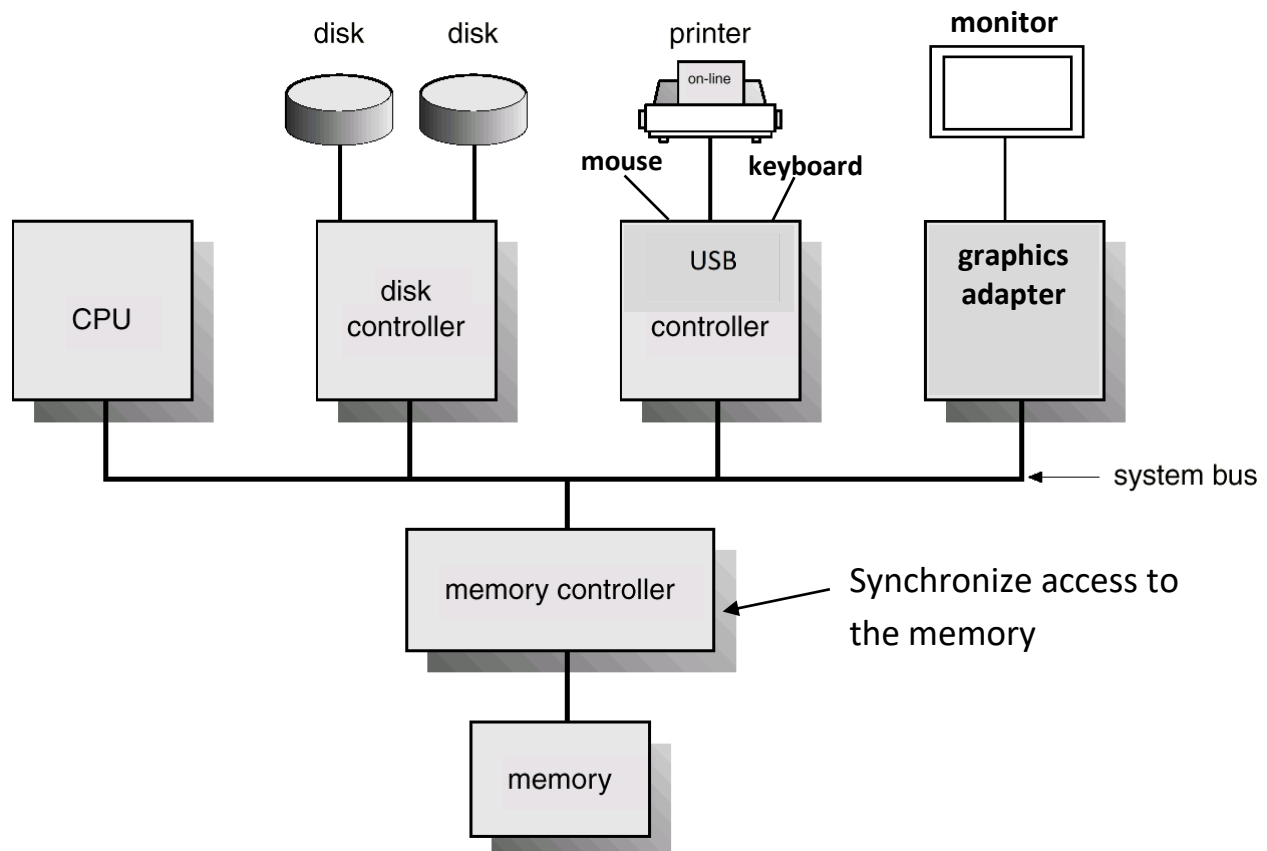


Figure 2.1: Computer System Architecture

Computer-System Operation

- 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.
- Device controller informs CPU that it has finished its operation by causing an *interrupt*.

For computer to start running, it needs to have an initial program to run. It is the **bootstrap** program, it initializes:

- CPU registers
- Device controllers
- Memory contents

The bootstrap program must locate and load into memory the operating system kernel and waits for some event (interrupt) to occur.

Hardware interrupt → triggered by sending a signal to the CPU by way of system bus.

Software interrupts → by executing a system call.

Types of events that may trigger an interrupt:

1. Completion of an I/O operation
2. Division by zero
3. Invalid memory access
4. Request for some operating system service

To deal with interrupt, a service routine is provided. The routine will call the interrupt specific-handler. A table of pointers to interrupt routines can be used instead. The table is stored in low memory (first 100 or so locations).

Common Functions of 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 is being processed to prevent a *lost interrupt*.
- A *trap* is a software-generated interrupt caused either by an error or a user request.
- Modern operating system is *interrupt* driven.

Interrupt Handling

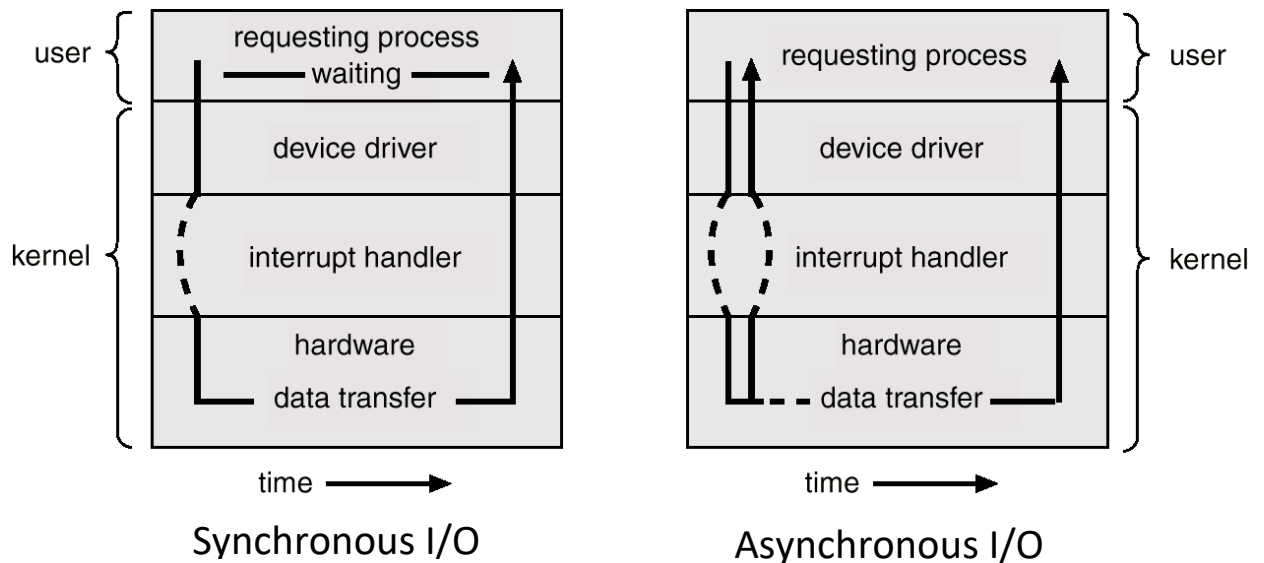
- The operating system preserves the state of the CPU by storing registers and the program counter.
- Determines which type of interrupt has occurred:
- Separate segments of code determine what action should be taken for each type of interrupt

I/O Structure

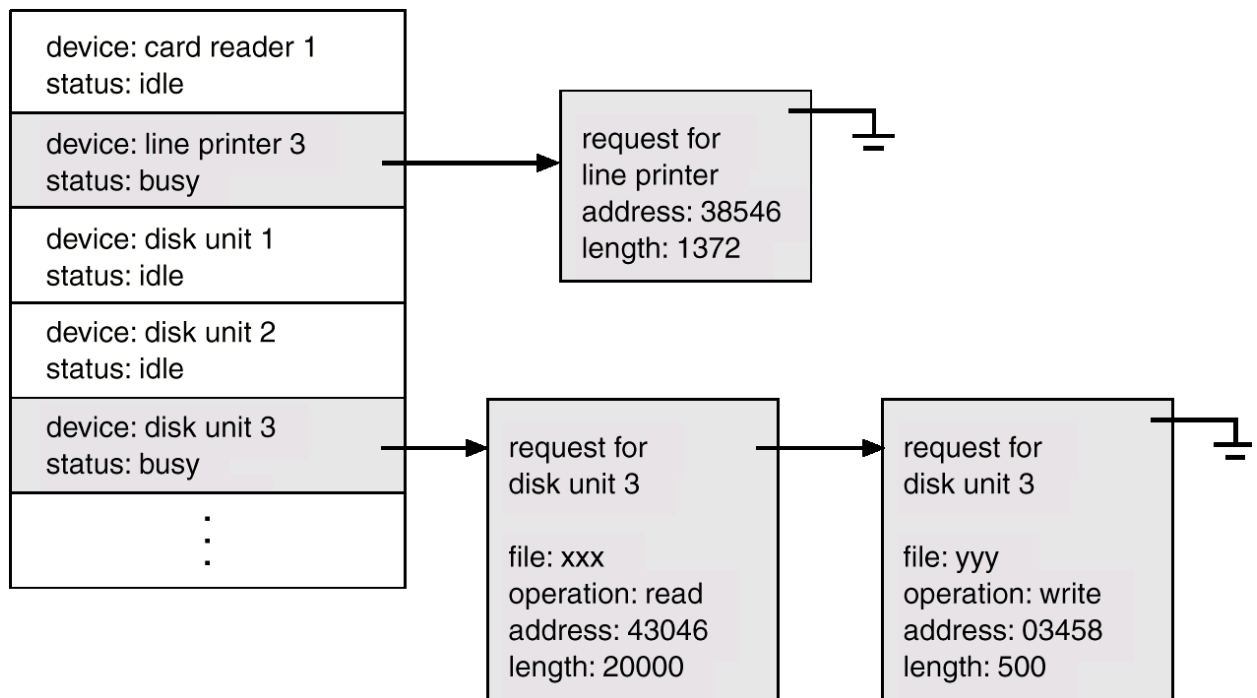
- **Synchronous I/O:** After I/O starts, control returns to user program only upon I/O completion.
 - ◆ Wait instruction idles the CPU until the next interrupt
 - ◆ Wait loop (contention for memory access).
 - ◆ At most one I/O request is outstanding at a time, no simultaneous I/O processing.
- **Asynchronous I/O:** After I/O starts, control returns to user program without waiting for I/O completion.

- ◆ *System call* – request to the operating system to allow user to wait for I/O completion.
- ◆ *Device-status table* contains entry for each I/O device indicating its type, address, and state.
- ◆ Operating system indexes into I/O device table to determine device status and to modify table entry to include interrupt.

Two I/O Methods



Device-Status Table



Direct Memory Access Structure

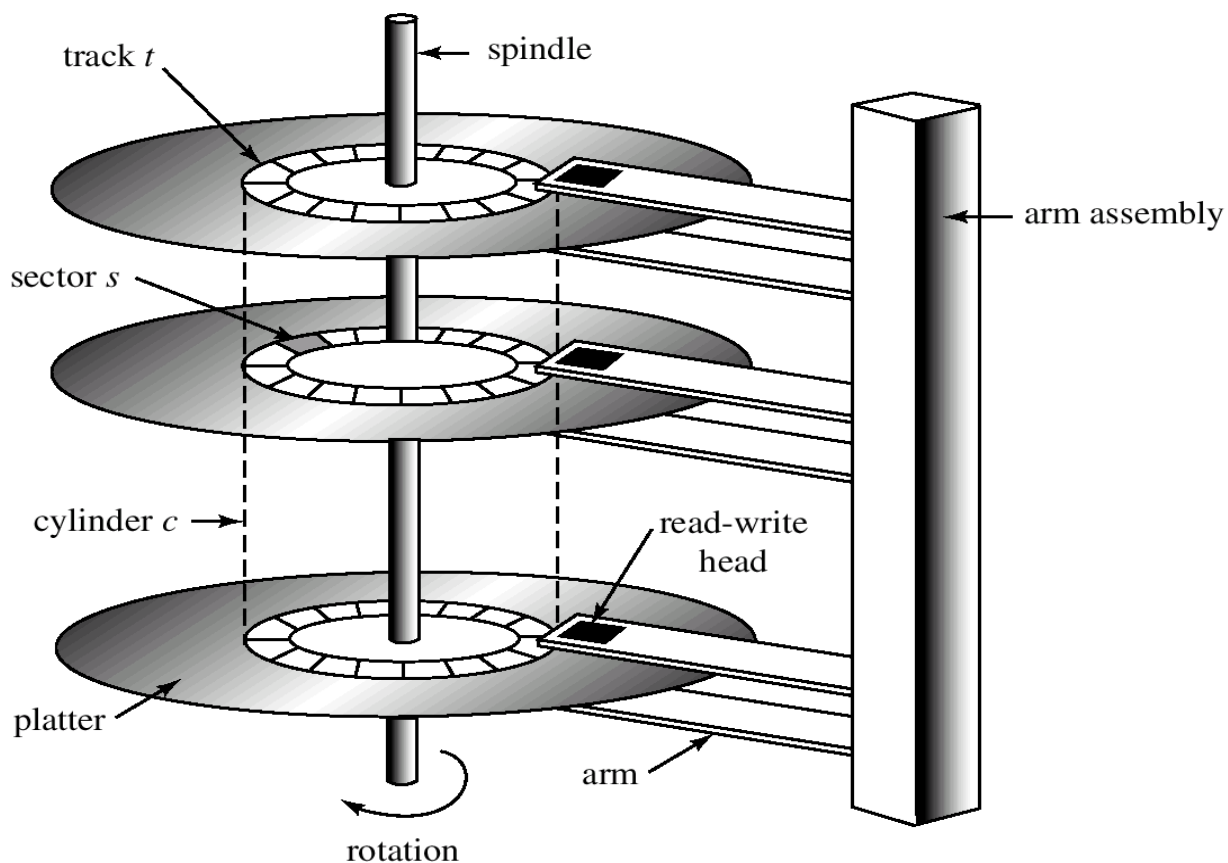
- Used for high-speed I/O devices able to transmit information at close to memory speeds.
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention.
- Only one interrupt is generated per block, rather than the one interrupt per byte.

Storage Structure

- 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 (صفائح) 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.

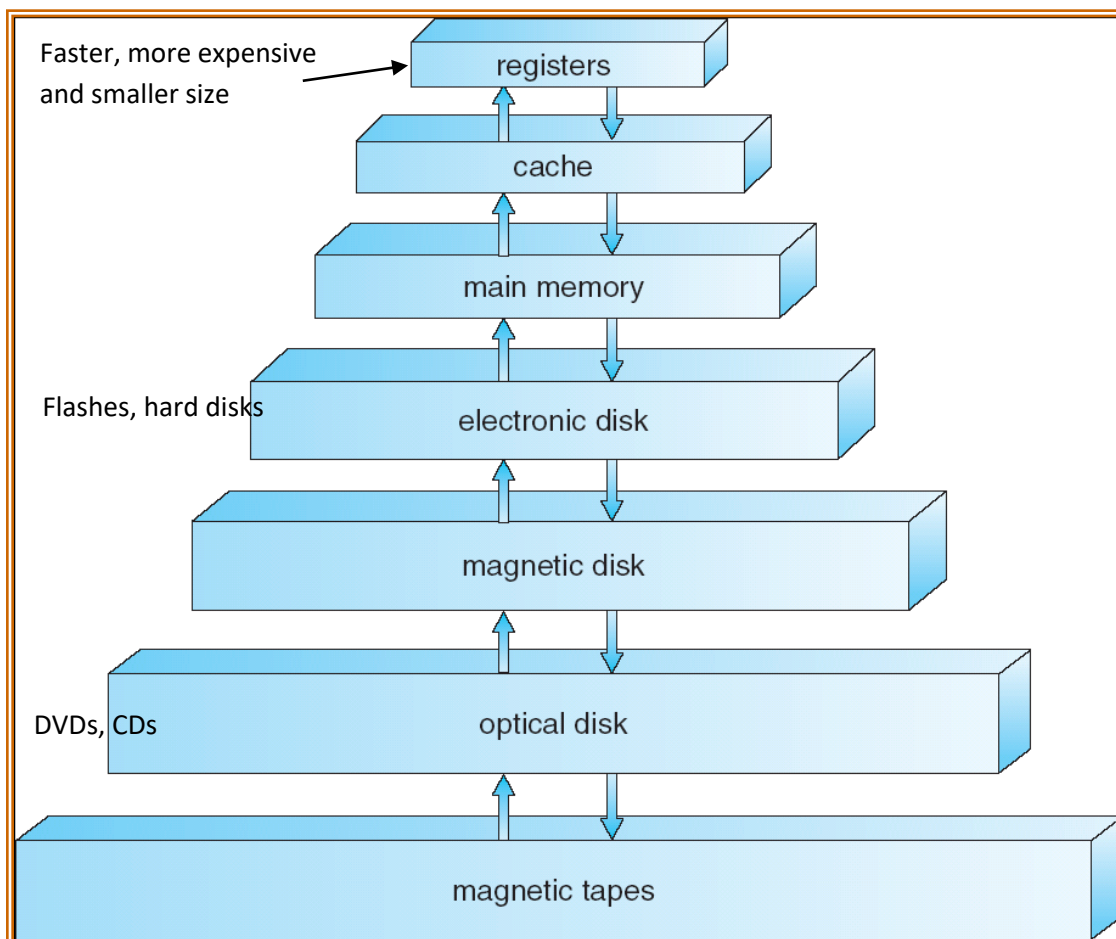
Moving-Head Disk Mechanism



Storage Hierarchy

- Storage systems organized in hierarchy based on:
 - ◆ 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



When we need a particular piece of information:

1. Check the cache, if it is there, use the information directly.
2. If it is not, use the information from the main storage system.

Caching Types:

1. The programmer (or compiler) implements the register-allocation and register-replacement algorithms to decide which information to keep in registers and which to keep in main memory.
2. Hardware-only caches: caches that are implemented totally in HW. Most systems have an instructions expected to be executed. This type is outside of the control of the operating system.
3. Software controlled caches: since caches have limited size. Cache management is an important design problem. Careful selection of the cache size and of a replacement policy can result in 80-90% of all accesses being in the cache, causing extremely high performance.
4. Main memory can be viewed as a fast cache for secondary memory, since data on secondary storage must be copied into main memory for use, and data must be in memory before being moved to secondary storage for safekeeping. This is usually controlled by the operating system.

Coherency (ترابط) and Consistency (مطابقة)

In a hierarchical storage structure, the same data may appear in different levels of the storage system. For example, consider an integer A located in the file B resides on magnetic disk that is to be incremented by 1:

1. Issue an I/O operation to copy the disk block on which A resides to main memory.
2. Copy A to the cache, and by a copying of A to an internal register.
3. The increment takes place in the internal register.
4. The new value of A is written back to the magnetic disk.

The copy of A appears in several places.

Cache coherency problem occurs in multiprocessor environment where a copy of A may exist simultaneously in several caches. It is a hardware issue.

In distributed environment, the situation becomes more complex. Several copies of the same file can be kept on different computers that are distributed in space.