GH raisoni COLLEGE
Engineering and Management Nagpur

COMPUTER ARCHITECTURE AND OPERATING SYSTEM

Course Code: 23BTCYL204

B. Tech - IV Semester

Prepared by:

Shalini kumari

Assistant Professor

Department of Computer Science and Engineering(Cyber security)

Raisoni College of Engineering, Nagpur

Teaching and Evaluation Scheme

GH raisoni COLLEGE
Engineering and Management
Nagpur

Course Title: COMPUTER ARCHITECTURE AND OPERATING SYSTEM										
Semester	IV	Teaching Scheme				Evaluation Scheme				
						Theory			Practical	
Term	EVEN	Th	Tu	Pr	Credits	TAE	CAE	ESE	INT	EXT
Course Category	Core	4 Hrs	-	-	04	10	15	50	-	-
Course Code	23BTCYL204									
Teaching Mode	Offline	4 Hrs.			Total	75			-	
Duration of ESE	3 Hrs.					75				

Course Objectives and Outcomes

Course Objectives	1. To provide essential understanding of different subsystems of modern computer system and design aspects these subsystems
	2. Introduces general idea, structure and functions of operating system
	3. Making students aware of basic mechanisms used to handle processes, memory, Storage devices and files.
Course Outcomes	Upon successful completion of this course, student shall be able to:
	CO 1. Explain basic computer components, architectures, and pipelining concepts
	CO 2. Describe memory organization, cache techniques, virtual memory, and multiprocessor architectures.
	CO 3. Understand OS structure, types, process management, IPC, and thread models.
	CO 4. Analyze CPU scheduling, process synchronization, and deadlock handling.
	CO 5. Apply memory management, evaluate disk scheduling, file systems, and I/O techniques.

Mapping of Course Outcomes with Program Outcomes and Program Specific Outcomes:

Following mapping is only for reference

Course Outcomes	Program Outcomes and Program Specific Outcomes														
	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO 10	PO 11	PO 12	PSO 1	PSO 2	PSO 3
CO1	2	2	1	2	1	1	1	0	2	2	2	2	3	3	3
CO2	3	3	3	3	2	1	1	1	2	2	3	2	3	2	2
CO3	2	3	2	2	2	1	1	2	3	2	2	2	2	3	2
CO4	2	2	3	3	3	1	1	1	3	2	3	2	3	2	3
CO5	2	2	3	2	2	1	1	2	2	1	2	2	2	3	3

Syllabus

GHraisoni COLLEGE
Engineering and Management
Nagpur

Unit	Contents
Unit 1	**Computer Organization :** Introduction to CPU and concepts of ALU ,Introduction to basic computer architecture,Stored Program Concepts: Von Neumann & Harvard Architecture ,Pipelining: Basic concepts, instruction and arithmetic pipeline, data hazards, control hazards and structural hazards, techniques for handling hazards ,Pipeline vs. Parallelism, Levels of parallelism
Unit 2	Introduction to memory-RAM and ROM , Register transfer, memory transfer, Tri-state bus buffer, Memory Hierarchy: Secondary memory, Main Memory , Cache Memory , Mapping Technique in cache memory: Direct, Full Associative and Set Associative , Performance Implementation in Cache Memory , Virtual memory Concepts, page replacement policies .Multiprocessor architecture: taxonomy of parallel architectures; Flynn‘s Classification
Unit 3	**Operating System:** Functionalities of Operating System, Evolution of Operating System. Types of Operating System: batch, multi-programmed, time-sharing, real-time, distributed, Mobile Operating Systems,Network Operating System, Multi-Processing, Processes: Concept of processes, process states, PCB, process scheduling, co-operating processes, independent process, suspended process, Interaction between processes and OS, Threads: overview, benefits of threads, user and kernel level threads, Thread models.

Syllabus

GHraisoni COLLEGE
Engineering and Management
Nagpur

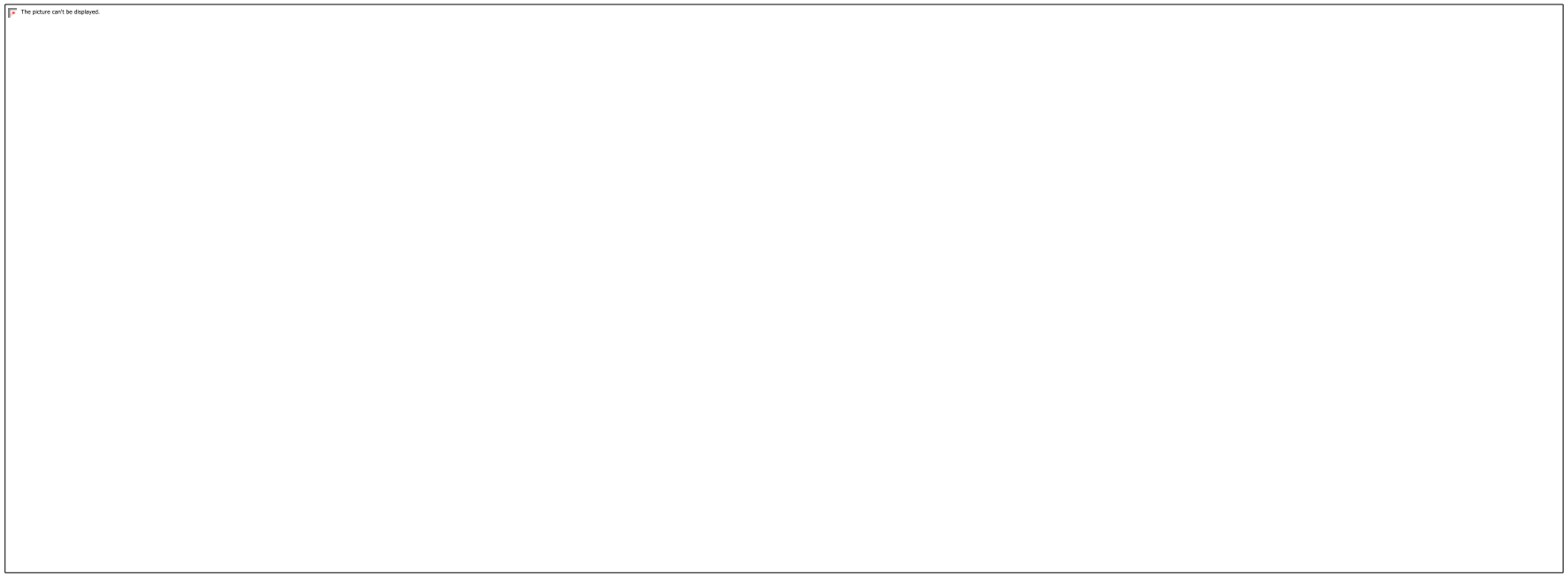
Unit	Contents
Unit 4	CPU scheduling: Scheduling criteria, preemptive & non-preemptive scheduling, scheduling algorithms Process Synchronization: background, critical section problem, synchronization hardware, classical problems of synchronization (producer-consumer, readers-writer, dining philosophers, etc), semaphores, monitors. Deadlocks: deadlock characterization, methods for handling deadlocks, deadlock prevention, deadlock avoidance, deadlock detection, recovery from deadlock.
Unit 5	Logical vs. physical address space, swapping, paging, Segmentation, TLB. Virtual Memory: demand paging, page replacement algorithms (FCFS, LRU, Optimal), thrashing, Working set model. Disk scheduling (FCFS, SSTF, SCAN, C-SCAN, LOOK, C-LOOK etc), File: file system structure, File Directories, File Allocation Methods (contiguous, linked, indexed), I/O: I/O hardware, polling, interrupts, DMA.

Study Material

Text Books	1	Computer Organization and Architecture by Carl Hamacher 4th Edition
	2	Advance computer Architecture by Kai Hwang under Tata McGraw Hill publications
	3	Introduction to Parallel Processing: Algorithms &Architectures, BehroozParhami in Springer Shop
	4	Operating System concepts – Silberchatz; Galvin, Addison Wesley, 6 thEdn.
E-Books	1.	Operating System Concepts -Abraham Silberschatz, Peter Galvin, Greg Gagne
	2.	Modern Operating Systems-Andrew S. Tanenbaum
Reference Books	1	Introduction to Parallel Processing by P. Ravi Prakash, M. Sasikumar, Dinesh Shikhare By PHI Publications
	2	Fundamentals of Parallel Processing by Jordan Harry, Alaghband Gita, PHI
	4	Operating System –Milan Milenkovik, McGraw-Hill, 1987
	5	Operating Systems - 3 rd Edition by Gary Nutt, Pearson Education.

Unit 1

Computer Organization :



CPU (Central Processing Unit)

The central processing unit (CPU) is the main part of a computer that controls how it works. It is made up of the control unit, main memory, and the arithmetic logic unit (ALU). The CPU handles instructions from programs, processes data, stores information, and produces results. Without the CPU, a computer cannot perform tasks or run any applications.

CU (Control Unit)

The control unit manages how the processor works by sending control signals. It decides how data should move inside the computer, controls input and output operations, and fetches the instructions from memory for execution.

ALU (Arithmetic and Logic Unit)

The arithmetic and logic unit is the part of the CPU that handles the calculations and decision-making tasks. It performs arithmetic operations like addition and subtraction, logical operations such as comparisons, and tasks like shifting bits in data.

Registers

Registers are the fastest type of memory located inside the CPU. They temporarily store information that the processor is currently working on, making program execution and operations faster and more efficient. Register serve as the CPU's primary working memory.

Arithmetic Logic Unit (ALU)

The ALU is a digital circuit within the CPU that performs all arithmetic and logical operations. It takes input from registers, processes the operation, and sends the result back to a destination register or memory.

- Performs operations like addition, subtraction, AND, OR, NOT, etc.
- Controlled by the control unit based on the instruction type.

Functions and Operations of ALU

1. Arithmetic Operations

ALU performs arithmetic operations to carry out basic mathematical calculations on binary numbers.

- Operations include addition, subtraction, increment, and decrement.
- Used during instruction execution for tasks like ADD, SUB, INC, and DEC.

2. Logical Operations

These operations manipulate data at the bit level using logic gates.

- Includes bitwise operations like AND, OR, XOR, and NOT.
- Essential for masking, setting, clearing, or toggling specific bits.

3. Shift Operations
ALU can shift bits left or right to aid in fast arithmetic and bit manipulation.

- Types include logical shift, arithmetic shift, and rotate operations.
- Often used for multiplication/division by powers of two and bit-level data formatting.

4. Comparison Operations
ALU supports comparison by evaluating two operands and setting appropriate flags.

- It checks for conditions like equality, greater than, or less than.
- Useful in branching and conditional instructions based on flag values.

5. Status Flag Generation
The ALU sets flags based on the result of operations to influence program control.

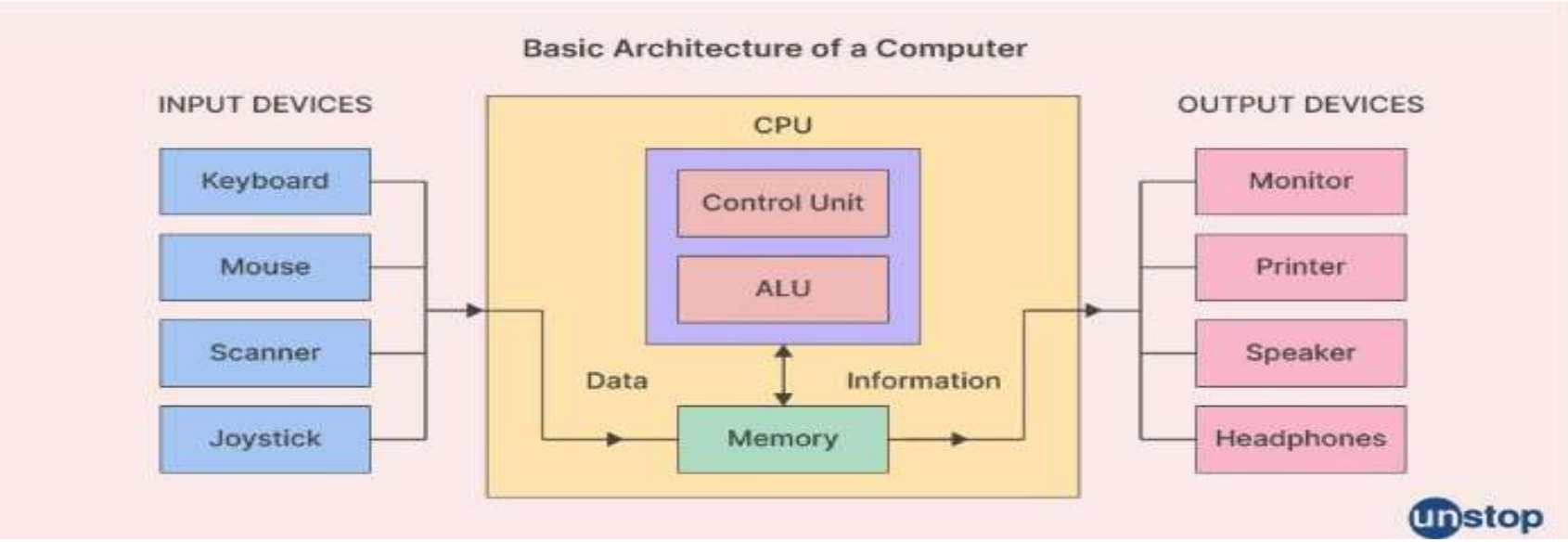
- Common flags: Zero (Z), Carry (C), Sign (S), Overflow (V).
- Flags are used in conditional execution and exception handling.

Introduction to basic computer architecture

Computer architecture is a specification describing how computer software and hardware connect and interact to create a computer network. It determines the structure and function of computers and the technologies it is compatible with – from the central processing unit (CPU) to memory, input/output devices, and storage units.

Understanding the meaning of computer architecture is crucial for both computer scientists and enthusiasts, as it forms the basis for designing innovative and efficient computing solutions. It also helps programmers write software that can take full advantage of a computer's capabilities, allowing them to create everything from web applications to large language models (LLMs).

Components of Computer Architecture



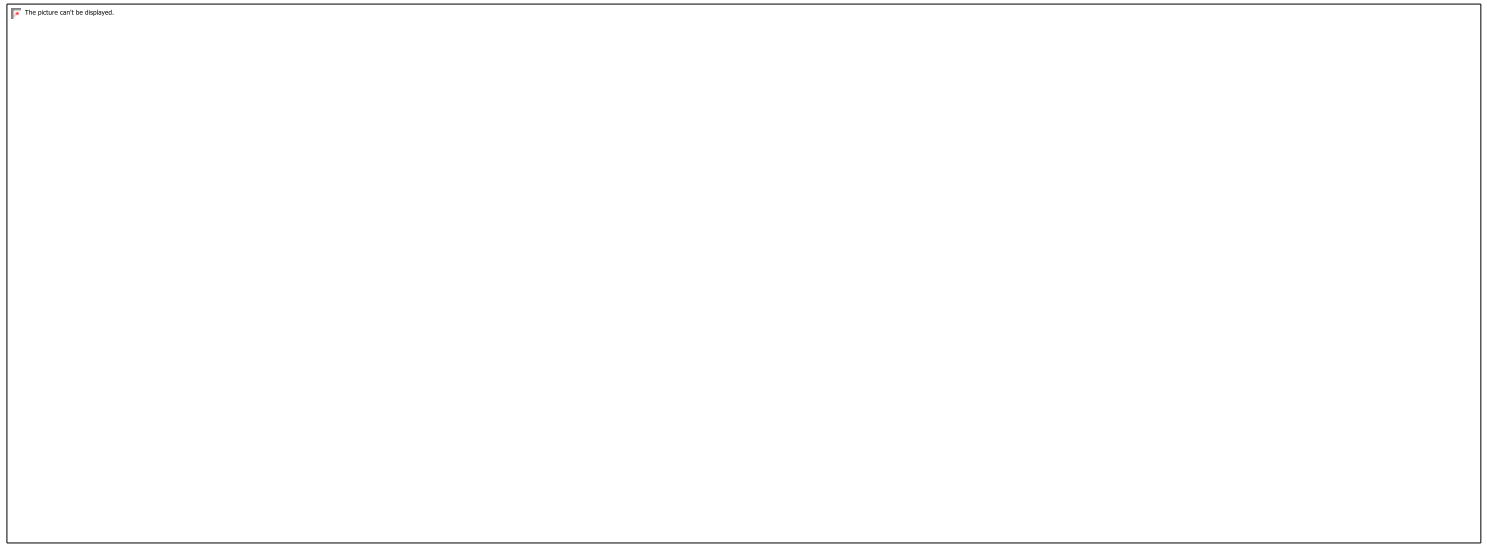
Stored Program Concepts

The Stored Program Concept defines computers holding data & instructions in shared memory, realized by Von Neumann Architecture (single bus, sequential access, simpler, common in PCs) and contrasted with Harvard Architecture (separate buses/memories for data/instructions, parallel access, faster, used in DSPs/microcontrollers), both key models for how CPUs fetch and execute instructions, with Harvard offering speed and Von Neumann offering flexibility and lower cost.

Von Neumann Architecture

Von Neumann architecture is a computer design where instructions and data are stored in the same memory space. This means the CPU fetches both instructions and data from the same memory, using the same pathways. Historically there have been 2 types of Computers:

- **Fixed Program Computers** - Their function is very specific, and they couldn't be reprogrammed, e.g., calculators.
- **Stored Program Computers** - These can be programmed to carry out many different tasks; applications are stored on them, hence the name.

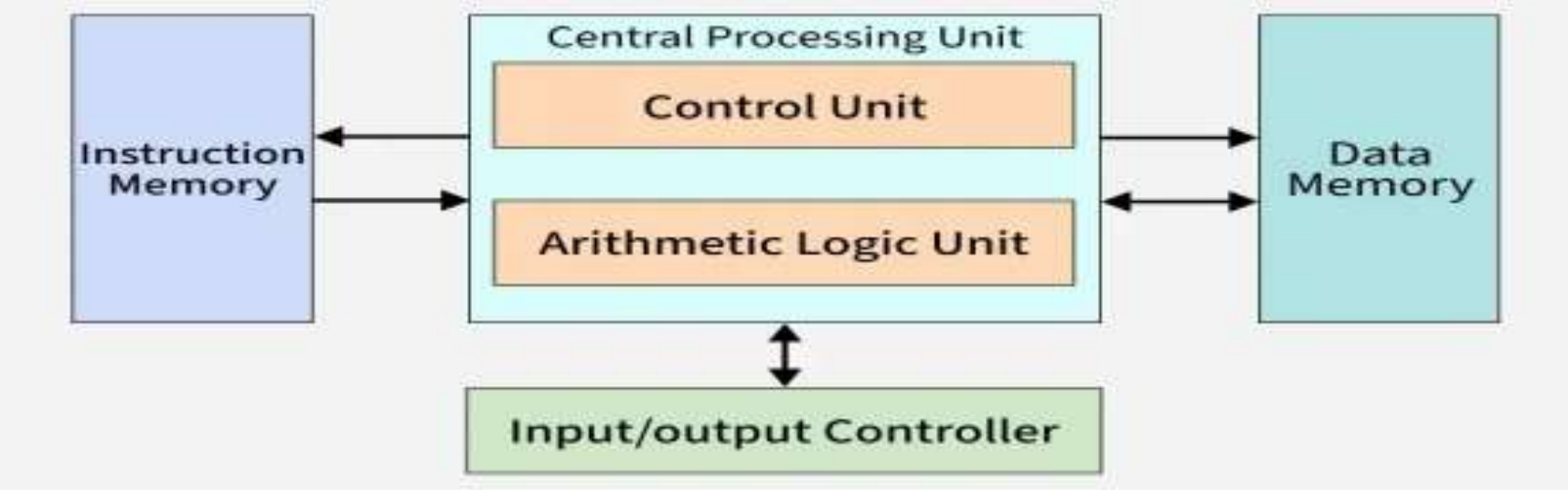


Harvard Architecture

Harvard architecture is a computer design model where program instructions and data are stored in separate memory units that are accessed through independent buses. This separation allows the processor to fetch instructions and access data simultaneously, which helps avoid the bottleneck present in traditional Von Neumann systems.

- Eliminates the Von Neumann bottleneck.
- Faster and predictable performance (suitable for real-time systems).

Parallel access to both instructions and data.



What is Pipelining?

Pipelining is an arrangement of the CPU's hardware components to raise the CPU's general performance. In a pipelined processor, procedures called 'stages' are accomplished in parallel, and the execution of more than one line of instruction occurs. Now let us look at a real-life example that should operate based on the pipelined operation concept. Consider a water bottle packaging plant. For this case, let there be 3 processes that a bottle should go through, ensing the bottle(I), Filling water in the bottle(F), Sealing the bottle(S).
It will be helpful for us to label these stages as stage 1, stage 2, and stage 3. Let each stage take 1 minute to complete its operation. Now, in a non-pipelined operation, a bottle is first inserted in the plant, and after 1 minute it is moved to stage 2 where water is filled. Now, in stage 1 nothing is happening. Likewise, when the bottle is in stage 3 both stage 1 and stage 2 are inactive. But in pipelined operation, when the bottle is in stage 2, the bottle in stage 1 can be reloaded. In the same way, during the bottle 3 there could be one bottle in the 1st and 2nd stage accordingly. Therefore at the end of stage 3, we receive a new bottle for every minute. Hence, the average time taken to manufacture 1 bottle is:

Arithmetic Pipeline and Instruction Pipeline

1. Arithmetic Pipeline :

An arithmetic pipeline divides an arithmetic problem into various sub problems for execution in various pipeline segments. It is used for floating point operations, multiplication and various other computations. The process or flowchart arithmetic pipeline for floating point addition is shown in the diagram.Floating point addition using arithmetic pipeline :

The following sub operations are performed in this case:

1.Compare the exponents.

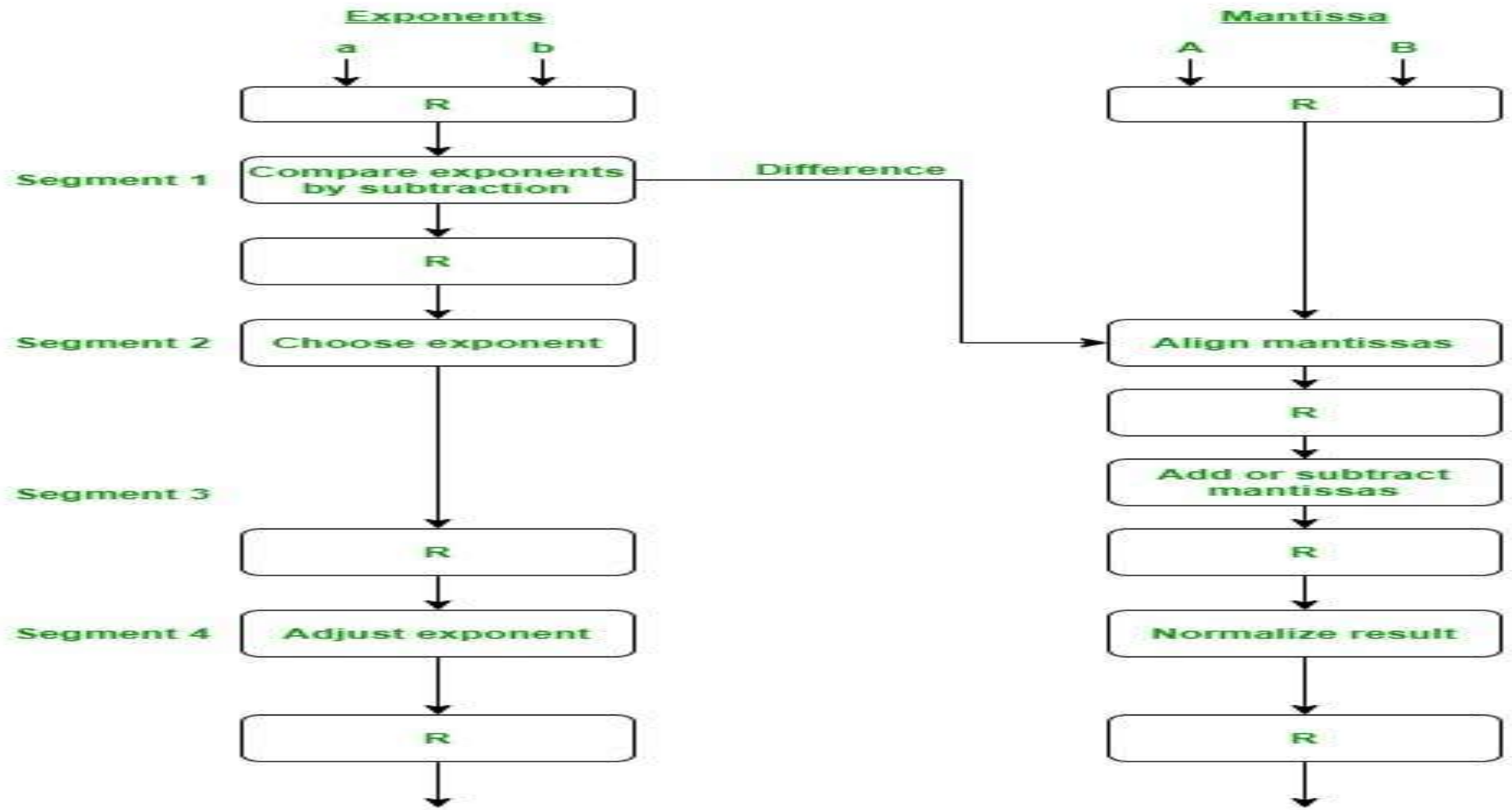
2.Align the mantissas.

3.Add or subtract the mantissas.

4.Normalise the result

First of all the two exponents are compared and the larger of two exponents is chosen as the result exponent. The difference in the exponents then decides how many times we must shift the smaller exponent to the right. Then after shifting of exponent, both the mantissas get aligned. Finally the addition of both numbers take place followed by normalisation of the result in the last segment.

Pipeline Organization for Floating point addition and subtraction

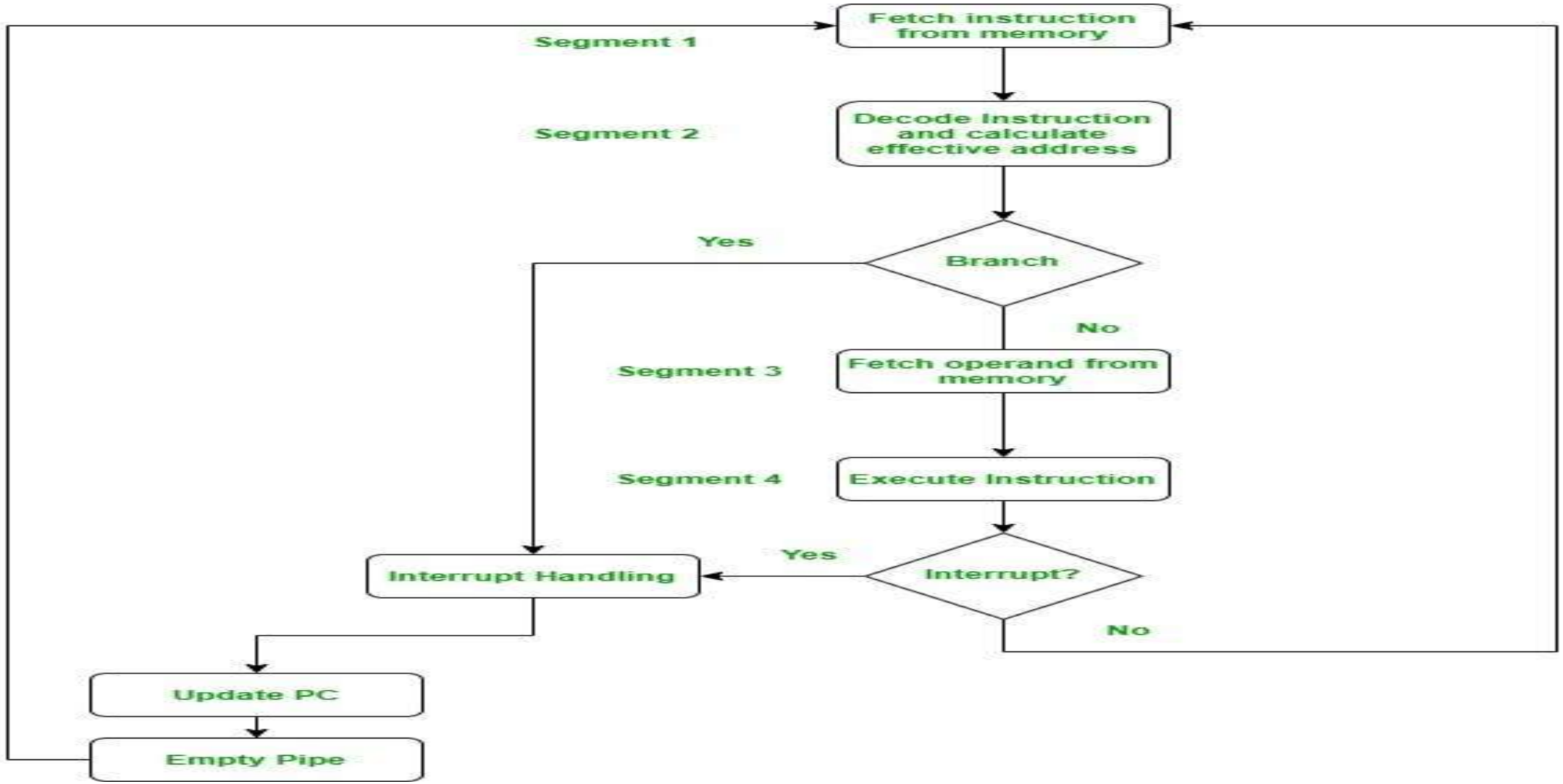


2. Instruction Pipeline :

In this a stream of instructions can be executed by overlapping fetch, decode and execute phases of an instruction cycle. This type of technique is used to increase the throughput of the computer system. An instruction pipeline reads instruction from the memory while previous instructions are being executed in other segments of the pipeline. Thus we can execute multiple instructions simultaneously. The pipeline will be more efficient if the instruction cycle is divided into segments of equal duration. In the most general case computer needs to process each instruction in following sequence of steps:

1) Fetch the instruction from memory (FI)
2) Decode the instruction (DA)
3) Calculate the effective address
4) Fetch the operands from memory (FO)
5) Execute the instruction (EX)
6) Store the result in the proper place

The flowchart for instruction pipeline is shown below.


Segment 1
Fetch instruction from memory
Segment 2
Decode Instruction and calculate effective address
Yes
Branch
No
Segment 3
Fetch operand from memory
Segment 4
Execute Instruction
Yes
Interrupt Handling
Interrupt?
No
Update PC
Empty Pipe

Types of Pipeline Hazards in Computer Architecture

The three different types of hazards in computer architecture are:

1. Structural
2. Data
3. Control

Structural Hazard

Structural hazards occur when **hardware resources are insufficient** to support all pipeline stages simultaneously.

Techniques to Handle Structural Hazards

✧ **Duplicating Hardware Resources**

Example: separate instruction & data memory (Harvard architecture).

✧ **Adding More Functional Units**

Multiple ALUs, multiple load/store units, etc.

✧ **Resource Scheduling**

Pipeline schedules access to resources at different cycles.

✧ **Stalling**

Hold some instructions until the resource is free.

Data Hazards

Data hazards occur when an instruction depends on the result of a previous instruction that has not yet completed.

Types of Data Hazards

- **RAW (Read After Write)** – True dependency
- **WAR (Write After Read)** – Anti-dependency
- **WAW (Write After Write)** – Output dependency

Techniques to Handle Data Hazards

✧ **Forwarding / Bypassing**

Uses ALU output directly instead of waiting for register write.

✧ **Stalling (Pipeline Interlock)**

Pipeline is paused until data is available.

✧ **Register Renaming**

Avoids WAR and WAW hazards by assigning different physical registers.

✧ **Compiler Scheduling / Instruction Reordering**

Compiler rearranges instructions to avoid dependencies.

Control Hazards

Control hazards occur due to **branch instructions** where the next instruction address depends on a condition.

Techniques to Handle Control Hazards

✧ **Stalling / Freezing the Pipeline**

Wait until the branch is resolved.

✧ **Branch Prediction**

- Static prediction (always taken / always not taken)
- Dynamic prediction (1-bit, 2-bit predictors, branch history table)

✧ **Branch Target Buffer (BTB)**

Stores predicted target addresses.

✧ **Delayed Branching**

Compiler places useful instructions in the branch delay slot.

✧ **Speculative Execution**

Processor executes predicted instructions ahead of time.

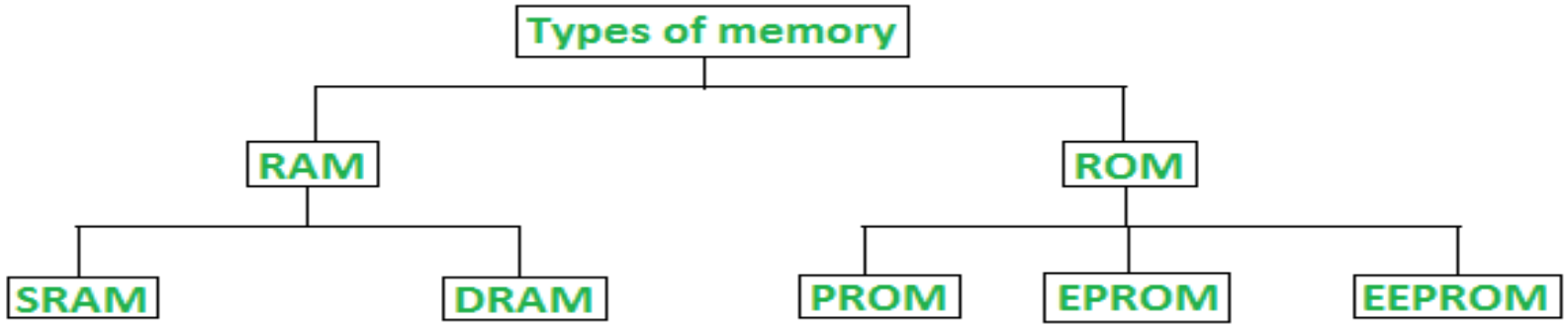
Unit 2

Memory

Memory is a fundamental component of computing systems, essential for performing various tasks efficiently. It plays a crucial role in how computers operate, influencing speed, performance, and data management. In the realm of computer memory, two primary types stand out: Random Access Memory (RAM) and Read-Only Memory (ROM). Understanding the difference between RAM and ROM is vital for appreciating their respective roles in a computer's operation. While RAM enables efficient processing and multitasking, ROM provides the necessary stability and reliability for system functions. Together, these memory types ensure that computers can operate smoothly and effectively in a digital landscape.

Computer memory is of two basic types:

Primary memory (RAM and ROM)
Secondary memory (Hard Drive, CD, etc)



Classification of computer memory

Random Access Memory (RAM)

Random Access Memory (RAM) is a type of computer memory that is used to temporarily store data that the computer is currently using or processing. RAM is volatile memory, which means that the data stored in it is lost when the power is turned off. RAM is typically used to store the operating system, application programs, and data that the computer is currently using.

- It is also called read-write memory or the main memory or the primary memory .
- The programs and data that the CPU requires during the execution of a program are stored in this memory.
- It is a volatile memory as the data is lost when the power is turned off.

Types of Random Access Memory (RAM)

1. **Static RAM**: SRAM stands for Static Random Access Memory. It is a type of semiconductor which is widely used in computing devices and microprocessors.

2.**Dynamic RAM**: DRAM stands for Dynamic Random Access Memory. It is made of Capacitors and has smaller data life span than Static RAM.

Read-Only Memory (ROM)

Read Only Memory (ROM) is a type of computer memory that is used to permanently store data that does not need to be modified. ROM is non-volatile memory, which means that the data stored in it is retained even when the power is turned off. ROM is typically used to store the computer's BIOS (basic input/output system), which contains the instructions for booting the computer, as well as firmware for other hardware devices.

- Stores crucial information essential to operate the system, like the program essential to boot the computer.
- It is non-volatile.
- Always retains its data.
- Used in embedded systems or where the programming needs no change.
- Used in calculators and peripheral devices.
- ROM is further classified into four types- M ROM , PROM , EPROM , and EEPROM .

Types of Read-Only Memory (ROM)

- PROM (Programmable Read-Only Memory)
- EPROM (Erasable Programmable Read Only Memory)
- EEPROM (Electrically Erasable Programmable Read Only Memory)
- MROM (Mask Read Only Memory)

Register transfer

Register transfer, memory transfer, and tri-state buffers are fundamental concepts in computer architecture for moving data: Register Transfer uses RTL to describe data flow between CPU registers (e.g., R1←R2), Memory Transfer moves data to/from RAM (read/write), and Tri-state Buffers enable multiple devices to share a bus by providing 'high-impedance' (disconnected) states, preventing signal conflicts on the shared lines. A common bus uses these buffers and decoders to route data efficiently, allowing only one register to output data at a time.

1. Register Transfer

Definition: Describes data movement between registers (like IR, PC, Accumulator) within the CPU using symbolic notation (Register Transfer Language - RTL).

Example: R1 ← R2 means data from Register 2 is copied to Register 1.

- Purpose: To define the basic operations (micro-operations) a processor performs, like addition, shifting, or loading.

2. Memory Transfer

Definition: Moving data between the CPU and main memory (RAM).

Operations:

- Read: Data from a memory address is fetched into a Memory Buffer Register (MBR).
- Write: Data from a register (like MBR) is stored into a specified memory location.

3. Tri-State Bus Buffer

Function: A special buffer with three output states: Logic 1, Logic 0, and High-Impedance (Hi-Z).

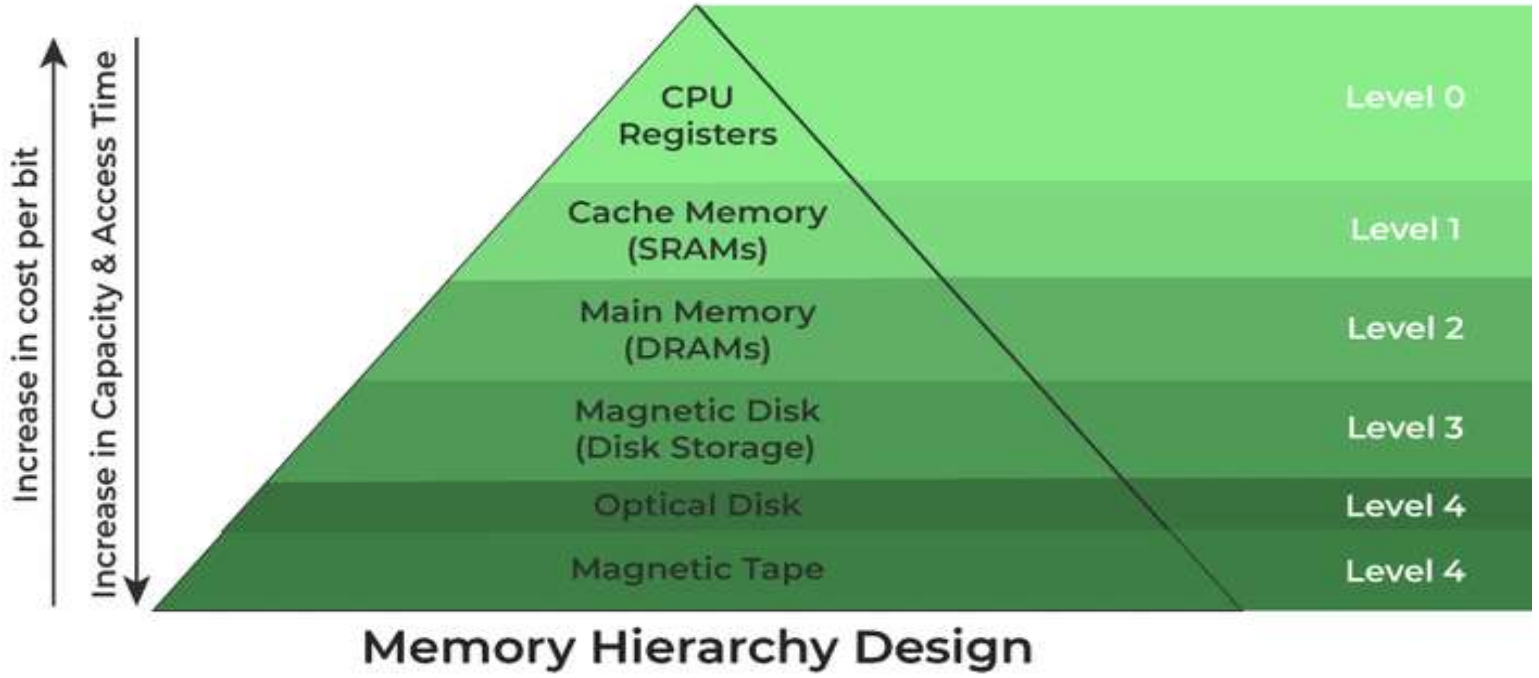
Operation:

- Enabled (Control=1): Acts as a normal buffer, passing the input signal to the output.
- Disabled (Control=0): Output goes to Hi-Z, effectively disconnecting it from the bus.

Memory Hierarchy Design

1. Registers

Registers are small, high-speed memory units located in the CPU. They are used to store the most frequently used data and instructions. Registers have the fastest access time and the smallest storage capacity, typically ranging from 16 to 64 bits.



Memory Hierarchy Design

2. Cache Memory

Cache memory is a small, fast memory unit located close to the CPU. It stores frequently used data and instructions that have been recently accessed from the main memory. Cache memory is designed to minimize the time it takes to access data by providing the CPU with quick access to frequently used data.

3. Main Memory

Main memory , also known as RAM (Random Access Memory), is the primary memory of a computer system. It has a larger storage capacity than cache memory, but it is slower. Main memory is used to store data and instructions that are currently in use by the CPU.

4. Secondary Storage

Secondary storage, such as hard disk drives (HDD) and solid-state drives (SSD) , is a non-volatile memory unit that has a larger storage capacity than main memory. It is used to store data and instructions that are not currently in use by the CPU. Secondary storage has the slowest access time and is typically the least expensive type of memory in the memory hierarchy.

5. Magnetic Disk

Magnetic Disks are simply circular plates that are fabricated with either a metal or a plastic or a magnetized material. The Magnetic disks work at a high speed inside the computer and these are frequently used.

6. Magnetic Tape

Magnetic Tape is simply a magnetic recording device that is covered with a plastic film. It is generally used for the backup of data. In the case of a magnetic tape, the access time for a computer is a little slower and therefore, it requires some amount of time for accessing the strip.

Mapping Technique in cache memory :Cache mapping techniques (Direct, Fully Associative, Set Associative) determine where main memory blocks reside in the cache, balancing speed, cost, and complexity

1. Direct Mapping

In direct mapping, each block of main memory maps to exactly one specific cache line. The mapping is done using a modulus operation.

Mapping Formula : Cache Line=(Block Number) mod (Number of Cache Lines)

Characteristics :

- Simplicity : Very simple to implement due to fixed mapping.
- Low Cost : Requires fewer hardware resources.
- Collision : High chance of conflict misses because multiple blocks compete for the same cache line.
- Tag Size : Relatively smaller, as each cache line corresponds to only one memory block.

2. Associative Mapping

In associative mapping, a memory block can be placed in any cache line. There is no fixed mapping; the placement is entirely flexible.

Characteristics :

- Flexibility : Any block can go into any cache line, reducing conflict misses.
- High Cost : Requires associative hardware to search all cache lines simultaneously (Content Addressable Memory or CAM).
- Replacement Policy : Requires an algorithm like LRU (Least Recently Used) or FIFO (First In, First Out) to handle cache replacements.
- Tag Size : Larger compared to direct mapping, as the tag must uniquely identify blocks in all cache lines.

3. Set-Associative Mapping

Set-associative mapping is a hybrid approach that combines aspects of direct and associative mapping. The cache is divided into sets, and each memory block maps to exactly one set. Within that set, a block can occupy any cache line.

Mapping Formula:

Set Number=(Block Number) mod (Number of Sets)

Characteristics:

- Flexibility in Sets : Reduces conflict misses compared to direct mapping.
- Moderate Cost : Balances cost and complexity between direct and associative mapping.
- Replacement Policy : Similar to associative mapping, replacement policies like LRU or FIFO are used within each set.
- Tag Size : Smaller than fully associative mapping, as the tag identifies blocks within a set.

Performance Implementation in Cache Memory

Cache Memory Basics

Cache memory is the small and fastest memory used in a computer system to store the most frequently used data and instructions. The cache memory is mainly used to bridge the gap between the speeds of the central processing unit and the main memory.

It is basically a volatile (temporary) memory that acts as a buffer between the CPU and the main memory, and stores subsets of recently used data and instructions that the CPU needs for processing in future. Whenever the CPU needs data and instructions, it first searches the cache memory before going to check the main memory. Cache memory is typically integrated into the CPU chip and works on the locality principle.

Cache Memory Performance

Cache memory performance can be evaluated by using several metrics. Some commonly used metrics are Cache Hit Ratio, Cache Latency, Cache Replacement Policies, and Cache Coherence. Let's briefly explore each of these metrics used to measure the cache memory performance.

Cache Hit Ratio

The cache hit rate is considered one of the most effective metrics to evaluate the cache memory performance. In this method, the performance of the cache memory is measured in terms of hit ratio. Where the hit ratio is the number of times the processor finds the desired data and instructions in the cache memory without accessing the main memory. A high hit ratio specifies an efficient and high-performance cache memory design.

The hit ratio can be determined by using the following expression,
HitRatio=Number of hits Total CPUreferences to memory ⇒HitRatio=Number of hits number of hits+number of misses to cache

Cache Latency

Cache latency is another important metric to measure cache memory performance. This aspect measures the time taken by the CPU to access data from the cache memory. A low cache latency indicates faster retrieval of data and improved performance of the system.

Cache Replacement Policies

The third important metric to measure the cache performance is cache replacement policies. These policies determine how the cache memory manages the new coming data when it is fully occupied. Some popular cache replacement policies are First-In-First-Out (FIFO), Least Recently Used (LRU), and Random Replacement (RR) Policies.

Cache Coherence

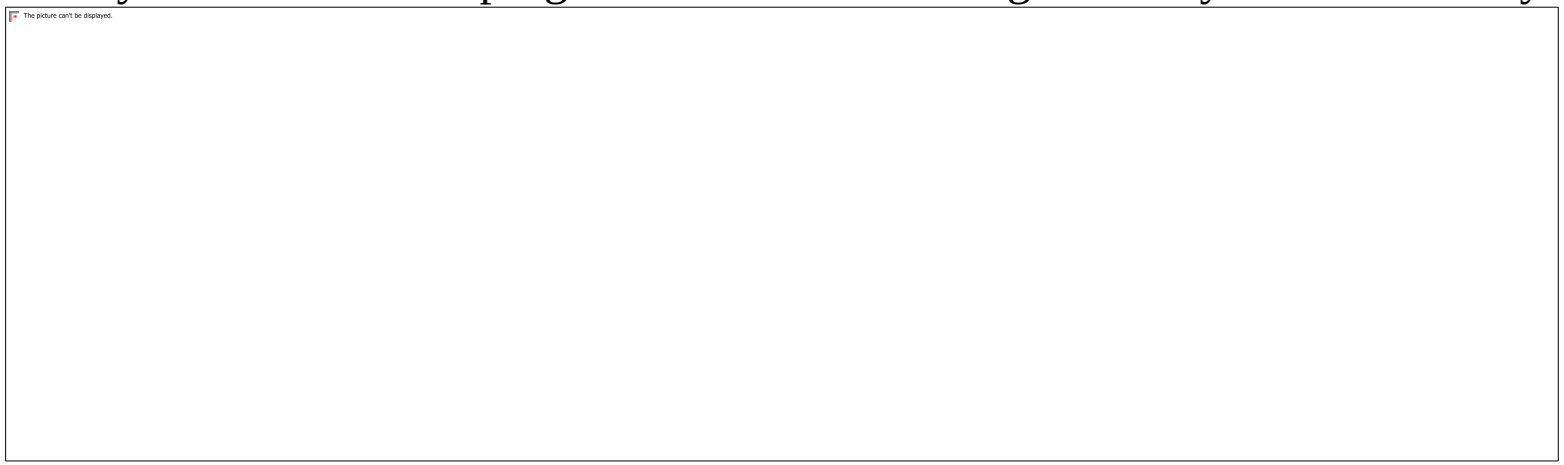
Another crucial metric used to measure the cache memory performance is cache coherence. It is more important in the case of a multi-processor system where each processor has its cache memory. Cache coherence is responsible for ensuring consistency in data stored by all the cache memories, resulting in the elimination of race conditions and improved performance of the system.

Virtual Memory Concepts

Virtual memory is a memory management technique in computer architecture that gives programs the illusion of having a large, contiguous memory space, even if physical RAM is limited, by using disk storage (SSD/HDD) as an extension. It works by swapping pages of a process between RAM and disk, managed by the Memory Management Unit (MMU), allowing larger programs to run, simplifying programming, and improving multitasking by only loading necessary parts into fast RAM, though slower disk access introduces latency

Objectives of Virtual Memory

- A program doesn't need to be fully loaded in memory to run. Only the needed parts are loaded.
- Programs can be bigger than the physical memory available in the system.
- Virtual memory creates the illusion of a large memory, even if the actual memory (RAM) is small.
- It uses both RAM and disk storage to manage memory, loading only parts of programs into RAM as needed.
- This allows the system to run more programs at once and manage memory more efficiently.



Types of Virtual Memory

In a computer, virtual memory is managed by the Memory Management Unit (MMU), which is often built into the CPU. The CPU generates virtual addresses that the MMU translates into physical addresses. There are two main types of virtual memory:

- Paging
- Segmentation

1. Paging

Paging divides memory into small fixed-size blocks called pages. When the computer runs out of RAM, pages that aren't currently in use are moved to the hard drive, into an area called a swap file. Here,

- The swap file acts as an extension of RAM.
- When a page is needed again, it is swapped back into RAM, a process known as page swapping.
- This ensures that the operating system (OS) and applications have enough memory to run.

2. Segmentation

Segmentation divides virtual memory into segments of different sizes. Segments that aren't currently needed can be moved to the hard drive. Here,

- The system uses a segment table to keep track of each segment's status, including whether it's in memory, if it's been modified and its physical address.
- Segments are mapped into a process's address space only when needed.

Page Replacement

When memory is over-allocated, page faults can occur. If a process requires a page that is not in memory, the operating system must find a free frame. However, if all frames are in use, the operating system must use a page replacement algorithm to select an existing page to replace.

Page replacement is a key feature of demand paging, completing the separation between logical and physical memory.

To implement demand paging, two critical problems must be addressed:

1.Frame-allocation algorithm: Deciding how to allocate frames to processes.

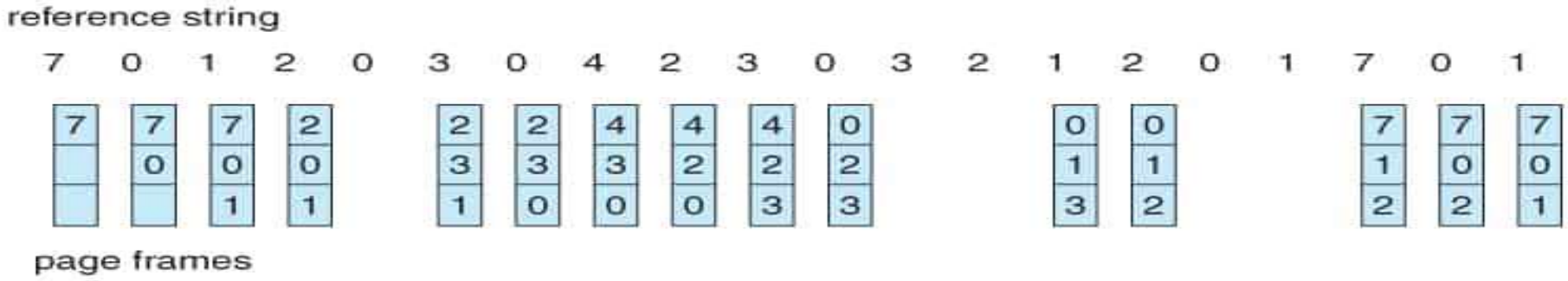
2.Page-replacement algorithm: Choosing which pages to replace when new pages need to be loaded.

Page Replacement Algorithms

Various page-replacement algorithms exist to determine which page to swap out when a page fault occurs. These include:

FIFO Page Replacement

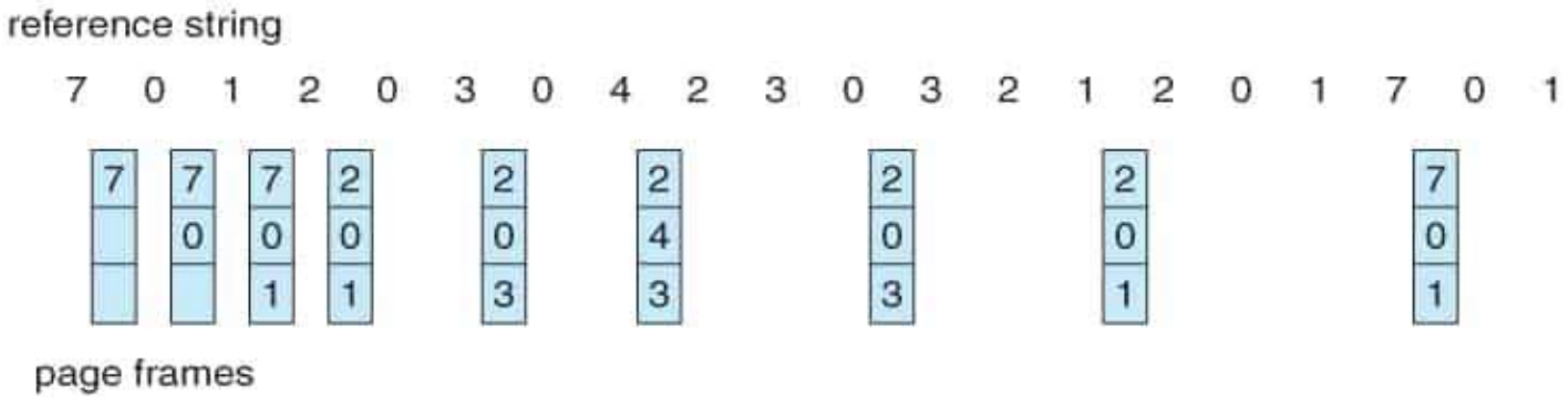
The simplest page-replacement algorithm is First-In-First-Out (FIFO). The FIFO algorithm associates each page with the time it was brought into memory. When a page must be replaced, the oldest page is selected.



Optimal Page Replacement

The optimal page-replacement algorithm selects the page that will not be used for the longest time in the future. This approach guarantees the lowest possible page-fault rate but requires knowledge of future page references, which makes it impractical for real-world systems. As such, it is typically used for comparison studies.

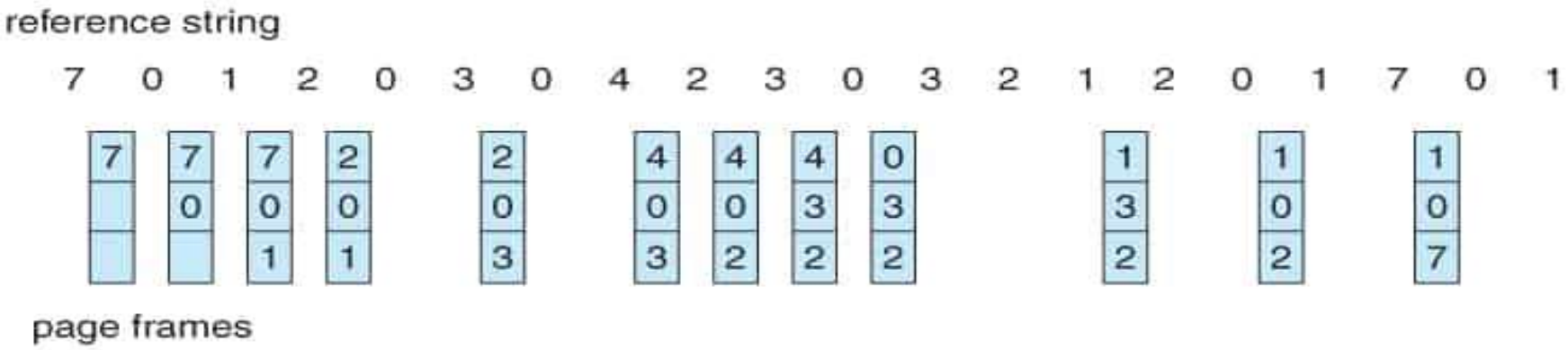
Optimal Page Replacement Algorithm



LRU Page Replacement

The Least Recently Used (LRU) algorithm approximates the optimal page-replacement strategy. It replaces the page that has not been used for the longest period of time, using past access patterns to predict future behavior.

LRU Page Replacement Algorithm



Multiprocessor architecture

Multiprocessor architectures are classified by Flynn's Taxonomy (SISD, SIMD, MISD, MIMD) based on single/multiple instruction/data streams, with MIMD (Multiple Instruction, Multiple Data) being the dominant form for modern parallel systems, encompassing shared-memory (easier programming) and distributed-memory (more scalable) models, all aiming for increased speed through parallel processing.

Flynn's Taxonomy

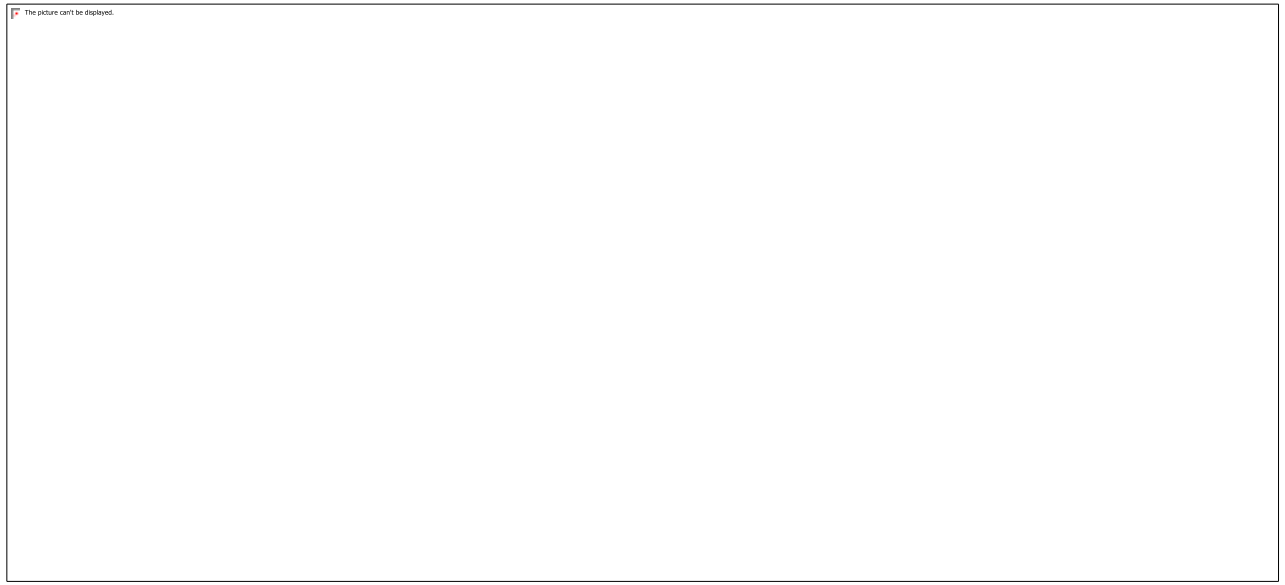
Flynn's Taxonomy classifies computer architectures according to how many instruction streams (processes) and data streams they can process simultaneously, dividing them into four categories: SISD, SIMD, MISD, and MIMD.



Single-Instruction, Single-Data (SISD) Systems

An SISD computing system is a uniprocessor machine which is capable of executing a single instruction, operating on a single data stream. In SISD, machine instructions are processed in a sequential manner, and computers adopting this model are popularly called sequential computers. Most conventional computers have SISD architecture. All the instructions and data to be processed have to be stored in primary memory.

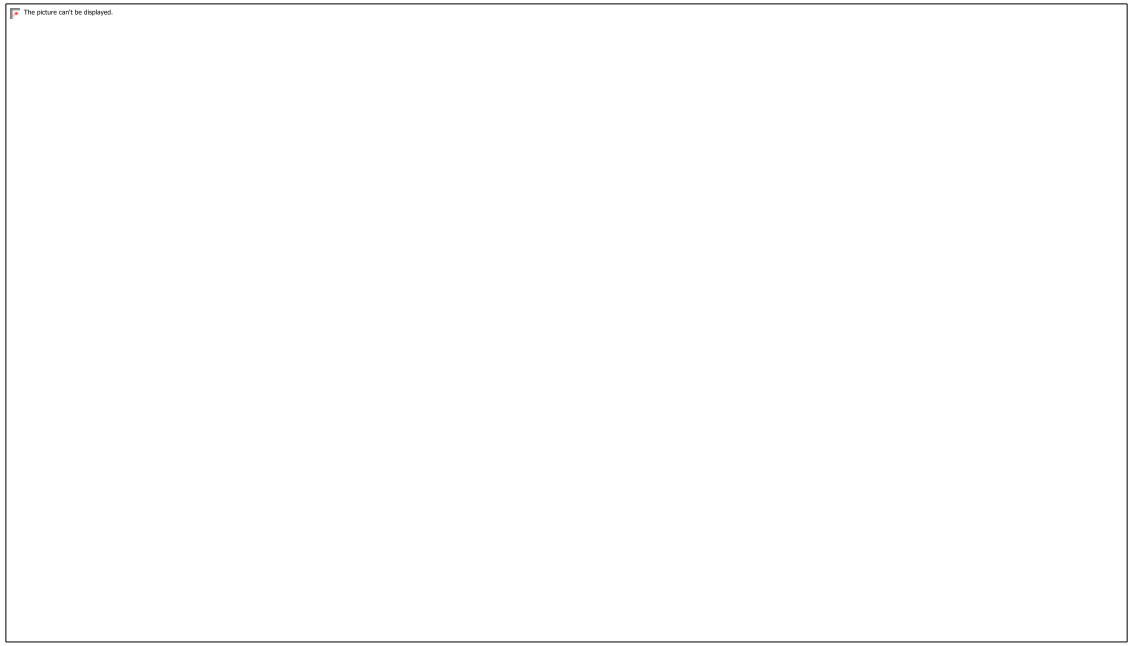
The speed of the processing element in the SISD model is limited(dependent) by the rate at which the computer can transfer information internally. Dominant representative SISD systems are IBM PC, workstations.



Single-Instruction, Multiple-Data (SIMD) Systems

An SIMD system is a multiprocessor machine capable of executing the same instruction on all the CPUs but operating on different data streams. Machines based on an SIMD model are well suited to scientific computing since they involve lots of vector and matrix operations.

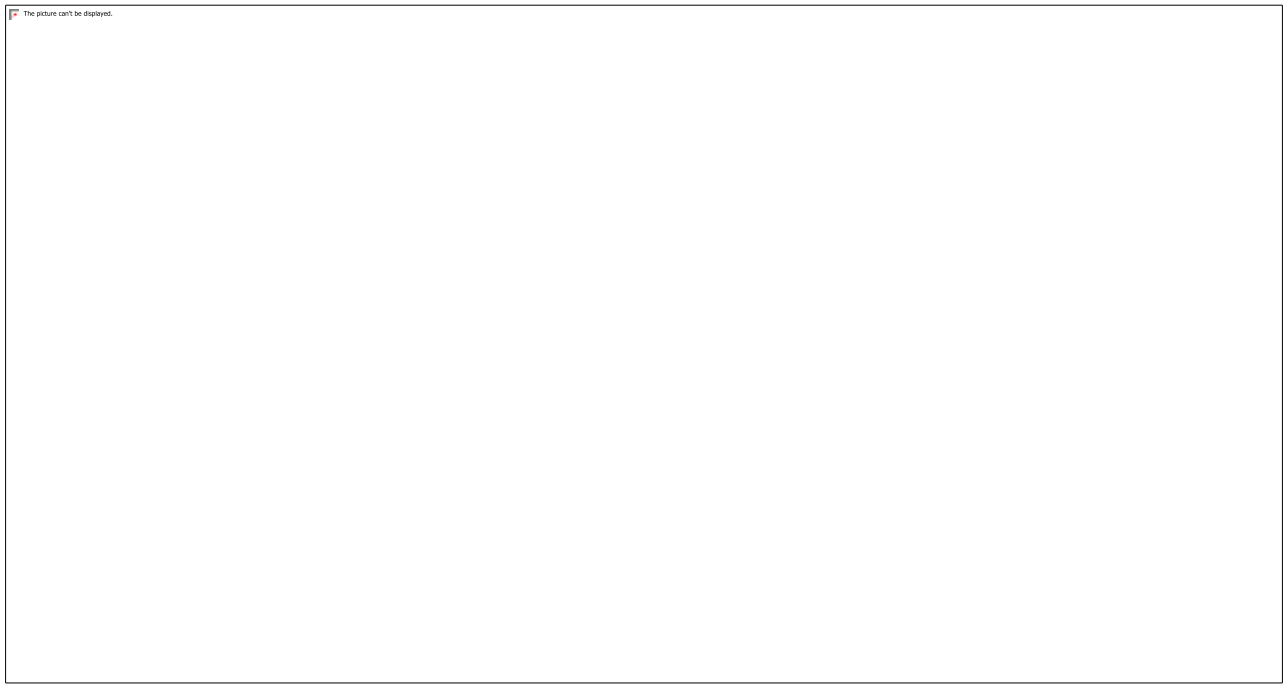
So that the information can be passed to all the processing elements (PEs) organized data elements of vectors can be divided into multiple sets(N-sets for N PE systems) and each PE can process one data set. Dominant representative SIMD systems is Cray's vector processing machine.



Multiple-Instruction, Single-Data (MISD) systems

An MISD computing system is a multiprocessor machine capable of executing different instructions on different PEs but all of them operating on the same dataset.

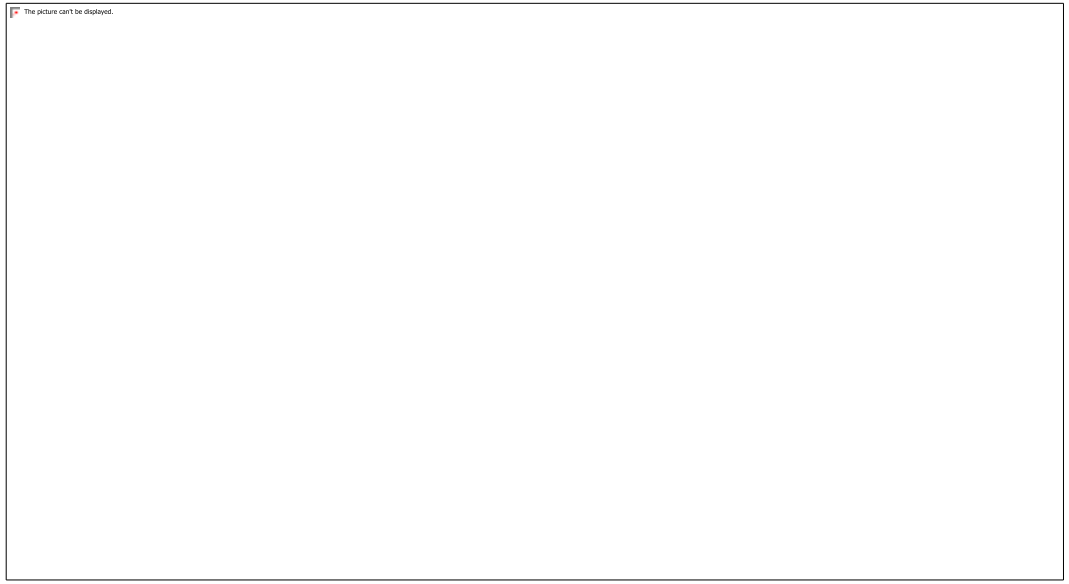
Example Z = sin(x)+cos(x)+tan(x) The system performs different operations on the same data set. Machines built using the MISD model are not useful in most of the application, a few machines are built, but none of them are available commercially.



Multiple-Instruction, Multiple-Data (MIMD) Systems

An MIMD system is a multiprocessor machine which is capable of executing multiple instructions on multiple data sets. Each PE in the MIMD model has separate instruction and data streams; therefore machines built using this model are capable to any kind of application. Unlike SIMD and MISD machines, PEs in MIMD machines work asynchronously.

MIMD machines are classified into shared-memory and distributed-memory models depending on how processors connect to memory. In a shared-memory MIMD system (tightly coupled), all processors use the same global memory, and communication happens through it. Any change made by one processor is visible to all others. Examples include Silicon Graphics and Sun/IBM's SMP system



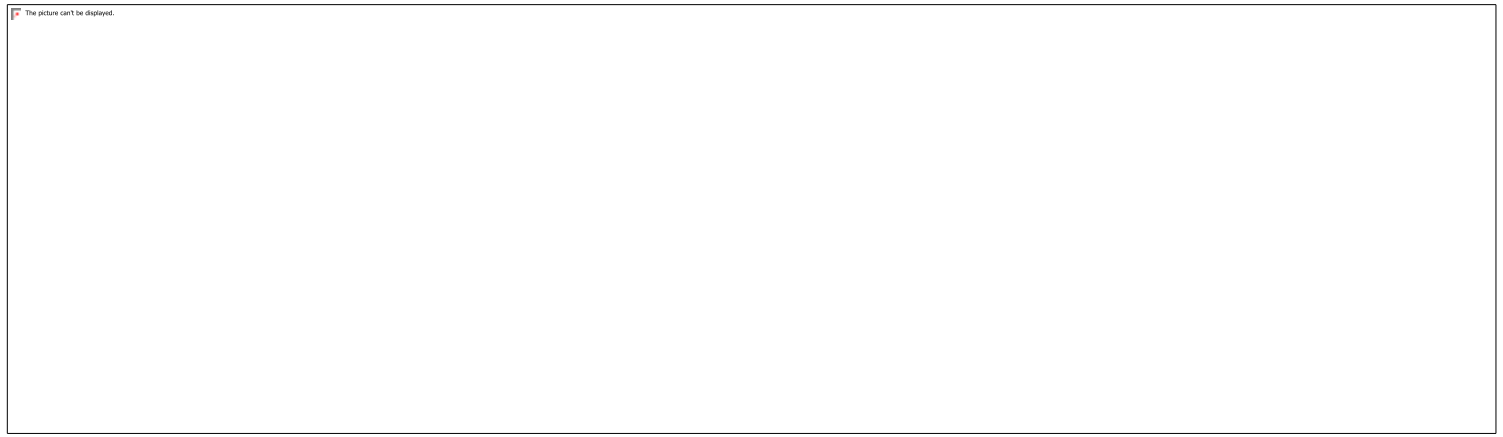
Unit 3

- **Operating System:Functionalities of Operating System**

1. Process Management

Process management in operating system is about managing processes. A Process is a running program. The life cycle of process is from the moment program start until it finishes.

Process State Diagram

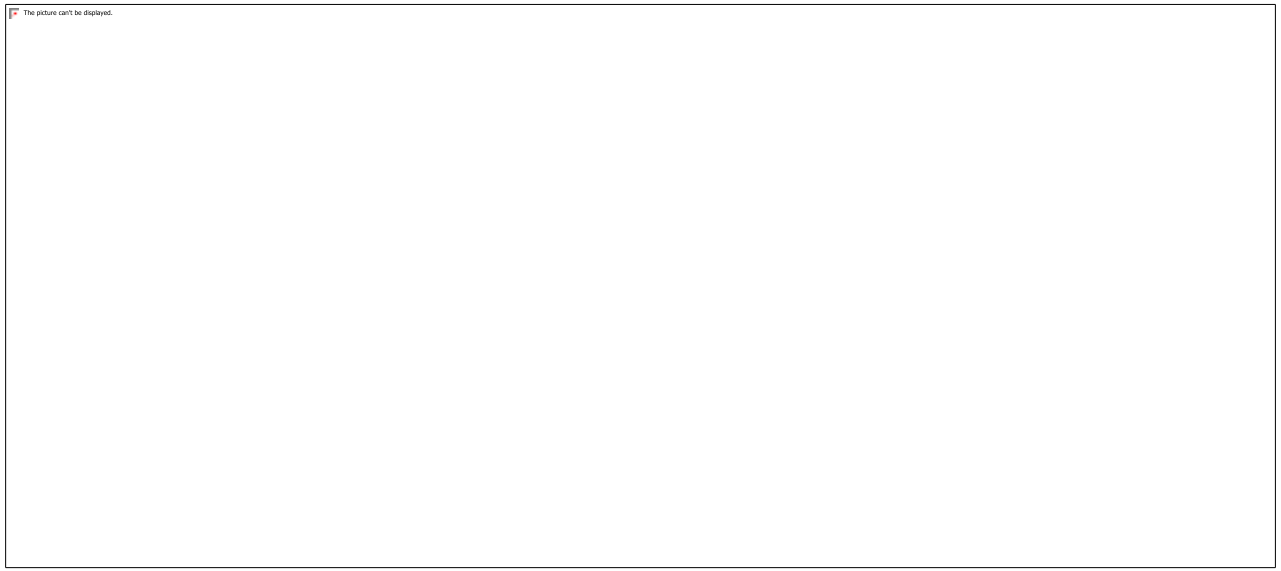


Core Functions in Process Management:

- Scheduling: Decides which process uses the CPU next (using algorithms like Round Robin or Priority Scheduling).
- Synchronization: Ensures orderly execution using locks, semaphores, or monitors to prevent race conditions.
- Deadlock Handling: Detects and prevents situations where processes wait forever for resources.
- Inter-Process Communication (IPC): Allows processes to exchange data via shared memory or message passing.

2. Memory Management

Memory management in an operating system controls how data is stored and organized in main (primary) memory and secondary storage, ensuring programs get the memory they need and freeing it when no longer used.



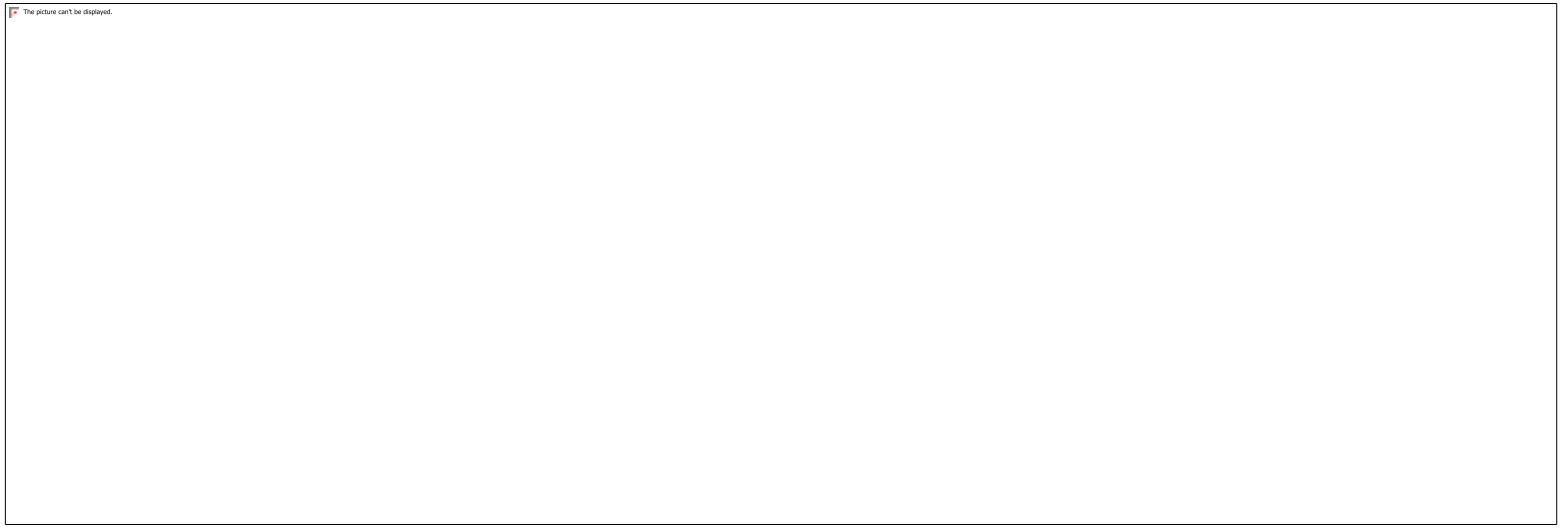
Memory Management

Key Activities in Memory Management:

- Allocation & Deallocation: Assigns and frees memory as needed.
- Protection: Prevents processes from interfering with each other’s memory..
- Virtual Memory: Uses disk space as extra memory to run large programs.
- Fragmentation Handling: Reduces wasted space through compaction.
- Disk Management: Handles file storage, free space, disk scheduling, and backups.

3. File System Management

File management in an operating system organizes and controls how data is stored, named, and accessed on storage devices, making it easy for users to find and use their files.



File System Management

File System Management includes managing of:

- File Attributes: Name, type, size, and permissions.
- File Types: Text, binary, and executable files.
- Operations: Create, read, write, and delete files.
- Access Methods: Sequential, direct, and indexed access for faster retrieval.

4. Device Management (I/O System)

Device management in an operating system controls how the computer communicates with hardware devices like printers or disks, using drivers and techniques to ensure smooth and efficient operation.



Device Management

Major components in Device Management:

- Device Drivers: Interface between hardware and OS.
- Buffering & Caching: Temporarily store data to match device speeds and improve performance.
- Spooling: Spooling manages data waiting to be processed, particularly in devices like printers. The OS places print jobs in a spool (a temporary storage area), allowing the CPU to continue other tasks while the printer works through the queue.

5. Protection and Security

Protection and security mechanisms in an OS are designed to safeguard system resources from unauthorized access or misuse. These mechanisms control which processes or users can access specific resources (such as memory, files, and CPU time). It also ensures that only authorized users can perform specific actions.



Protection & Security

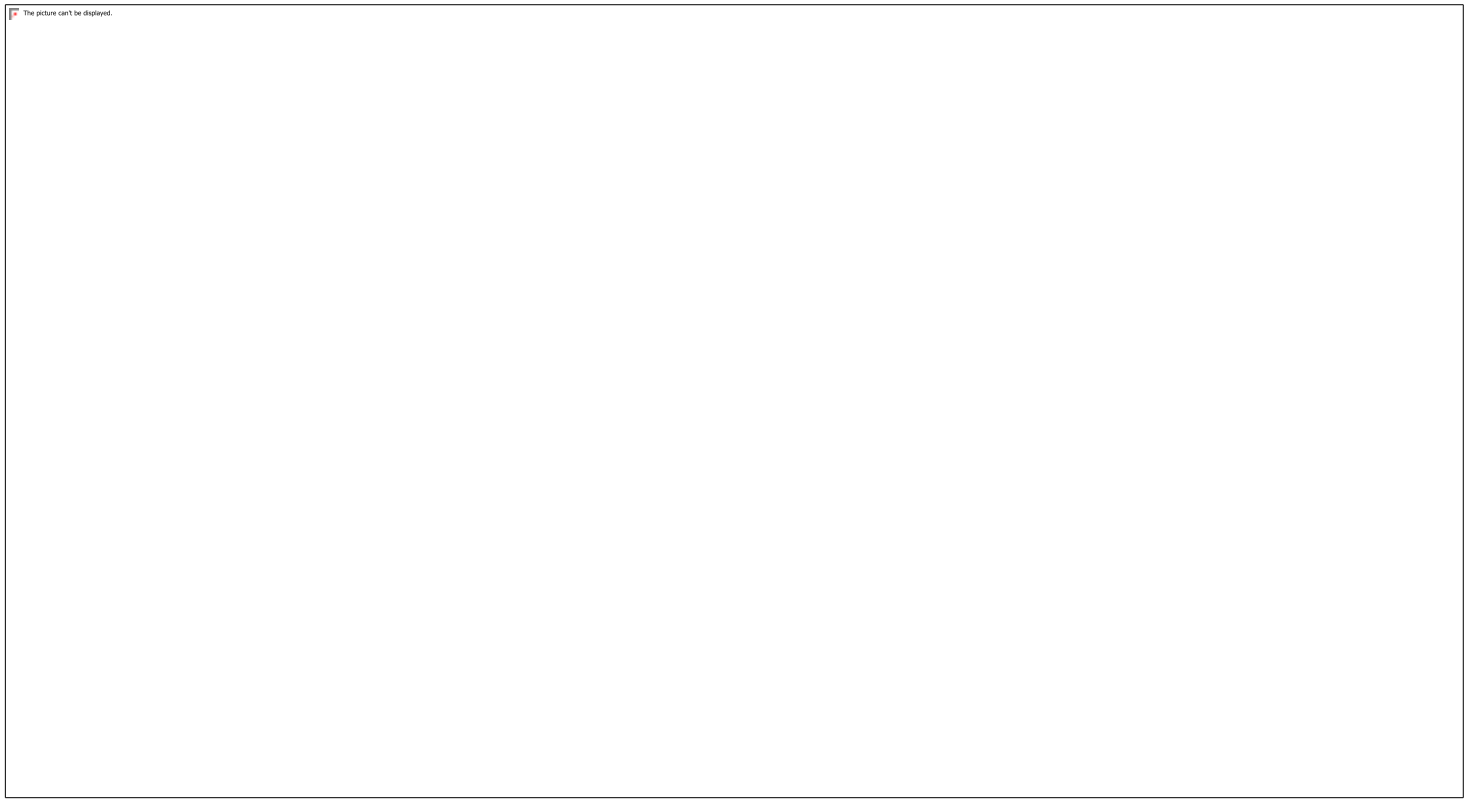
The OS protects system resources and user data from unauthorized access and attacks.

- Access Control: Limits user and process permissions.
- Authentication: Verifies users through credentials (UIDs/SIDs).
- Resource Protection: Prevents misuse of files, memory, or devices.
- Security: Guards against malware, denial-of-service attacks, and data theft

Types of Operating Systems

An operating system (OS) is software that manages computer hardware and software resources. It acts as a bridge between users and the computer, ensuring smooth operation. Different types of OS serve different needs; some handle one task at a time, while others manage multiple users or real-time processes.

Types of Operating Systems



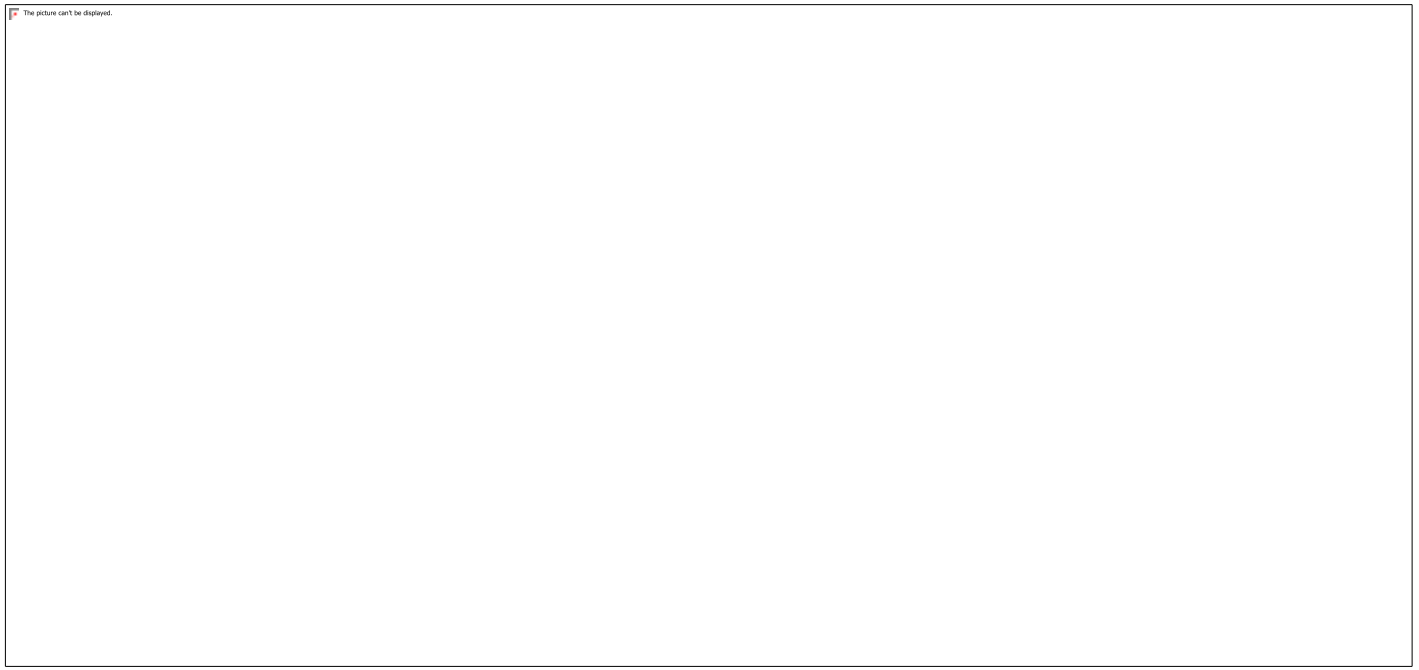
1. Batch Operating System

A Batch Operating System is designed to handle large groups of similar jobs efficiently. It does not interact with the computer directly but instead processes jobs that are grouped by an operator. These jobs are queued and executed one after the other, without user interaction during the process.

Processes groups of similar jobs automatically in batches without user interaction.

Examples:

- Payroll Systems
- Bank Statements



2. Multi-Programming Operating System

In a Multi-Programming Operating System, multiple programs run in memory at the same time. The CPU switches between programs, utilizing its resources more effectively and improving overall system performance.

Runs multiple programs in memory at the same time to maximize CPU usage.

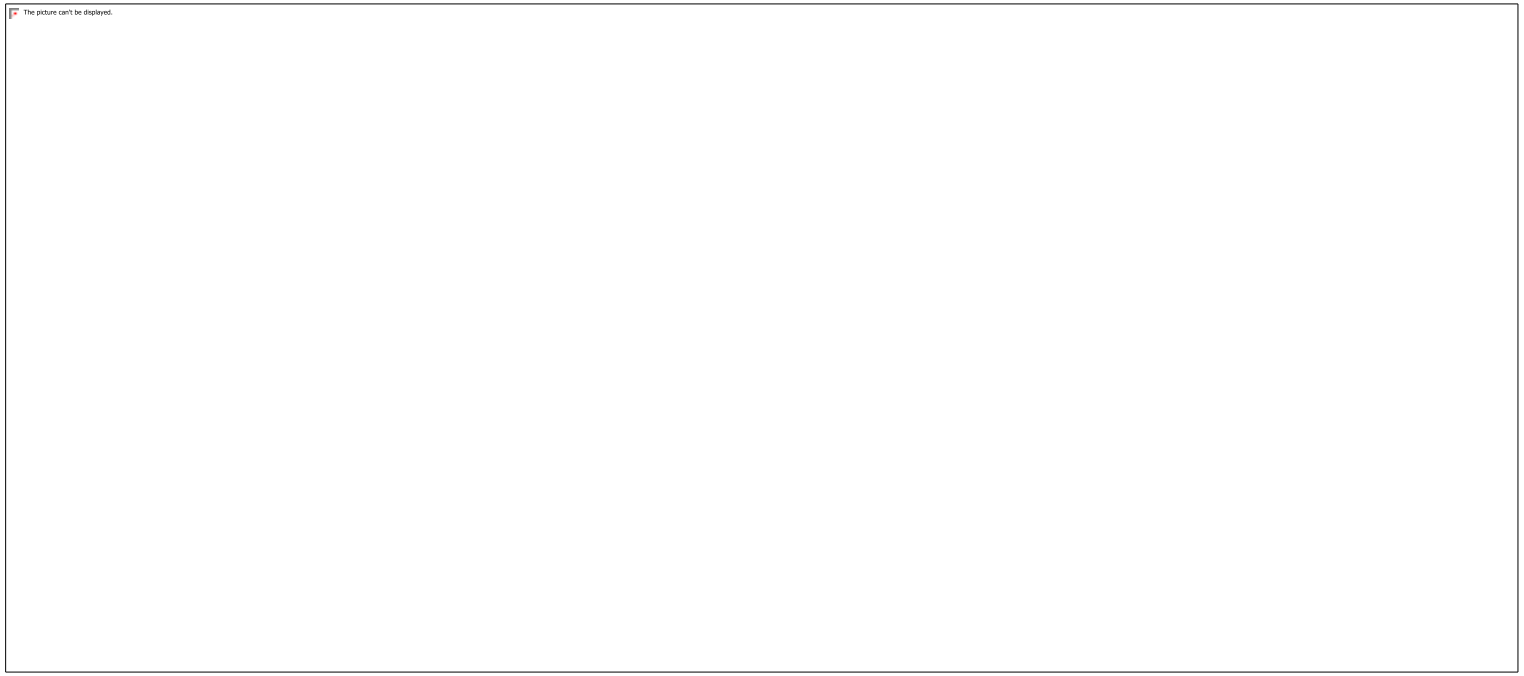
- Banking systems
- Railway servers
- Billing machines



3. Multi-tasking/Time-sharing Operating systems

Multitasking OS is a type of multiprogramming system where each process runs in a round-robin manner. Every task gets a fixed time slice called a quantum. After the quantum ends, the OS switches to the next task, allowing multiple tasks whether from one user or many to run smoothly on a single system.

Allows multiple tasks to run by giving each a small time slice for smooth, shared system use.

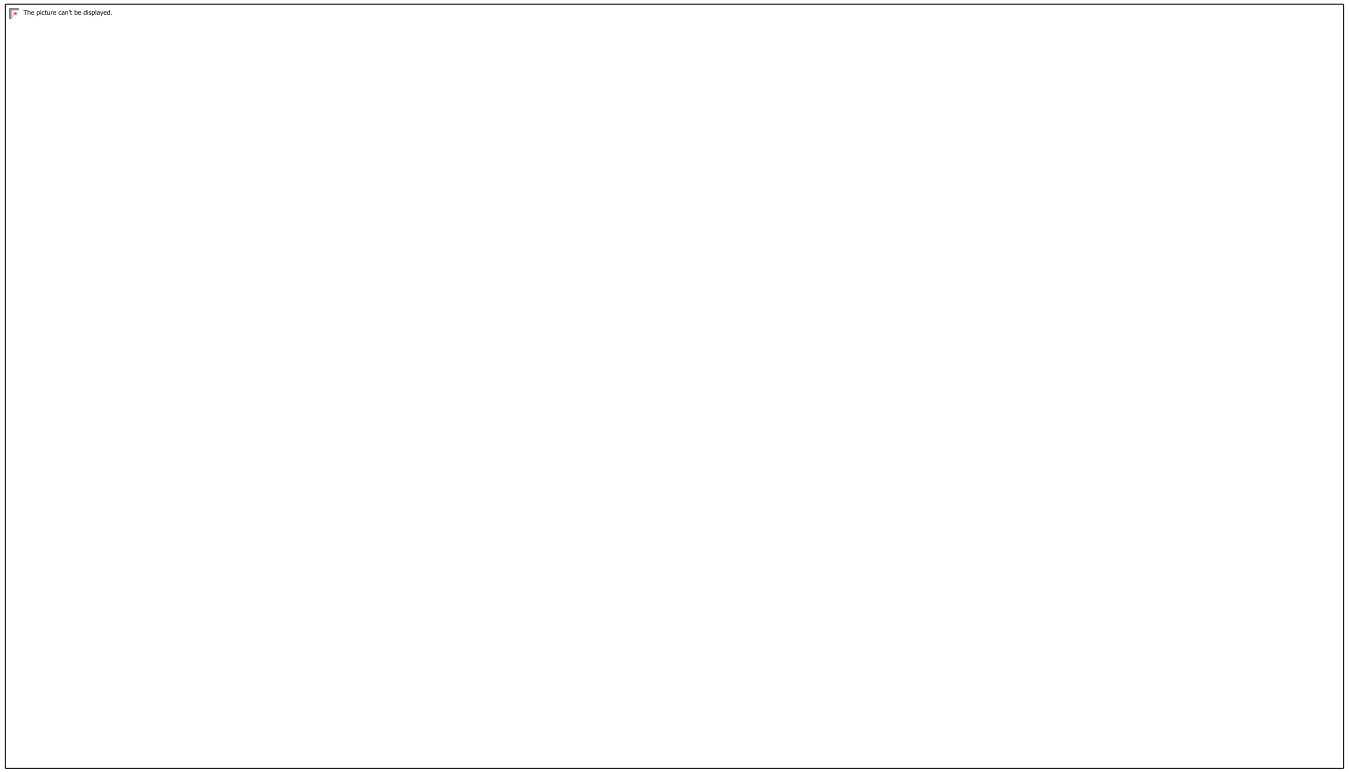


4. Multi-Processing Operating System

A Multi-Processing Operating System is a type of Operating System in which more than one CPU is used for the execution of resources. It betters the throughput of the System.

Uses two or more CPUs simultaneously to increase speed and reliability.

- UNIX
- Linux (Ubuntu, Red Hat, Debian)
- macOS



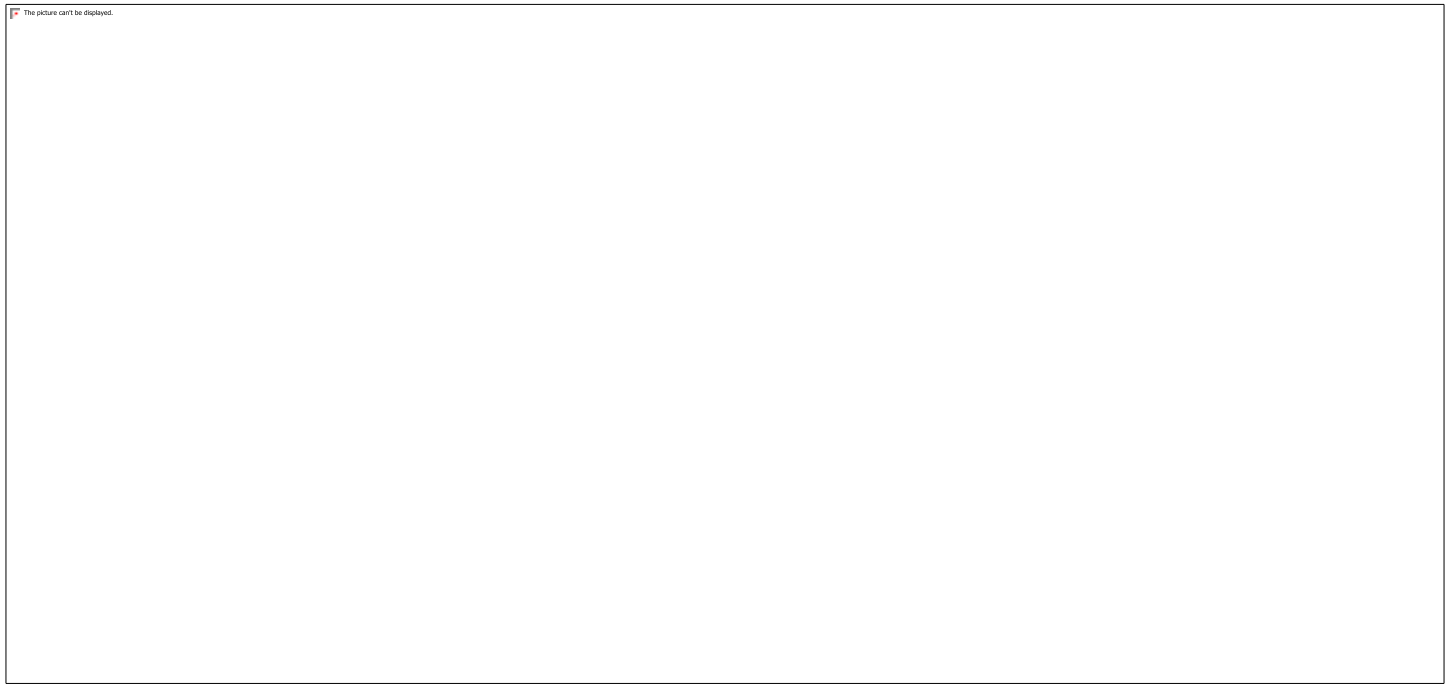
5. Distributed Operating System

Distributed operating systems connects multiple independent computers through a shared communication network. Each system has its own CPU and memory but works together as a single unit. The main benefit is remote access, allowing users to use files and software stored on other connected systems.

Connects multiple independent computers to function as a single coordinated system.

Examples:

- LOCUS
- MICROS
- Amoeba



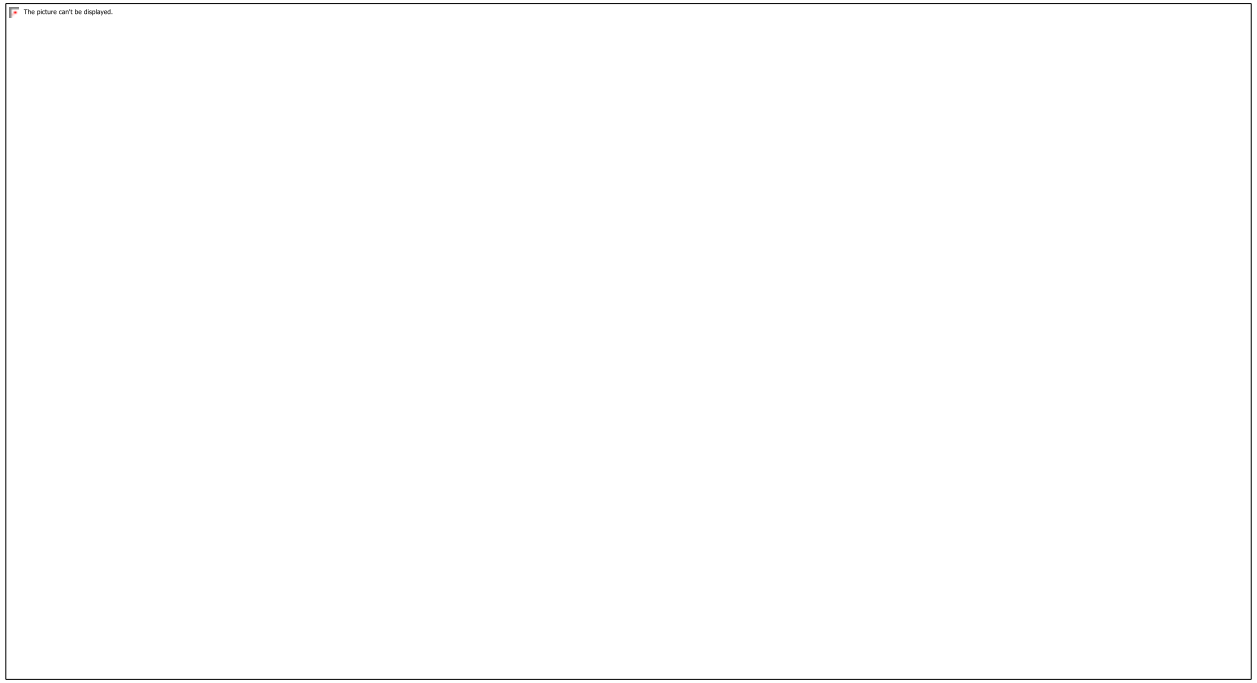
6. Network Operating System

A Network Operating System (NOS) runs on a server and manages data, users, security, applications, and other network functions. It allows shared access to files, printers, and resources within a small private network. Users can see the configuration and connections of other users, which is why these systems are considered tightly coupled systems.

Manages and supports users, data, and security across connected computers within a network.

Examples:

- Microsoft Windows Server 2003
- UNIX, Linux
- Mac OS X



Processes are the fundamental unit of work in an operating system, representing a program during its execution. The OS manages processes through various states, a dedicated data structure called the Process Control Block (PCB), and scheduling algorithms, while processes interact with the OS primarily via system calls.

Concept of Processes

A process is an active entity that is an instance of a program in execution, in contrast to a program which is a passive set of instructions stored on disk.

Each process in memory is typically divided into four sections:

- Text section: The executable code.
- Data section: Global and static variables.
- Stack: Temporary data storage for function parameters, return addresses, and local variables.
- Heap: Memory that is dynamically allocated during runtime.

Process States

As a process executes, it changes states. The common states in a 5-state model are:

- New: The process is being created and has not yet been loaded into the main memory.
- Ready: The process is in main memory, waiting to be assigned to a processor.
- Running: Instructions are being executed by the CPU. Only one process can be in this state per CPU core at a time.
- Waiting (Blocked): The process is paused, waiting for some event to occur, such as the completion of an I/O operation or user input.
- Terminated: The process has finished execution, and the operating system is removing it from memory.

Process Control Block (PCB)

The Process Control Block (PCB) is a data structure maintained by the OS for each process, acting as a repository for all the information needed to manage it.

The PCB holds essential details such as the Process ID, current state, program counter, CPU register values, scheduling information, memory management data, accounting statistics, and I/O status information.

Process Scheduling

Process scheduling is the OS's method for determining which process gains access to the CPU and when, aiming to optimize CPU usage, throughput, response time, and waiting time. Different types of schedulers exist: the long-term scheduler moves processes from storage to memory, the short-term scheduler allocates the CPU to processes in memory, and the medium-term scheduler swaps processes between memory and storage. Common scheduling algorithms include First-Come, First-Served (FCFS), Shortest Job First (SJF), Priority Scheduling, and Round Robin (RR), each with different approaches to selecting the next process for execution.

Co-operating and Independent Processes

Processes can be either independent or co-operating. An independent process operates without affecting or being affected by others, with predictable results. Co-operating processes, however, can interact and potentially share resources, requiring synchronization mechanisms to ensure correct execution.

Suspended Process

A suspended process is temporarily moved from main memory to secondary storage, often to manage memory resources. A process can be in a suspend ready state (ready to run but swapped out) or a suspend wait state (waiting for an event and swapped out).

Threads: An Overview

A thread represents a single sequence of execution within a process. It is often referred to as a "lightweight process" because it shares resources (code, data, open files) with other threads within the same process but maintains its own program counter, register set, and stack.

Benefits of Threads

Concurrency within a Process:

Threads enable multiple parts of a single application to execute concurrently, improving responsiveness and perceived performance.

Reduced Context Switching Time:

Switching between threads within the same process is generally faster and less resource-intensive than switching between separate processes.

Efficient Communication:

Threads within a process can easily share data and communicate through shared memory, simplifying inter-thread communication compared to inter-process communication.

Resource Utilization:

Threads can effectively utilize multi-processor or multi-core architectures by allowing different threads to run in parallel on different cores.

Responsiveness:

In applications with user interfaces, threads can keep the UI responsive by performing background tasks in separate threads.

User-Level Thread

The User-level Threads are implemented by the user-level software. These threads are created and managed by the thread library, which the operating system provides as an API for creating, managing, and synchronizing threads. it is faster than the kernel-level threads, it is basically represented by the program counter, stack, register, and PCB.

- User-level threads are typically employed in scenarios where fine control over threading is necessary, but the overhead of kernel threads is not desired.
- They are also useful in systems that lack native multithreading support, allowing developers to implement threading in a portable way.

Example: User threads library includes POSIX threads, Mach C-Threads

Kernel-Level Thread

Threads are the units of execution within an operating system process. The OS kernel is responsible for generating, scheduling, and overseeing kernel-level threads since it controls them directly.

- The Kernel-level threads are directly handled by the OS directly whereas the thread's management is done by the kernel.
- Each kernel-level thread has its own context, including information about the thread's status, such as its name, group, and priority.

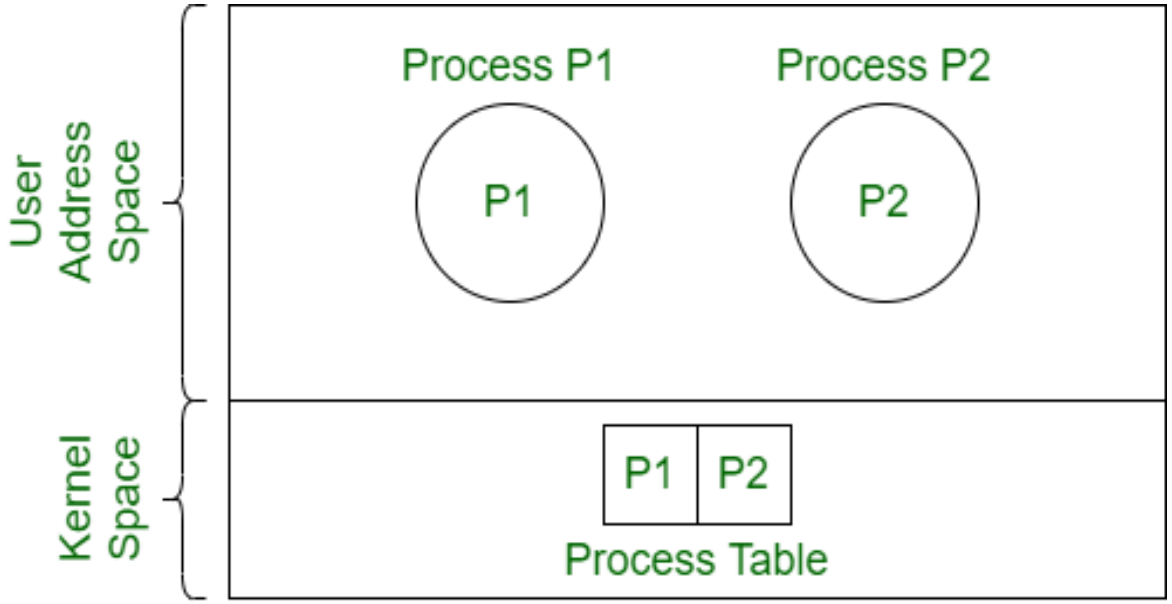
Example: The example of Kernel-level threads are Java threads, POSIX thread on Linuxs, etc.

Thread Models in Operating System

A thread is a light weight process which is similar to a process where every process can have one or more threads. Each thread contains a Stack and a Thread Control Block. There are four basic thread models :

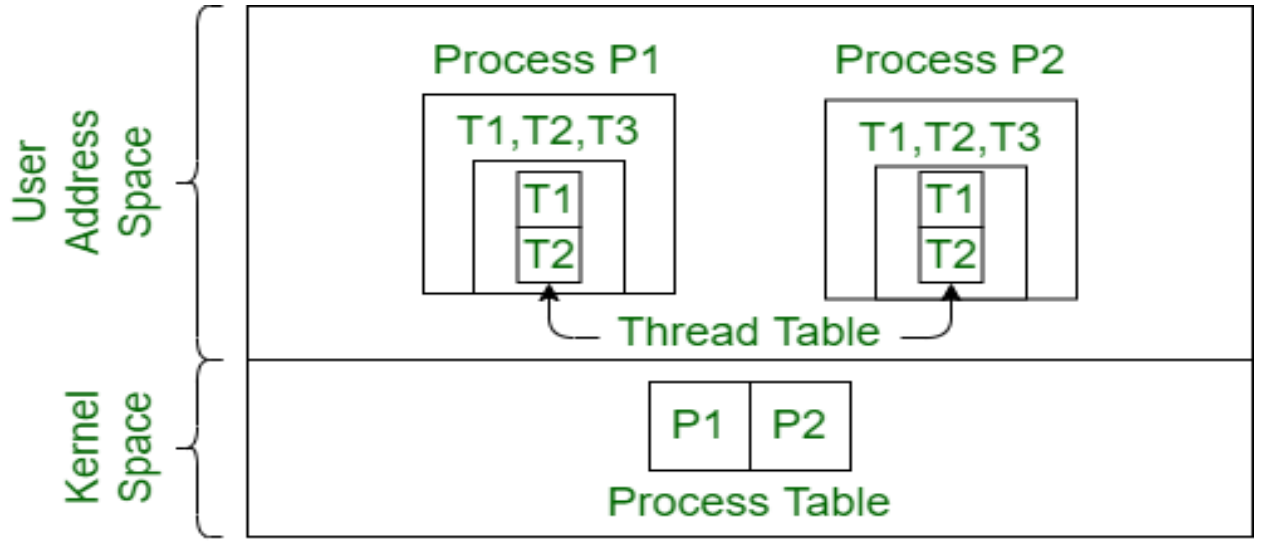
1. User Level Single Thread Model :

- Each process contains a single thread.
- Single process is itself a single thread.
- process table contains an entry for every process by maintaining its PCB.



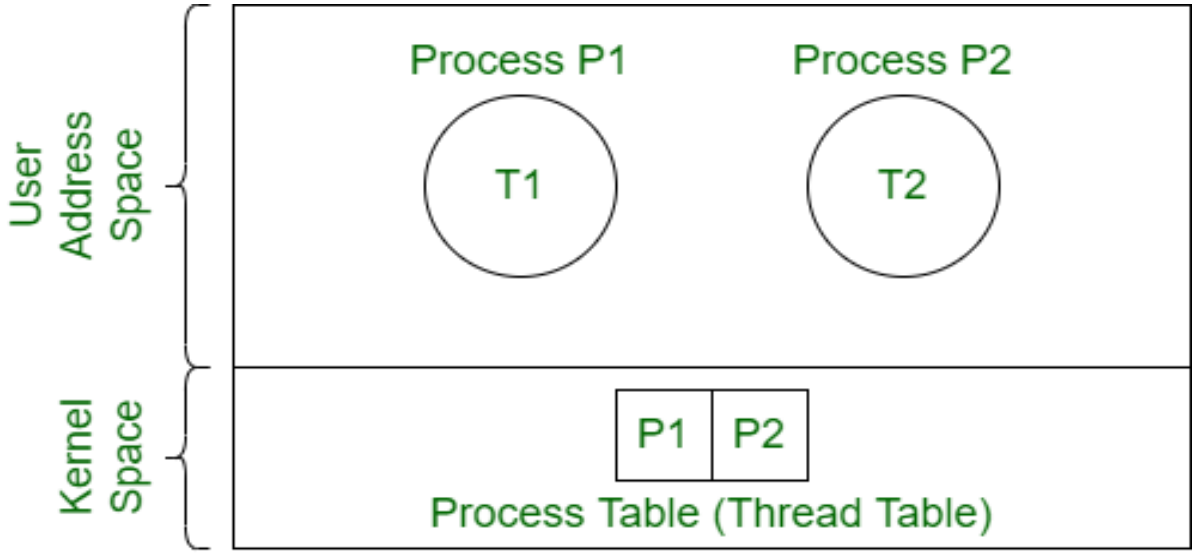
2. User Level Multi Thread Model :

- Each process contains multiple threads.
- All threads of the process are scheduled by a thread library at user level.
- Thread switching can be done faster than process switching.
- Thread switching is independent of operating system which can be done within a process.
- Blocking one thread makes blocking of entire process.
- Thread table maintains Thread Control Block of each thread of a process.
- Thread scheduling happens within a process and not known to Kernel.



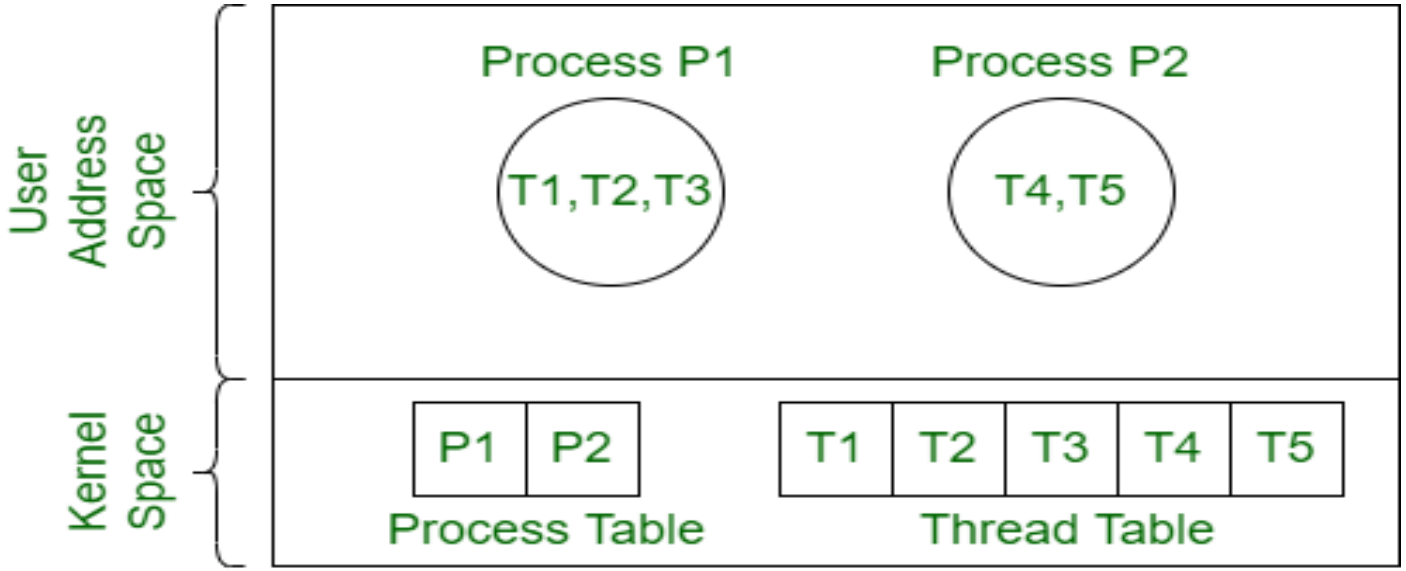
3. Kernel Level Single Thread Model :

- Each process contains a single thread.
- Thread used here is kernel level thread.
- Process table works as thread table.



4. Kernel Level Multi Thread Model :

- Thread scheduling is done at kernel level.
- Fine grain scheduling is done on a thread basis.
- If a thread blocks, another thread can be scheduled without blocking the whole process.
- Thread scheduling at Kernel process is slower compared to user level thread scheduling.
- Thread switching involves switch.



Unit 4

CPU scheduling

CPU scheduling is a process used by the operating system to decide which task or process gets to use the CPU at a particular time. This is important because a CPU can only handle one task at a time, but there are usually many tasks that need to be processed. The following are different purposes of a CPU scheduling time.

- Maximize the CPU utilization
- Minimize the response and waiting time of the process.Terminologies Used in CPU Scheduling

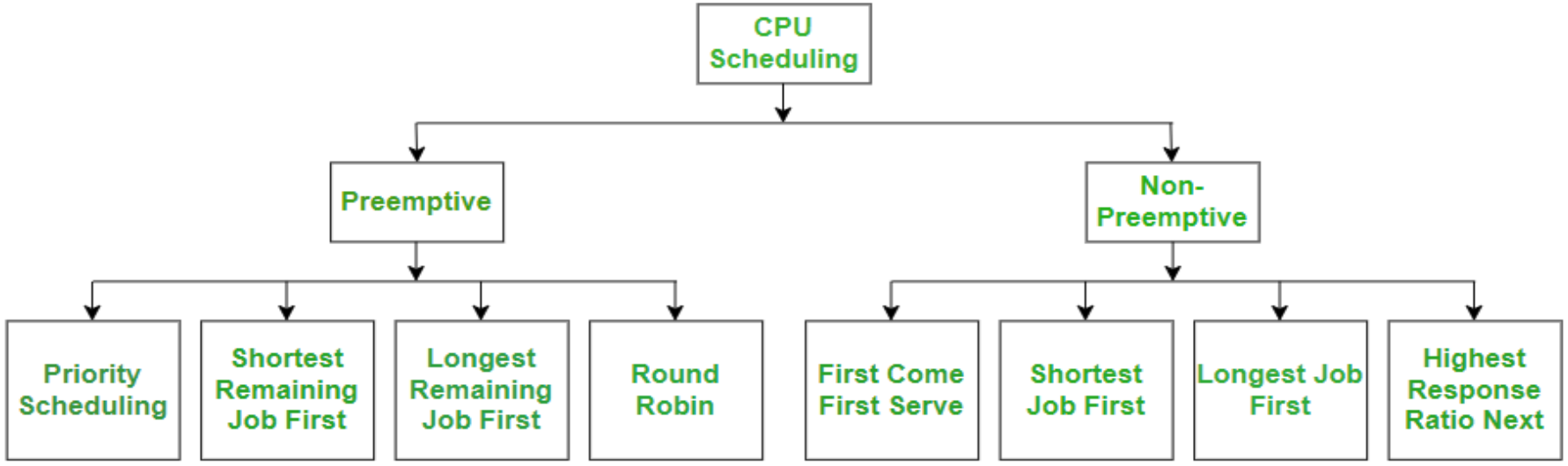
Terminologies Used in CPU Scheduling

Arrival Time: The time at which the process arrives in the ready queue.
Completion Time: The time at which the process completes its execution.
Burst Time: Time required by a process for CPU execution.
Turn Around Time: Time Difference between completion time and arrival time.
Turn Around Time = Completion Time – Arrival Time
Waiting Time(W.T): Time Difference between turn around time and burst time.
Waiting Time = Turn Around Time – Burst Time

Different Types of CPU Scheduling Algorithms

There are mainly two types of scheduling methods:

- **Preemptive Scheduling:** Preemptive scheduling is used when a process switches from running state to ready state or from the waiting state to the ready state.
- **Non-Preemptive Scheduling:** Non-Preemptive scheduling is used when a process terminates , or when a process switches from running state to waiting state.



Priority scheduling

Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems. Each process is assigned first arrival time (less arrival time process first) if two processes have same arrival time, then compare to priorities (highest process first). Also, if two processes have same priority then compare to process number (less process number first). This process is repeated while all process get executed.

Examples –

Input :

process no-> 1 2 3 4 5
arrival time-> 0 1 3 2 4
burst time-> 3 6 1 2 4
priority-> 3 4 9 7 8

Output :

Process_no	arrival_time	Burst_time	Complete_time	Turn_Around_Time	Waiting_Time
1	0	3	3	3	0
2	1	6	9	8	2
3	3	1	16	13	12
4	2	2	11	9	7
5	4	4	15	11	7

Average Waiting Time is : 5.6
Average Turn Around time is : 8.8

Shortest Remaining Time First (Preemptive SJF) Scheduling Algorithm

Consider the following table of arrival time and burst time for three processes P1, P2 and P3.

Process	Burst Time	Arrival Time
P1	6 ms	0 ms
P2	3 ms	1 ms
P3	7 ms	2 ms

Step-by-Step Execution:

Time 0-1 (P1): P1 runs for 1 ms (total time left: 5 ms) as it has shortest remaining time left.

Time 1-4 (P2): P2 runs for 3 ms (total time left: 0 ms) as it has shortest remaining time left among P1 and P2.

Time 4-9 (P1): P1 runs for 5 ms (total time left: 0 ms) as it has shortest remaining time left among P1 and P3.

Time 9-16 (P3): P3 runs for 7 ms (total time left: 0 ms) as it has shortest remaining time left.

Process	Arrival Time (AT)	Burst Time (BT)	Completion Time (CT)	Turn Around Time (TAT)	Waiting Time (WT)
P1	0	6	9	9-0 = 9	9-6 = 3
P2	1	3	4	4-1 = 3	3-3 = 0
P3	2	7	16	16-2 = 14	14-7 = 7

Average Turn around time = (9 + 14 + 3)/3 = 8.6 ms

Average waiting time = (3 + 0 + 7)/3 = 10/3 = 3.33 ms

LRJF Preemptive Scheduling Example

Scenario: Consider four processes, all arriving at time 0 (or shortly after) with varying burst times.

Process	Arrival Time (AT)	Burst Time (BT)
P1	0	3
P2	1	6
P3	2	4
P4	3	2

Gantt Chart:

P1	P2	P3	P1	P4
0	1	6	10	12	14

Process	**Arrival Time (AT)**	**Burst Time (BT)**	**Completion Time (CT)**	**Turn Around Time (TAT)**	**Waiting Time (WT)**
P1	0	3	12	12	9
P2	1	6	6	5	1
P3	2	4	10	8	4
P4	3	2	14	11	9

Average Turn around time = (12+5+8+11)/4 = 9 ms

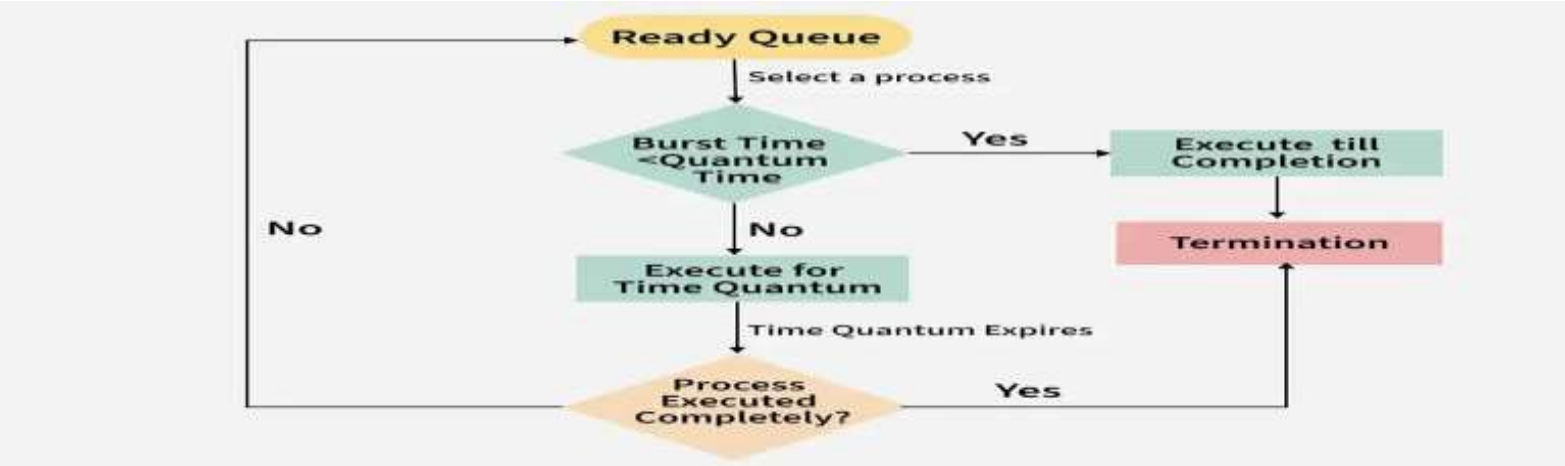
Average waiting time = (9+1+4+9)/4 = 5.75 ms

Round Robin Scheduling in Operating System

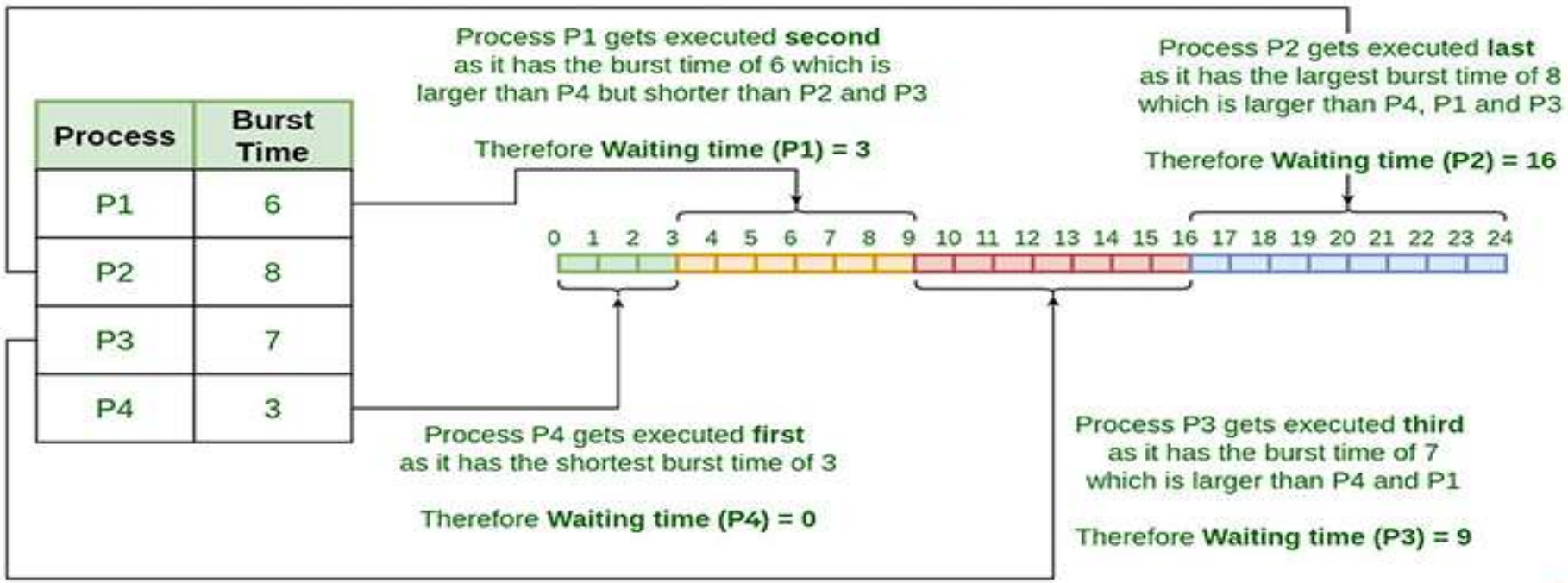
Round Robin Scheduling is a method used by operating systems to manage the execution time of multiple processes that are competing for CPU attention. It is called "round robin" because the system rotates through all the processes, allocating each of them a fixed time slice or "quantum", regardless of their priority.

The primary goal of this scheduling method is to ensure that all processes are given an equal opportunity to execute, promoting fairness among tasks.

- Process Arrival: Processes enter the system and are placed in a queue.
- Time Allocation: Each process is given a certain amount of CPU time, called a quantum.
- Execution: The process uses the CPU for the allocated time.
- Rotation: If the process completes within the time, it leaves the system. If not, it goes back to the end of the queue.
- Repeat: The CPU continues to cycle through the queue until all processes are completed.



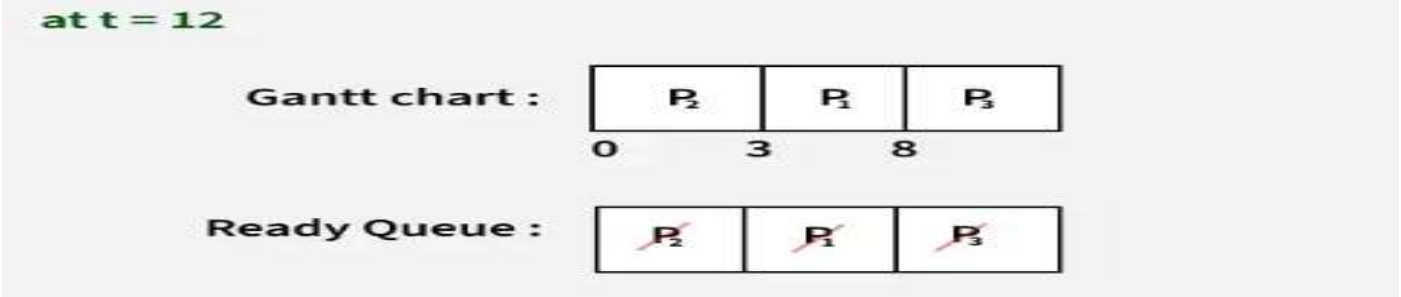
Shortest Job First (SJF) Scheduling Algorithm



FCFS - First Come First Serve CPU Scheduling

Consider the following table of arrival time and burst time for three processes P1, P2 and P3

Process	Burst Time (BT)	Arrival Time (AT)
P1	5 ms	2 ms
P2	3 ms	0 ms
P3	4 ms	4 ms



Process	Completion Time (CT)	Turnaround Time (TAT = CT - AT)	Waiting Time (WT = TAT - BT)
P2	3 ms	3 ms	0 ms
P1	8 ms	6 ms	1 ms
P3	12 ms	8 ms	4 ms

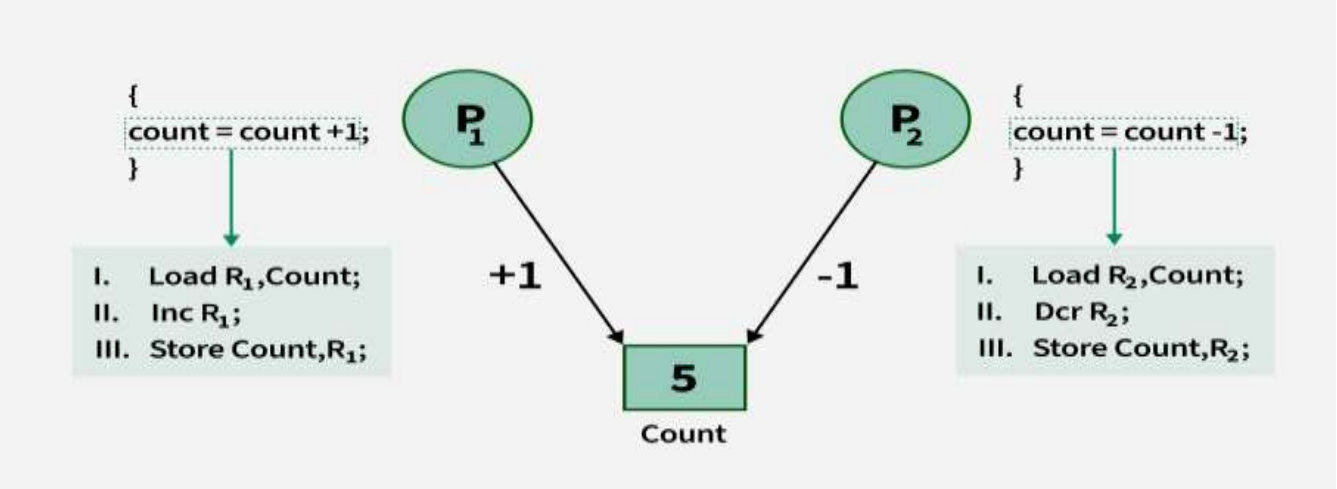
Now, lets calculate average waiting time and turn around time:

Average Turnaround time = 5.67

Average waiting time = 1.67

Process Synchronization

Process Synchronization is the coordination of execution of multiple processes in a multi-process system to ensure that they access shared resources in a controlled and predictable manner. It aims to resolve the problem of race conditions and other synchronization issues in a concurrent system.



Types of Process Synchronization

The two primary type of process Synchronization in an Operating System are:

Competitive: Two or more processes are said to be in Competitive Synchronization if and only if they compete for the accessibility of a shared resource.

Lack of Synchronization among Competing process may lead to either Inconsistency or Data loss.

Cooperative: Two or more processes are said to be in Cooperative Synchronization if and only if they get affected by each other i.e. execution of one process affects the other process.

Critical Regions in Operating System

In an operating system, a critical region refers to a section of code or a data structure that must be accessed exclusively by one method or thread at a time. Critical regions are utilized to prevent concurrent entry to shared sources, along with variables, information structures, or devices, that allow you to maintain information integrity and keep away from race conditions.

Critical Region Characteristics and Requirements

Following are the characteristics and requirements for critical regions in an operating system.

1. Mutual Exclusion

Only one procedure or thread can access the important region at a time. This ensures that concurrent entry does not bring about facts corruption or inconsistent states.

2. Atomicity

The execution of code within an essential region is dealt with as an indivisible unit of execution. It way that after a system or thread enters a vital place, it completes its execution without interruption.

3. Synchronization

Processes or threads waiting to go into a essential vicinity are synchronized to prevent simultaneous access. They commonly appoint synchronization primitives, inclusive of locks or semaphores, to govern access and put in force mutual exclusion.

4. Minimal Time Spent in Critical Regions

It is perfect to reduce the time spent inside crucial regions to reduce the capacity for contention and improve gadget overall performance. Lengthy execution within essential regions can increase the waiting time for different strategies or threads.

Classical inter process communication problems

Inter-Process Communication (IPC) is necessary for processes to communicate and share data. While basic communication between processes may sound simple, certain situations can cause issues that need specific solutions. These situations are known as Classical IPC Problems, which involve managing synchronization, avoiding deadlock, and ensuring that resources are accessed in a controlled manner.

Some of the well-known Inter Process Communication problems are:

- Producer Consumer Problem
- Readers-Writers Problem
- Dining Philosophers Problem

Producer Consumer Problem

The Producer-Consumer Problem involves two types of processes: the Producer, which creates data, and the Consumer, which processes that data. The challenge is ensuring that the Producer doesn't overfill the buffer, and the Consumer doesn't try to consume data from an empty buffer.

Reader-Writer Problem

The Reader-Writer Problem involves multiple processes that need to read from and write to shared data. Here, we have two types of processes: Readers, which only read the data, and Writers, which can modify the data.

Readers: These processes only read data from the shared resource and do not modify it.

Writers: These processes modify or write data to the shared resource.

Sleeping Barber Problem

In the Sleeping Barber Problem, there is a barber shop with one barber and several chairs for customers. The barber sleeps while there are no customers. When a customer arrives, they sit in an empty chair. If all chairs are taken, the customer leaves. If the barber is sleeping, the customer wakes him up; if the barber is busy, the customer waits.

The main challenge is to avoid deadlock (where no customer can get a haircut) and ensure fairness so that customers don’t starve waiting for service

What is Semaphores?

A semaphore is a synchronization tool used in concurrent programming to manage access to shared resources. It is a lock-based mechanism designed to achieve process synchronization, built on top of basic locking techniques.

Semaphores use a counter to control access, allowing synchronization for multiple instances of a resource. Processes can attempt to access one instance, and if it is not available, they can try other instances. Unlike basic locks, which allow only one process to access one instance of a resource. Semaphores can handle more complex synchronization scenarios, involving multiple processes or threads. It help prevent problems like race conditions by controlling when and how processes access shared data.

Monitors

- Definition: High-level synchronization construct ensuring only one process is active within the monitor at any time.
- Components: Shared data, procedures operating on data, and entry/exit protocols.
- Condition Variables: Used within monitors for processes to wait for specific conditions (e.g., wait(), signal()).

What is Deadlock?

Deadlock is a situation in computing where two or more processes are unable to proceed because each is waiting for the other to release resources. Key concepts include mutual exclusion, resource holding, circular wait, and no preemption. Consider a practical example when two trains are coming toward each other on the same track and there is only one track, none of the trains can move once they are in front of each other.

Deadlock is an infinite Process it means that once a process goes into deadlock it will never come out of the loop and the process will enter for an indefinite amount of time. There are only detection, resolution, and prevention techniques. But, there are no Deadlock-stopping techniques.

Deadlock Characterization (The Four Conditions)

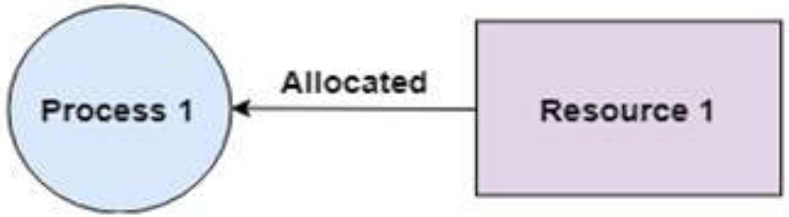
- Mutual Exclusion
- Hold and Wait
- No Pre Emption
- Circular Wait

A deadlock happens in operating system when two or more processes need some resource to complete their execution that is held by the other process.

A deadlock occurs if the four Coffman conditions hold true. But these conditions are not mutually exclusive. They are given as follows ?

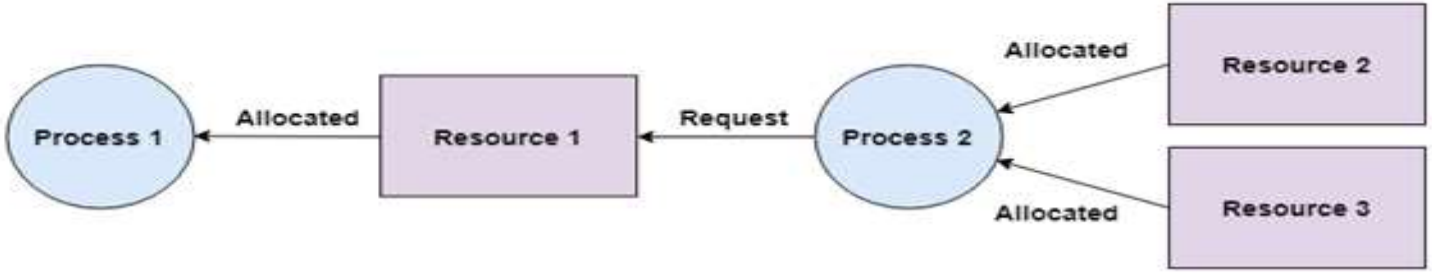
Mutual Exclusion

There should be a resource that can only be held by one process at a time. In the diagram below, there is a single instance of Resource 1 and it is held by Process 1 only.



Hold and Wait

A process can hold multiple resources and still request more resources from other processes which are holding them. In the diagram given below, Process 2 holds Resource 2 and Resource 3 and is requesting the Resource 1 which is held by Process 1.



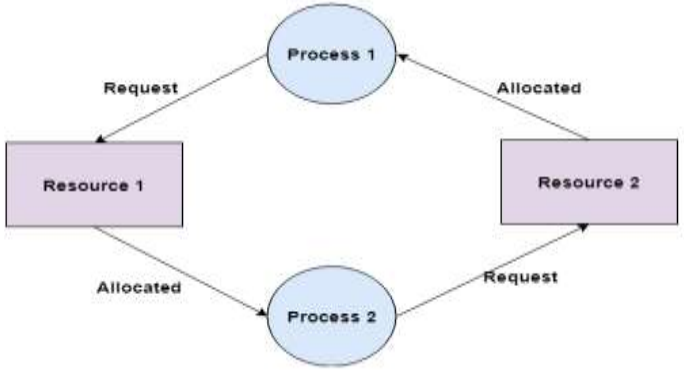
No Preemption

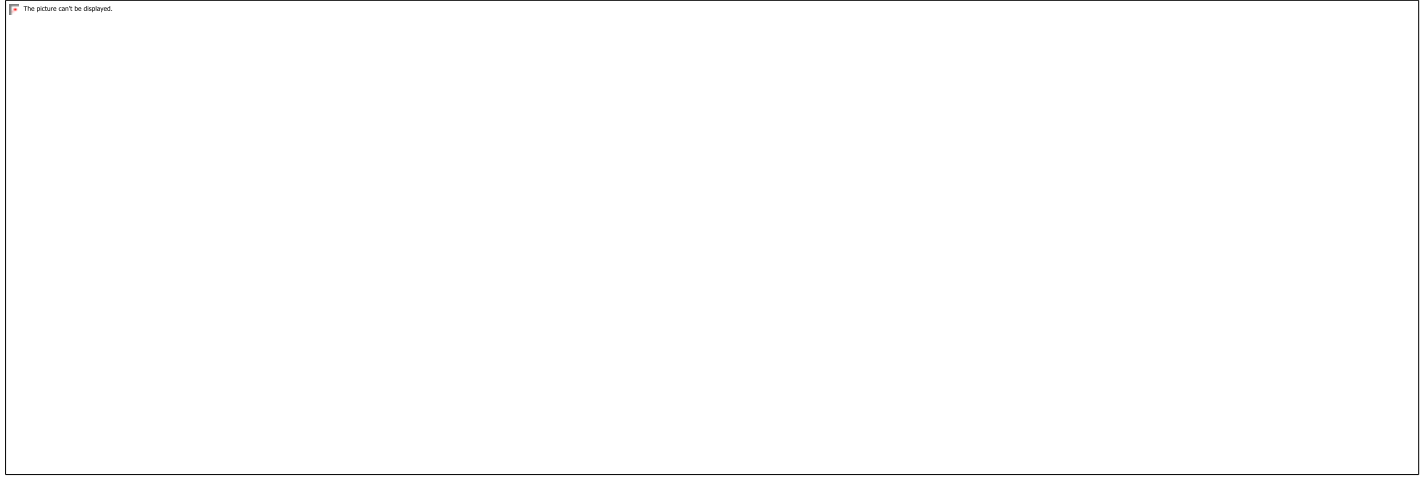
A resource cannot be preempted from a process by force. A process can only release a resource voluntarily. In the diagram below, Process 2 cannot preempt Resource 1 from Process 1. It will only be released when Process 1 relinquishes it voluntarily after its execution is complete.



Circular Wait

A process is waiting for the resource held by the second process, which is waiting for the resource held by the third process and so on, till the last process is waiting for a resource held by the first process. This forms a circular chain. For example: Process 1 is allocated Resource2 and it is requesting Resource 1. Similarly, Process 2 is allocated Resource 1 and it is requesting Resource 2. This forms a circular wait loop





1. Deadlock Prevention

Deadlock Prevention is a method of handling deadlocks where the operating system ensures that at least one of the necessary conditions for deadlock (mutual exclusion, hold and wait, no preemption, circular wait) never occurs. By breaking these conditions in advance, the system prevents deadlock from happening.

2. Deadlock Avoidance

Deadlock Avoidance is a method where the operating system makes decisions dynamically to ensure the system never enters an unsafe state, thereby avoiding the possibility of deadlock. This is usually done using algorithms like the Banker's Algorithm.

Deadlock avoidance mainly relies on algorithms that check resource allocation before making decisions. The two common types are:

1.Banker's Algorithm: Used when multiple instances of resources exist. It simulates allocation and ensures that the system remains in a safe state before granting resources.

2.Resource Allocation Graph (RAG) Algorithm: Used when each resource has only one instance. It checks for cycles in the graph to avoid unsafe states.

3. Deadlock Detection & Recovery

Deadlock Detection: Deadlock detection periodically checks for circular waits and resolves deadlocks by aborting and restarting processes, releasing their resources. It allows unrestricted resource access and immediate request fulfillment, enabling online handling. The main drawback is potential loss due to preemption.

Deadlock Recovery is the process of handling a deadlock after it has occurred. It involves:

1.Process Termination: Aborting one or more deadlocked processes to break the cycle.

2.Resource Preemption: Temporarily taking resources from some processes and reallocating them to others.

4. Deadlock Ignorance

In the Deadlock ignorance method the OS acts like the deadlock never occurs and completely ignores it even if the deadlock occurs. This method only applies if the deadlock occurs very rarely. The algorithm is very simple. It says, " if the deadlock occurs, simply reboot the system and act like the deadlock never occurred." That's why the algorithm is called the Ostrich Algorithm.

Unit 5

Logical and Physical Address

Logical Address

A logical address is generated by the CPU while a program runs. It represents the address from the process's perspective and does not exist physically, hence it is also called a virtual address.

- The logical address space is the set of all logical addresses a process can generate.
- Programs use logical addresses to reference memory, and the MMU translates them into physical addresses when accessing actual memory.

Physical Address

A physical address is the real location in main memory (RAM) where data or instructions are stored. The physical address space consists of all physical addresses corresponding to logical addresses.

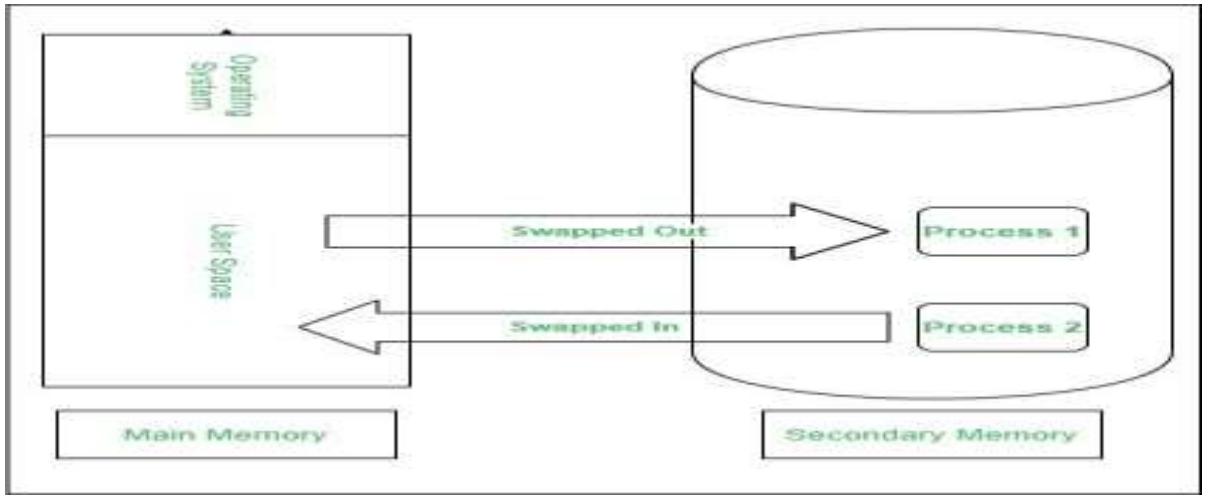
- The MMU performs address translation using a page table, mapping each logical page to a physical frame.
- This allows processes to access memory transparently, without knowing actual memory locations.

What is Swapping in the Operating System?

Swapping in an operating system is a process that moves data or programs between the computer's main memory (RAM) and a secondary storage (usually a hard disk or SSD). This helps manage the limited space in RAM and allows the system to run more programs than it could otherwise handle simultaneously.

It's important to remember that swapping is only used when data isn't available in RAM. Although the swapping process degrades system performance, it allows larger and multiple processes to run concurrently. Because of this, swapping is also known as memory compaction.

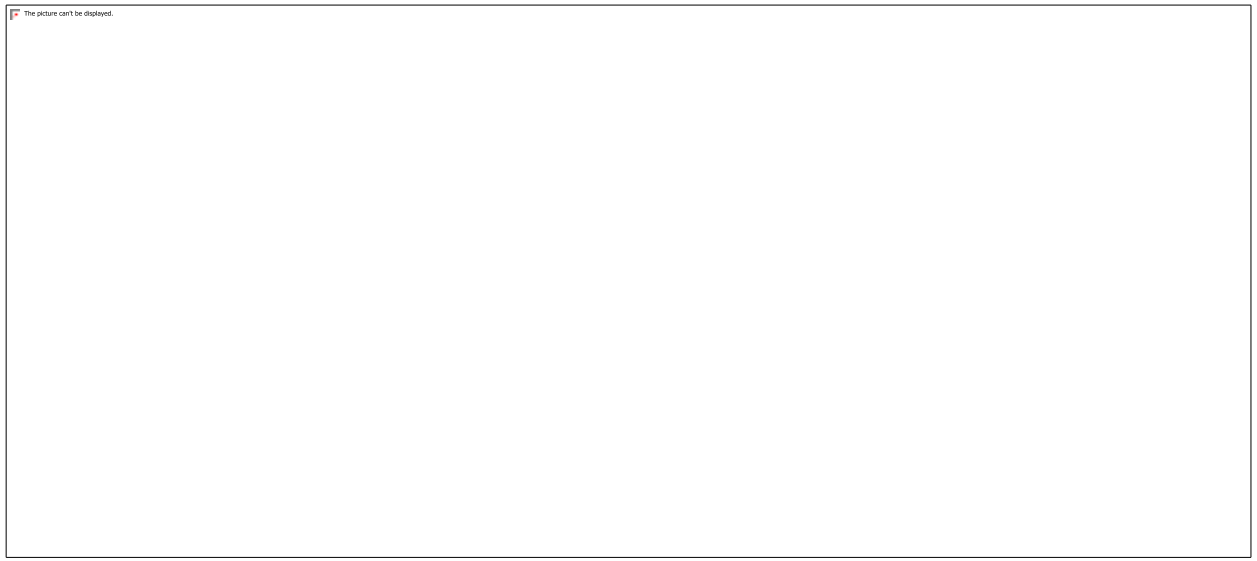
The CPU scheduler determines which processes are swapped in and which are swapped out. Consider a multiprogramming environment that employs a priority-based scheduling algorithm. When a high-priority process enters the input queue, a low-priority process is swapped out so the high-priority process can be loaded and executed. When this process terminates, the low-priority process is swapped back into memory to continue its execution. The below figure shows the swapping process in the operating system:



Paging

Paging is the process of moving parts of a program, called pages, from secondary storage into the main memory (RAM). The main idea behind paging is to break a program into smaller fixed-size blocks called pages.

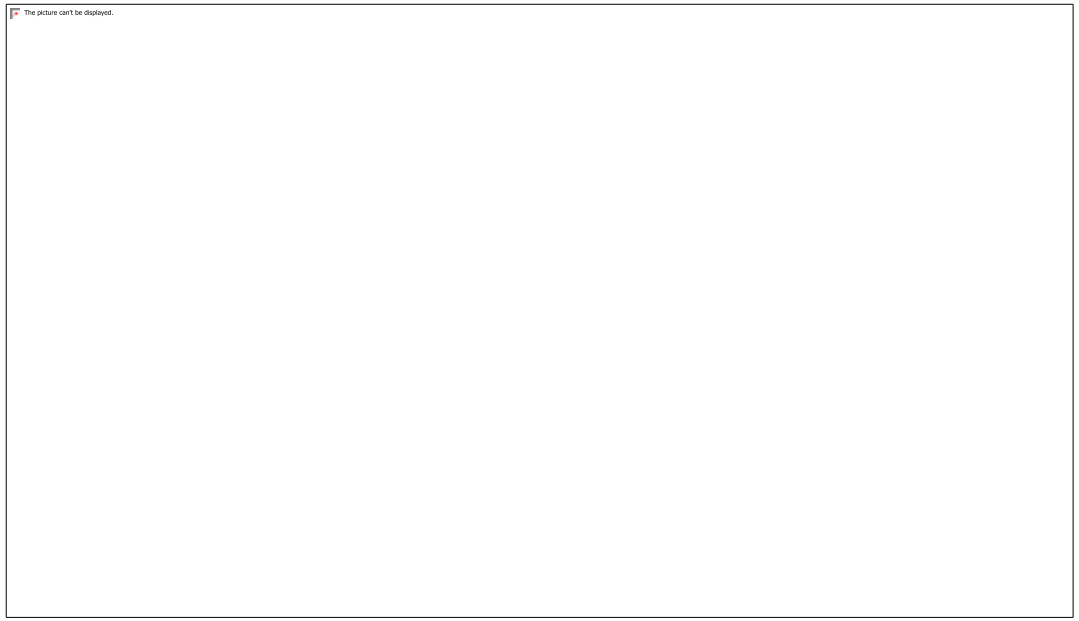
- Process does not have to be allocated in a contiguous memory space.
- The whole process does not have to be in main memory, some pages can be present and some pages can be loaded when needed. It allows more processes and even processes larger than main memory to run.
- Memory allocation is simplified as memory is always allocated in fixed sized pages.



Segmentation in Operating System

Segmentation is a memory management technique where a process is divided into variable-sized chunks called segments. Unlike paging, segmentation matches the user's logical view of a program (functions, arrays, modules) to physical memory.

It reflects the user's view of memory rather than the computer's physical organization, making it easier to manage and protect processes.



Translation Lookaside Buffer (TLB) in Paging

Operating systems use paging to map virtual addresses to physical addresses efficiently. Each process maintains its own Page Table, which contains Page Table Entries (PTEs) responsible for translating virtual page numbers into physical frame numbers. This mechanism enables flexible memory allocation and prevents processes from interfering with each other's memory space.

Key Points

- Paging divides memory into fixed-size pages and frames.
- Each process has a separate Page Table for address translation.
- Page Table Entries (PTEs) store the frame number and status bits.
- Paging improves memory protection, isolation, and efficient utilization.



What is Demand Paging in Operating System?

Demand paging is a technique used in virtual memory systems where pages enter main memory only when requested or needed by the CPU. OS loads only the necessary pages of a program into memory at runtime, instead of loading the entire program into memory at the start. Here,

- A page fault occurred when the program needed to access a page that is not currently in memory.
- The operating system then loads the required pages from the disk into memory and updates the page tables accordingly.
- This process is transparent to the running program and it continues to run as if the page had always been in memory.

Page Replacement Algorithms in Operating Systems

In an operating system that uses paging, a page replacement algorithm is needed when a page fault occurs and no free page frame is available. In this case, one of the existing pages in memory must be replaced with the new page.

The virtual memory manager performs this by:

1.Selecting a victim page using a page replacement algorithm.

2.Marking its page table entry as “not present.”

3.If the page was modified (dirty), writing it back to disk before replacement.

The efficiency of a page replacement algorithm directly affects the page fault rate, which in turn impacts system performance.

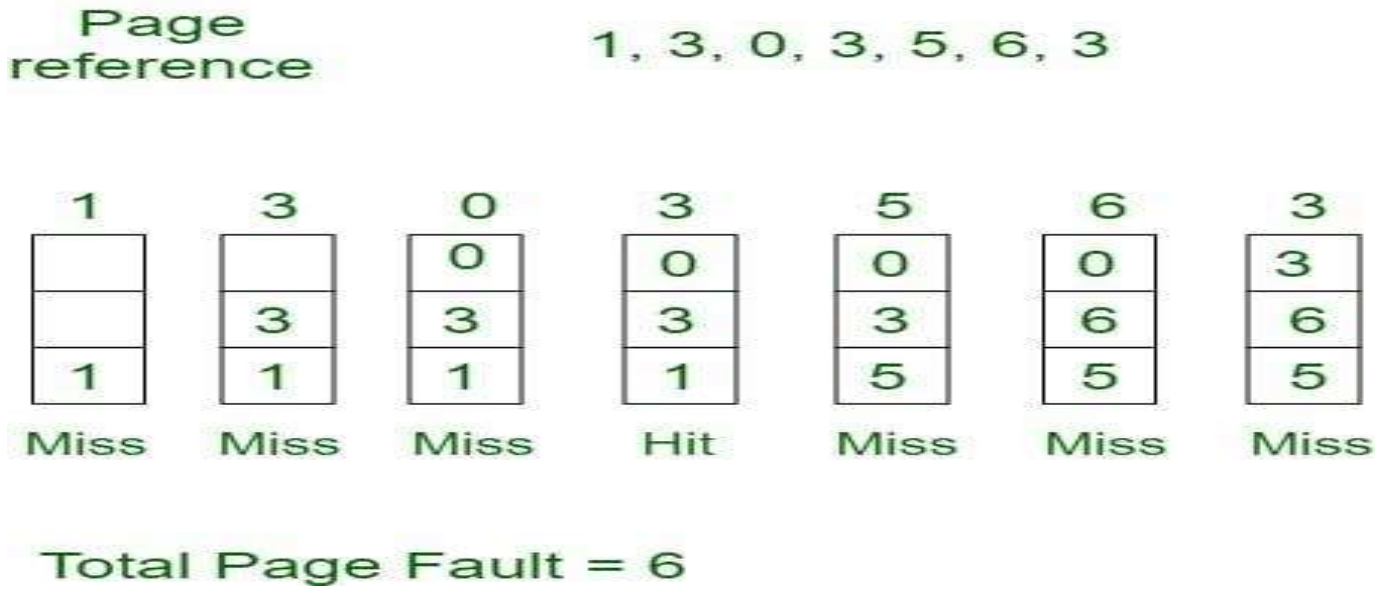
Common Page Replacement Techniques

- First In First Out (FIFO)
- Optimal Page replacement
- Least Recently Used (LRU)
- Most Recently Used (MRU)

1. First In First Out (FIFO)

This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

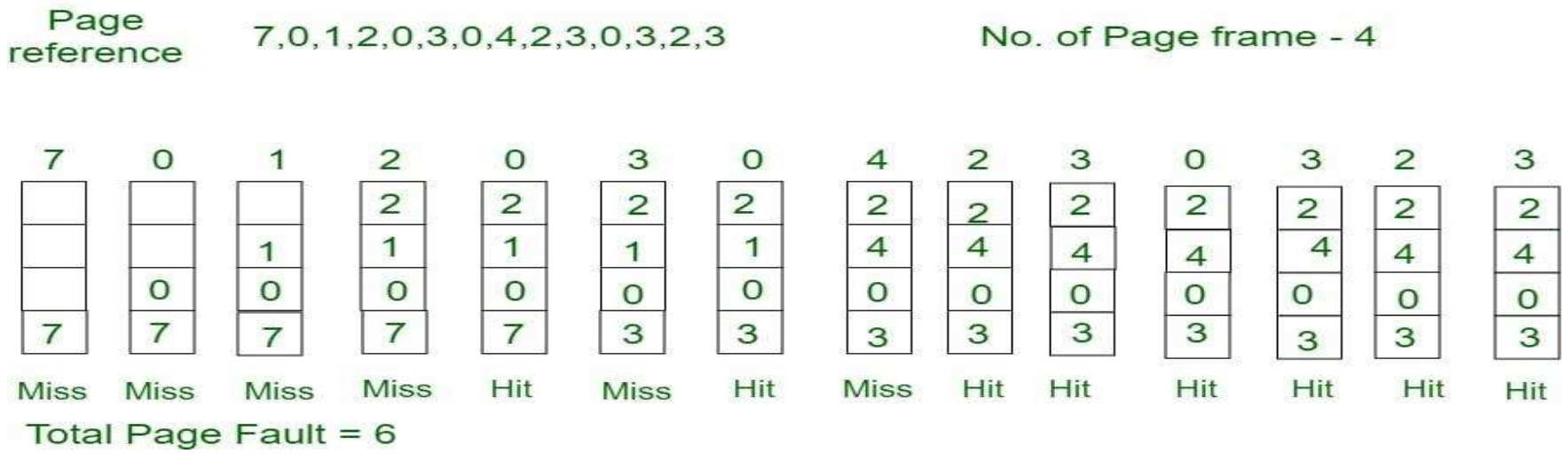
Example 1: Consider page reference string 1, 3, 0, 3, 5, 6, 3 with 3-page frames. Find the number of page faults using FIFO Page Replacement Algorithm.



2. Optimal Page Replacement

In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

Example: Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4-page frame. Find number of page fault using Optimal Page Replacement Algorithm.



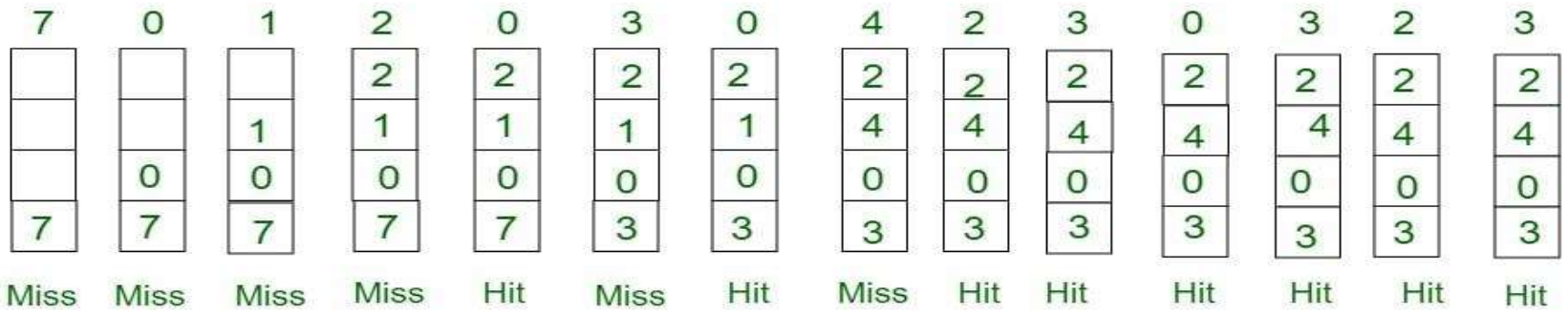
Now for the further page reference string ---> 0 Page fault because they are already available in the memory. Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

3. Least Recently Used

In this algorithm, page will be replaced which is least recently used.

Example Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4-page frames. Find number of page faults using LRU Page Replacement Algorithm.

Page reference 7,0,1,2,0,3,0,4,2,3,0,3,2,3 No. of Page frame - 4



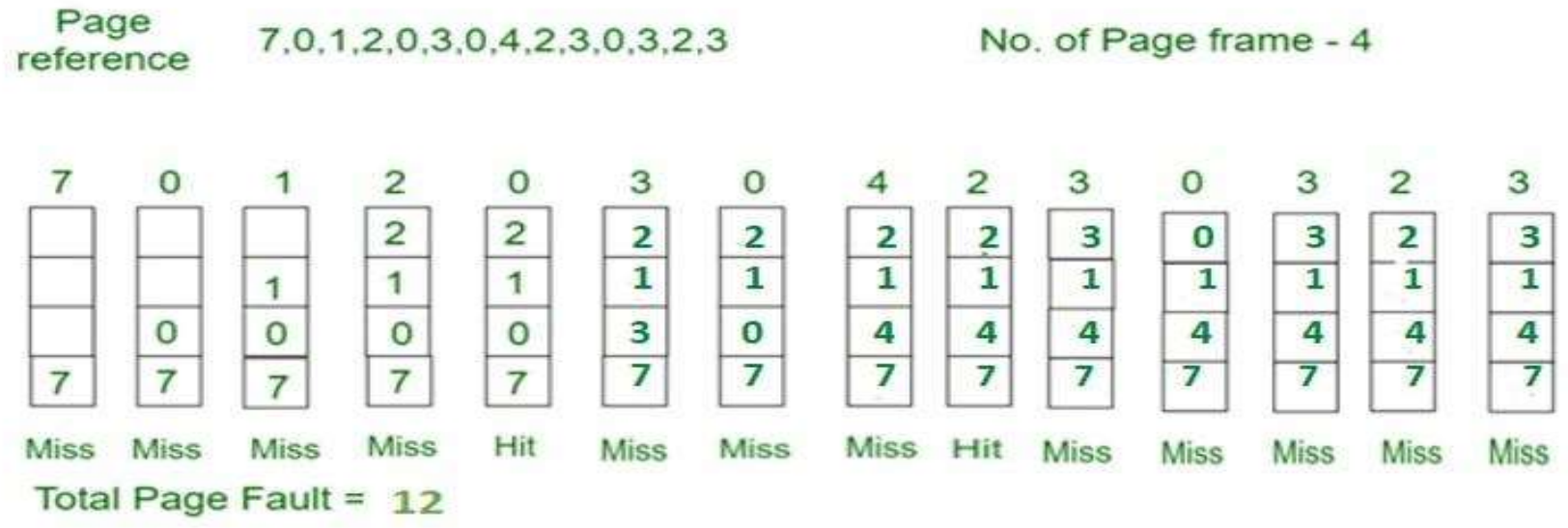
Total Page Fault = 6

Here LRU has same number of page fault as optimal but it may differ according to question.

4. Most Recently Used (MRU)

In this algorithm, page will be replaced which has been used recently. Belady's anomaly can occur in this algorithm.

Example 4: Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4-page frames. Find number of page faults using MRU Page Replacement Algorithm.

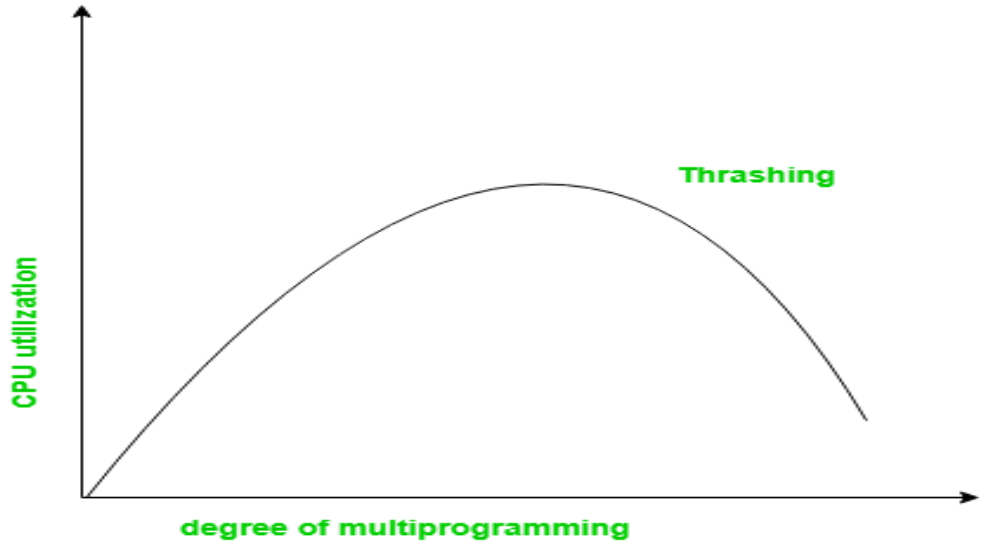


Thrashing

Thrashing occurs when the operating system spends more time swapping pages between main memory and disk than executing processes. This leads to excessive page faults and a significant drop in CPU utilization.

The cycle works like this:

- High degree of multiprogramming: Too many processes are loaded into memory.
- Lack of frames: Each process gets fewer frames than needed.
- Page replacement policy: Frequent replacements increase page faults.



This repeated cycle of low CPU utilization → more processes → more page faults is called Thrashing.

Working Set Model

The working set model aims to keep a process's working set entirely in main memory. It does this by:

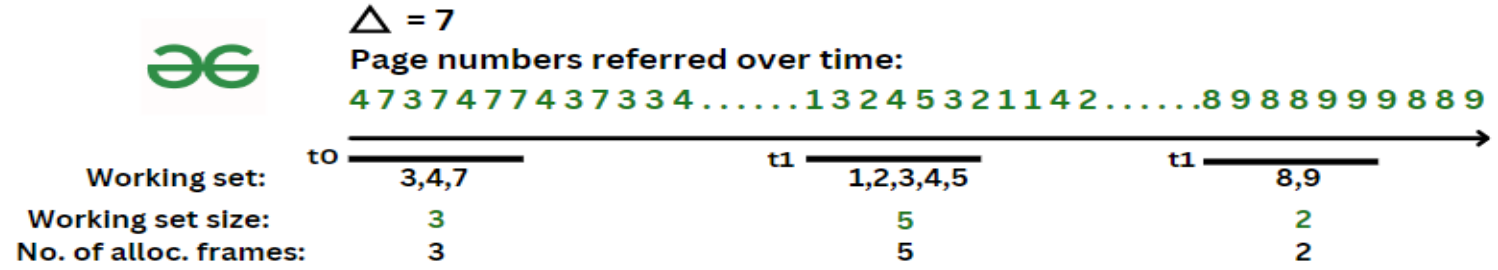
1.Defining the Working Set: The model uses a working set window (Δ), which is a time-based parameter (e.g., the last 1000 memory accesses). The working set for a process is the set of all unique pages referenced within this window. The size of this set is called the working set size (WSS).

2.Dynamic Frame Allocation: The operating system allocates a number of frames to a process equal to its working set size. This ensures that all the pages a process is actively using are in main memory, minimizing page faults and preventing thrashing.

3.Adapting to Change: As a process's locality of reference changes over time, its working set size also changes. The operating system dynamically adjusts the number of frames allocated to the process, giving more frames when its working set grows and reclaiming frames when it shrinks.

4.Handling Memory Shortage: If the total frames required by all active processes (the sum of all their working set sizes) exceed the total available frames in main memory, the operating system may suspend one or more processes to free up frames for others, thus preventing the entire system from thrashing.

Example:



Example of a working set model sampling at 3 time instances with delta = 7

Window Size (Δ = 7): At each point in time, the working set is calculated from the last 7 page references.

1. At t_0:
   - Recent references: 4,7,3,7,4,7,7
   - Unique pages = {3,4,7}, WSS = 3
   - Frames allocated = 3
2. At t_1:
   - Recent references: 1,3,2,4,5,3,2
   - Unique pages = {1,2,3,4,5}, WSS = 5
   - Frames allocated = 5
3. At t_2:
   - Recent references: 8,9,8,8,9,9,9
   - Unique pages = {8,9}, WSS = 2
   - Frames allocated = 2

Disk Scheduling Algorithms

Disk scheduling algorithms manage how data is read from and written to a computer's hard disk. These algorithms help determine the order in which disk read and write requests are processed.

- Disk scheduling is also known as I/O Scheduling.
- The main goals of disk scheduling are to optimize the performance of disk operations, reduce the time it takes to access data and improve overall system efficiency.
- Common disk scheduling methods include First-Come, First-Served (FCFS), Shortest Seek Time First (SSTF), SCAN, C-SCAN, LOOK, and C-LOOK.

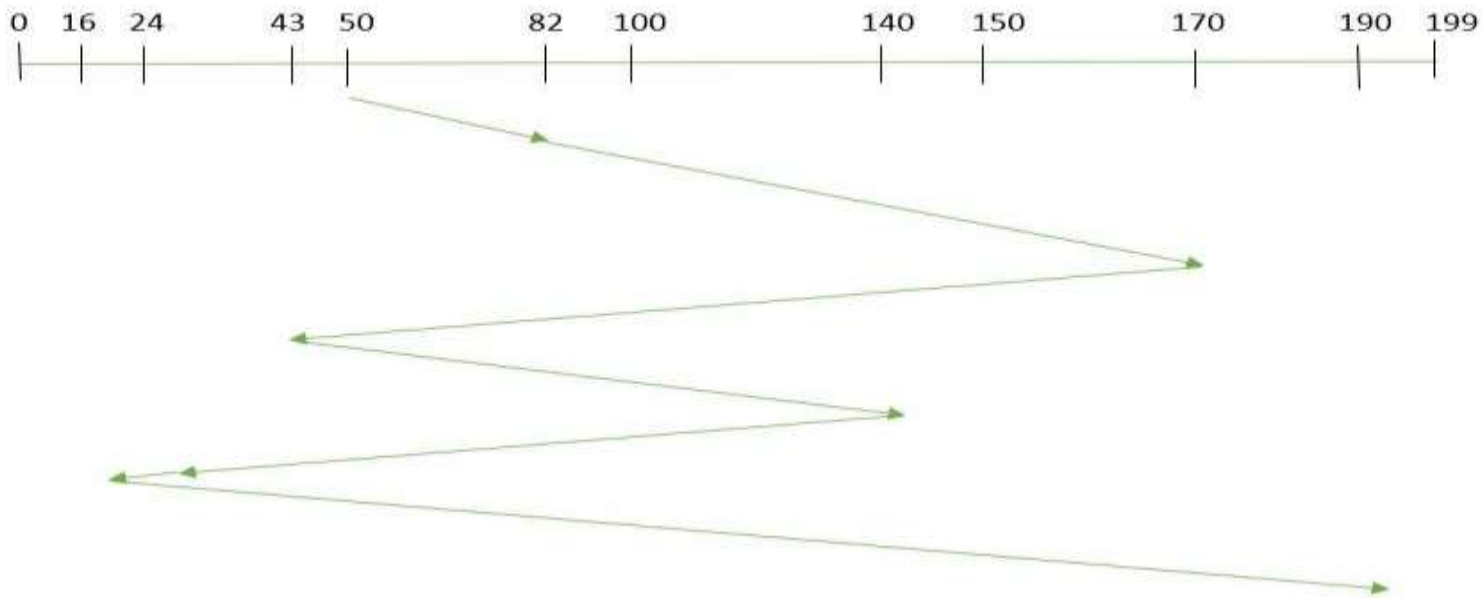
Disk Scheduling Algorithms

There are several Disk Several Algorithms. We will discuss in detail each one of them.

- FCFS (First Come First Serve)
- SSTF (Shortest Seek Time First)
- SCAN
- C-SCAN
- LOOK
- C-LOOK
- RSS (Random Scheduling)
- LIFO (Last-In First-Out)
- N-STEP SCAN
- F-SCAN

1. FCFS (First Come First Serve)

FCFS is the simplest of all Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue. Let us understand this with the help of an example.

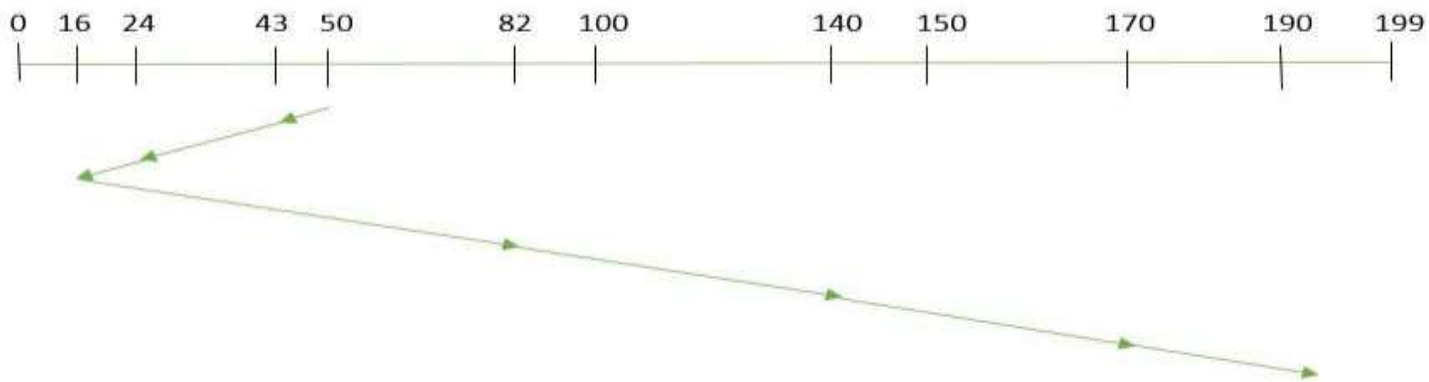


Suppose the order of request is- (82,170,43,140,24,16,190) and current position of Read/Write head is: 50 So, total overhead movement (total distance covered by the disk arm) = (82-50)+(170-82)+(170-43)+(140-43)+(140-24)+(24-16)+(190-16) =642

2. SSTF (Shortest Seek Time First)

In SSTF (Shortest Seek Time First), requests having the shortest seek time are executed first. So, the seek time of every request is calculated in advance in the queue and then they are scheduled according to their calculated seek time. As a result, the request near the disk arm will get executed first. SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of the system. Let us understand this with the help of an example.

Example:

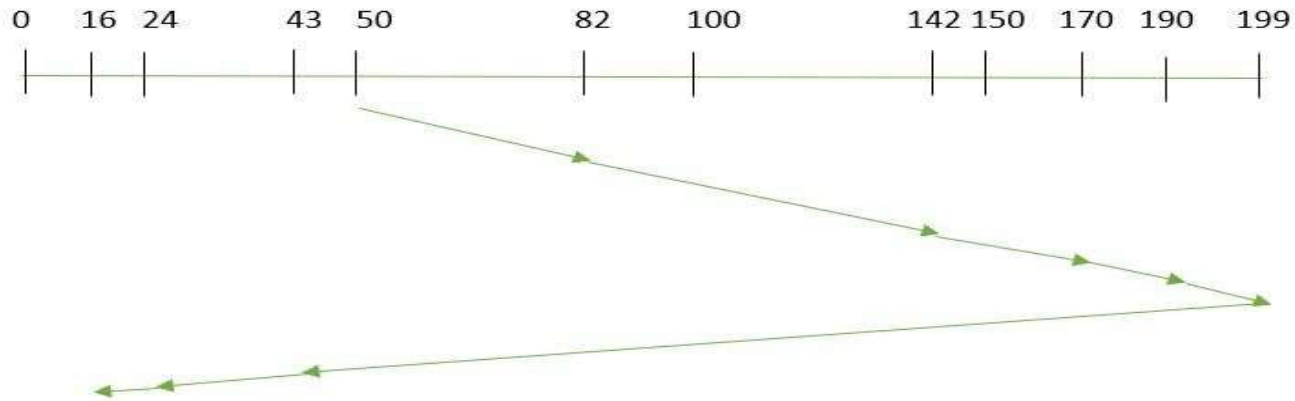


Suppose the order of request is- (82,170,43,140,24,16,190) and current position of Read/Write head is: 50 total overhead movement (total distance covered by the disk arm) =(50-43)+(43-24)+(24-16)+(82-16)+(140-82)+(170-140)+(190-170) =208

3. SCAN

In the SCAN algorithm the disk arm moves in a particular direction and services the requests coming in its path and after reaching the end of the disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works as an elevator and is hence also known as an elevator algorithm. As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

Example:



Suppose the requests to be addressed are-82,170,43,140,24,16,190 and the Read/Write arm is at 50, and it is also given that the disk arm should move "towards the larger value".

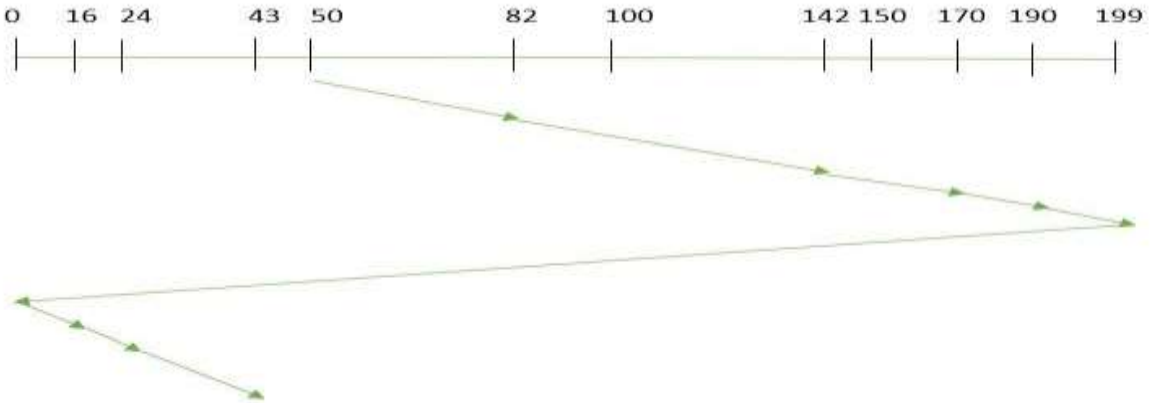
Therefore, the total overhead movement (total distance covered by the disk arm) is calculated as = (199-50) + (199-16) = 332

4. C-SCAN

In the SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction. So, it may be possible that too many requests are waiting at the other end or there may be zero or few requests pending at the scanned area.

These situations are avoided in the CSCAN algorithm in which the disk arm instead of reversing its direction goes to the other end of the disk and starts servicing the requests from there. So, the disk arm moves in a circular fashion and this algorithm is also similar to the SCAN algorithm hence it is known as C-SCAN (Circular SCAN).

Example:



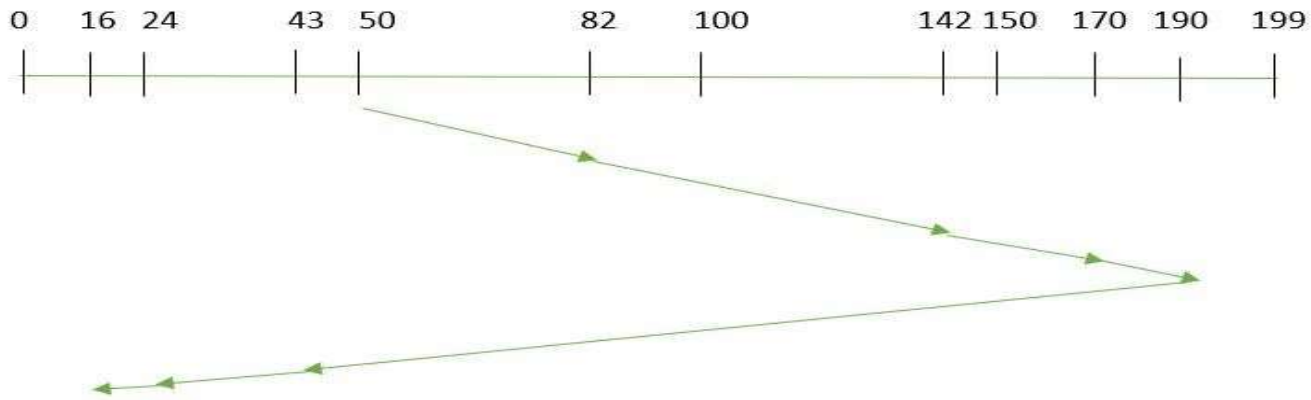
Suppose the requests to be addressed are- 82,170,43,140,24,16,190 and the Read/Write arm is at 50, and it is also given that the disk arm should move "towards the larger value".

So, the total overhead movement (total distance covered by the disk arm) is calculated as: =(199-50) + (199-0) + (43-0) = 391

5. LOOK

LOOK Algorithm is similar to the SCAN disk scheduling algorithm except for the difference that the disk arm in spite of going to the end of the disk goes only to the last request to be serviced in front of the head and then reverses its direction from there only. Thus it prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.

Example:

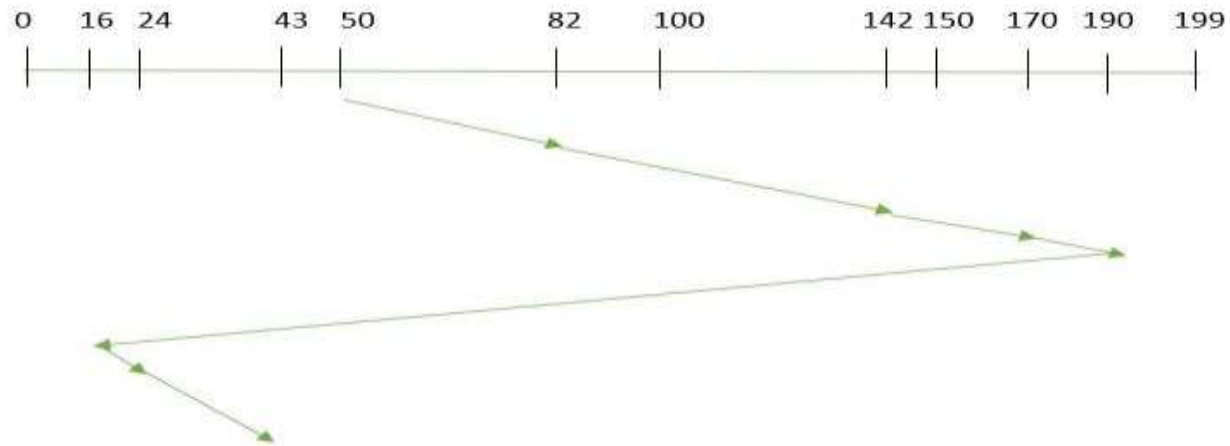


Suppose the requests to be addressed are- 82,170,43,140,24,16,190 and the Read/Write arm is at 50, and it is also given that the disk arm should move "towards the larger value".

So, the total overhead movement (total distance covered by the disk arm) is calculated as: = (190-50) + (190-16) = 314

6. C-LOOK

As LOOK is similar to the SCAN algorithm, in a similar way, C-LOOK is similar to the CSCAN disk scheduling algorithm. In CLOOK, the disk arm in spite of going to the end goes only to the last request to be serviced in front of the head and then from there goes to the other end's last request. Thus, it also prevents the extra delay which occurred due to unnecessary traversal to the end of the disk.



Example: Suppose the requests to be addressed are-82,170,43,140,24,16,190 and the Read/Write arm is at 50, and it is also given that the disk arm should move "towards the larger value"

C-LOOK So, the total overhead movement (total distance covered by the disk arm) is calculated as= (190-50) + (190-16) + (43-16) = 341

7. RSS (Random Scheduling)

It stands for Random Scheduling and just like its name it is natural. It is used in situations where scheduling involves random attributes such as random processing time, random due dates, random weights, and stochastic machine breakdowns this algorithm sits perfectly. Which is why it is usually used for analysis and simulation.

8. LIFO (Last-In First-Out)

In LIFO (Last In, First Out) algorithm, the newest jobs are serviced before the existing ones i.e. in order of requests that get serviced the job that is newest or last entered is serviced first, and then the rest in the same order.

Advantages of LIFO

Here are some of the advantages of the Last In First Out Algorithm.

- Maximizes locality and resource utilization
- Can seem a little unfair to other requests and if new requests keep coming in, it cause starvation to the old and existing ones.

9. N-STEP SCAN

It is also known as the N-STEP LOOK algorithm. In this, a buffer is created for N requests. All requests belonging to a buffer will be serviced in one go. Also once the buffer is full no new requests are kept in this buffer and are sent to another one. Now, when these N requests are serviced, the time comes for another top N request and this way all get requests to get a guaranteed service.

10. F-SCAN

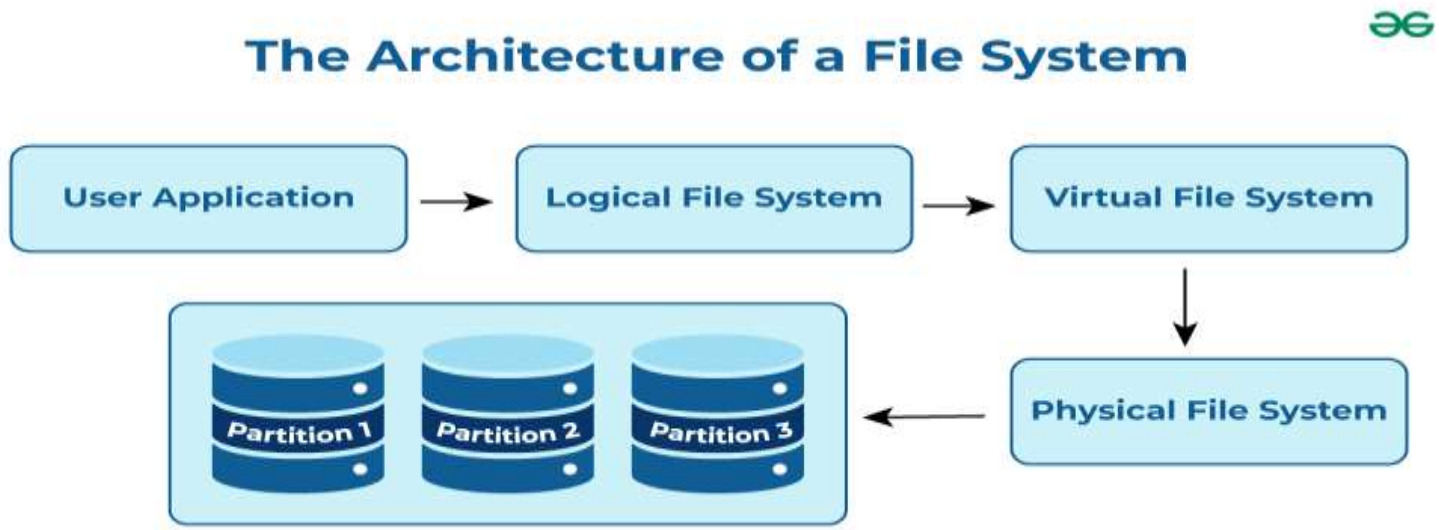
This algorithm uses two sub-queues. During the scan, all requests in the first queue are serviced and the new incoming requests are added to the second queue. All new requests are kept on halt until the existing requests in the first queue are serviced.

Advantages of F-SCAN

- Prevents Arm Stickiness: The head doesn't get stuck near one area because requests are split into two queues.
- Fairness: All requests in the first queue are guaranteed service before moving to the second, avoiding indefinite delays.

File Systems in Operating System

File systems are a crucial part of any operating system, providing a structured way to store, organize and manage data on storage devices such as hard drives, SSDs and USB drives.



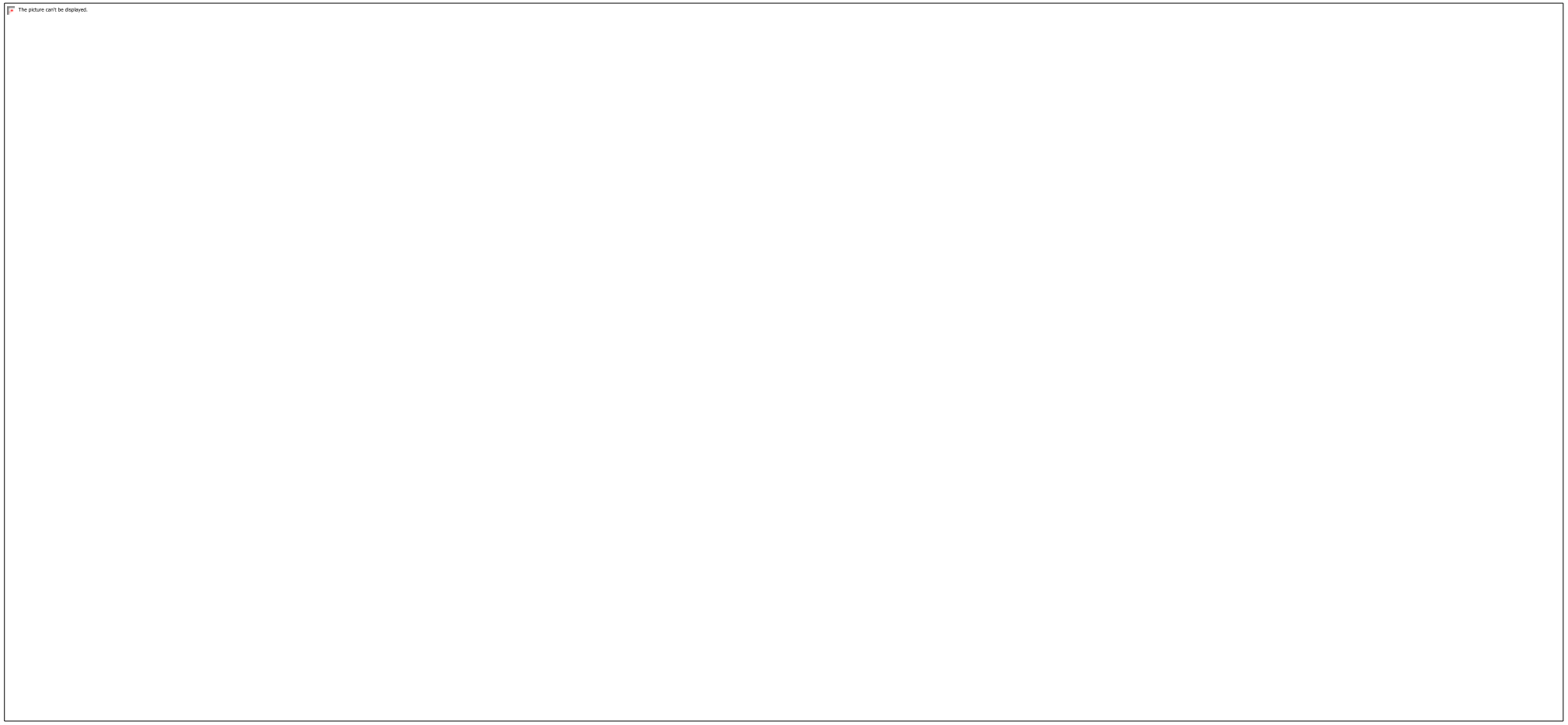
File System

- **User Application**: The programs or software that request file operations like read, write, or delete.
- **Logical File System**: Manages metadata, file names, directories, and access permissions.
- **Virtual File System (VFS)**: Acts as a bridge, allowing different file systems to work under a single interface.
- **Physical File System:** Handles the actual storage of data blocks on the disk.
- **Partition 1, Partition 2, Partition 3:** Divisions of the storage device where files are stored physically.

File Directories

The collection of files is a file directory and contains information about the files, including attributes, location, etc. Much of this information, is managed by the operating system.

File Types and Their Content



File Allocation Methods

The allocation methods define how the files are stored in the disk blocks. There are three main disk space or file allocation methods.

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

The main idea behind these methods is to provide Efficient disk space utilization & Fast access to the file blocks.

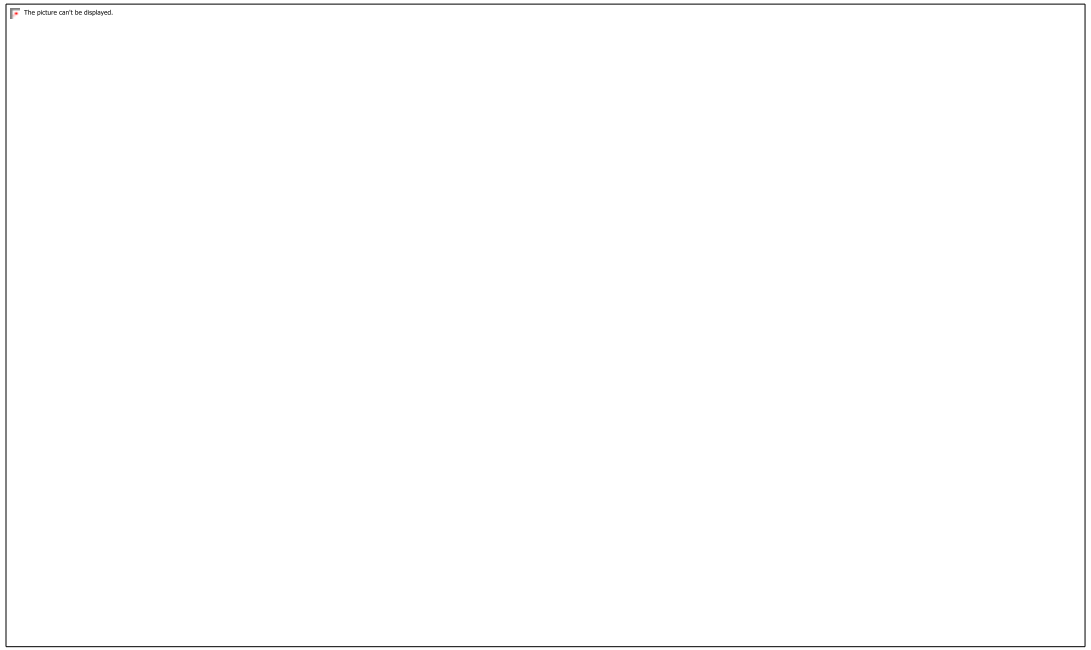
Contiguous Allocation

In this scheme, each file occupies a contiguous set of blocks on the disk. This means that given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file. The directory entry for a file with contiguous allocation contains:

- Address of starting block
- Length of the allocated portion.

Example: If a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be:

b, b+1, b+2,.....b+n-1.



Advantages

- Both the Sequential and Direct Accesses are supported by this. For direct access, the address of the kth block of the file which starts at block b can easily be obtained as (b+k)(b+k).
- This is extremely fast since the number of seeks are minimal because of contiguous allocation of file blocks.

Disadvantages

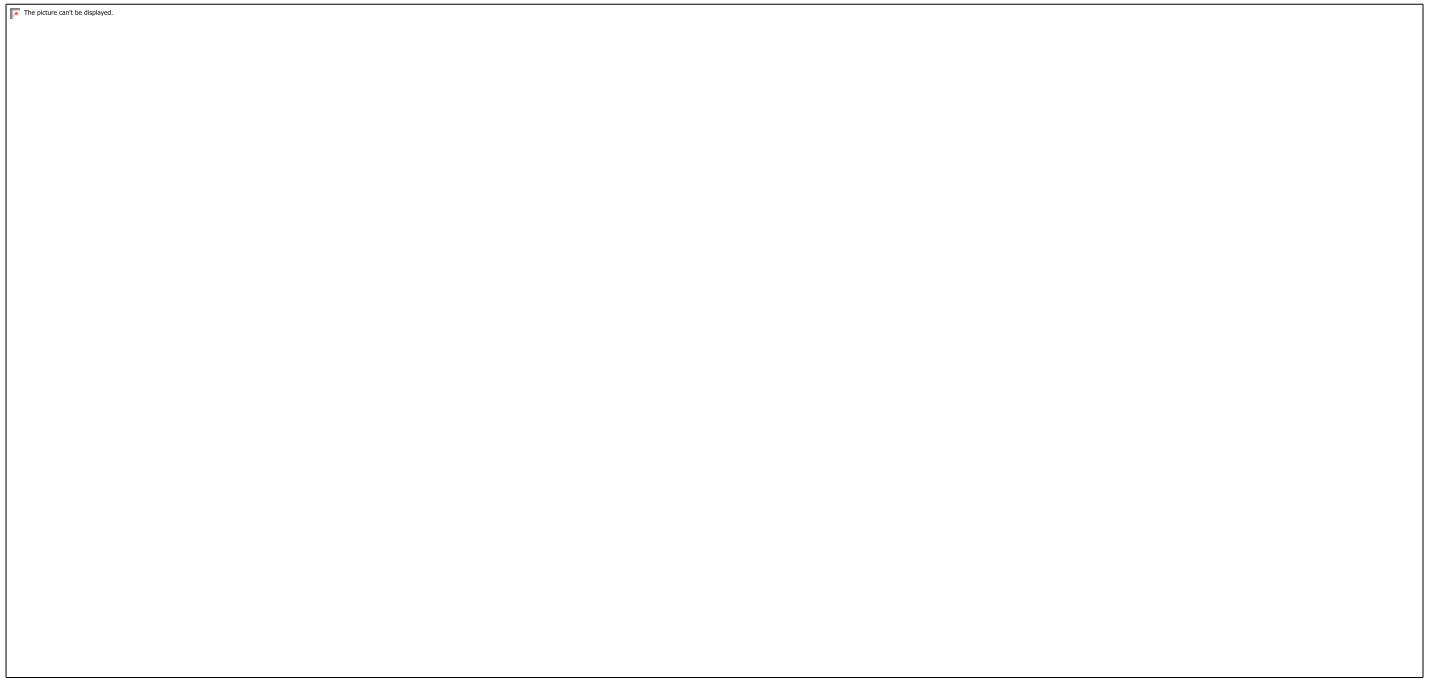
- This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.
- Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

Linked Allocation

In this scheme, each file is a linked list of disk blocks which need not be contiguous. Here,

- The disk blocks can be scattered anywhere on the disk. The directory entry contains a pointer to the starting and the ending file block.
- Each block contains a pointer to the next block occupied by the file.

Example: The file 'jeep' in following image shows how the blocks are randomly distributed. The last block (25) contains -1 indicating a null pointer and does not point to any other block.



Advantages

- This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for a contiguous chunk of memory.
- This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.

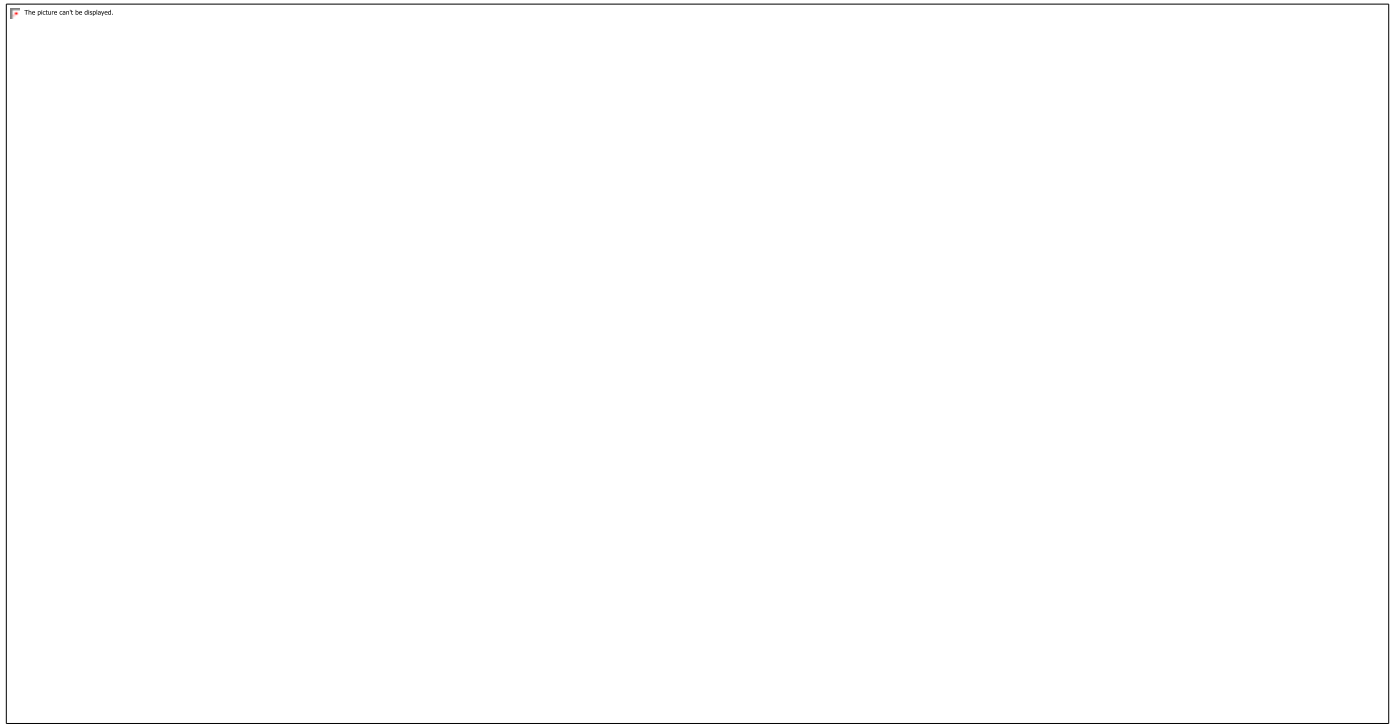
Disadvantages

- Because the file blocks are distributed randomly on the disk, a large number of seeks are needed to access every block individually. This makes linked allocation slower.
- It does not support random or direct access. We can not directly access the blocks of a file. A block k of a file can be accessed by traversing k blocks sequentially (sequential access) from the starting block of the file via block pointers.
- Pointers required in the linked allocation incur some extra overhead.

Indexed Allocation

In this scheme, a special block known as the Index block contains the pointers to all the blocks occupied by a file. Here,

- Each file has its own index block.
- The ith entry in the index block contains the disk address of the ith file block.
- The directory entry contains the address of the index block as shown in the image.



Advantages

- This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.
- It overcomes the problem of external fragmentation.

Disadvantages

- The pointer overhead for indexed allocation is greater than linked allocation.
- For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers which is inefficient in terms of memory utilization. However, in linked allocation we lose the space of only 1 pointer per block.

I/O Hardware in Operating System

I/O Hardware is a set of specialized hardware devices that help the operating system access disk drives, printers, and other peripherals. These devices are located inside the motherboard and connected to the processor using a bus. They often have specialized controllers that allow them to quickly respond to requests from software running on top of them or even respond directly to commands from an application program. This post will discuss in detail I/O Hardware basics such as daisy chain expansion bus controller memory-mapped I/O Direct Memory Access (DMA)

1. Polling

Polling is a software technique that uses a program to check the status of devices. The device can be a disk drive or any other peripheral device in the computer. The program polls the device for information, such as if it has data available or not. Polling is a slow way to get data from a device because it has to wait until another function occurs before being able to get information about its state.

In some cases polling may be desirable; for example, when there are several items being polled simultaneously and only one item updates its state at any time — the rest continue waiting until they receive an acknowledgment from one item that it's done updating their states (which could take seconds).

Polling can also be used to check if a device is online or not. If the device is offline, then this information can be used to take some appropriate action such as pausing or suspending tasks that depend on the device (e.g., stopping a backup in progress).

2. Interrupts

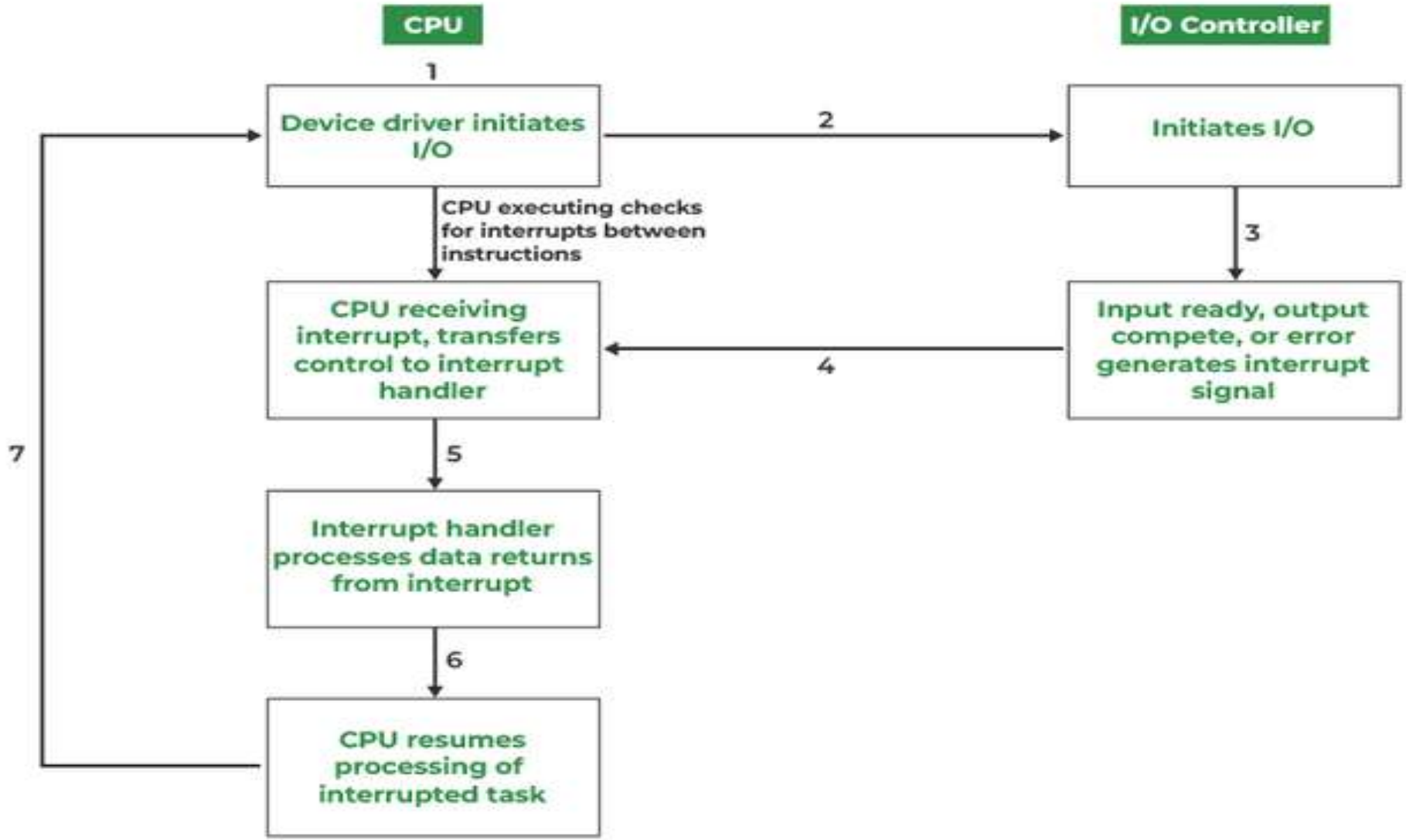
The CPU is interrupted by several different devices, such as I/O hardware and peripheral devices. These interrupts are used by the I/O device to notify the CPU when it needs attention. The CPU can be interrupted by several different devices at any given time, but only one interrupt will be delivered to it at any given time. This can happen because of a hardware or software error occurring on an I/O bus—for example, if a disk drive has failed then all other data transfers will pause until it's repaired; or perhaps another device wants access to memory (or vice versa). In either case, there won't be any way for your application program not to be written specifically for this particular machine!

Interrupt-driven I/O cycle

The operating system can also be interrupted by other processes that need CPU time. For example, if a user application program writes to a file and then exits, the operating system will write that file on disk before returning control back to the user. This is called multitasking and allows many different programs to run at once.

The operating system is responsible for scheduling the CPU's time among multiple programs. This is done by using a program called an interrupt handler or interrupt service routine (ISR). An ISR is simply a piece of code that runs when an interrupt occurs and typically ends with returning control back to the original program.

It's important to note that the operating system is never interrupted by an ISR.


CPU
I/O Controller
1
Device driver initiates I/O
2
Initiates I/O
CPU executing checks for interrupts between instructions
3
CPU receiving interrupt, transfers control to interrupt handler
4
Input ready, output compete, or error generates interrupt signal
5
7
Interrupt handler processes data returns from interrupt
6
CPU resumes processing of interrupted task

3. Direct Memory Access:

DMA is a way to move data between the CPU and I/O devices. When a user wants to access memory from an I/O device (such as a disk drive), the user must first inform the operating system that they are going to perform the operation. This is done by attaching a special command called an interrupt request (IRQ) handler routine for each device that needs access. The interrupt handler routine then tells the user program when it should be ready for another task—usually using interrupts generated by hardware components like joysticks or network cards—and may even tell other processes running on other CPUs not only about what's happening at that moment but also when something has happened before so they can take appropriate action as well!

The first step in using interrupts is to set up the interrupt controller. This is done by calling the BIOS INT 10h service with several parameters, including the number of IRQ lines available (16 is a common value), the maximum number of devices that can be connected to each line, and so on.

The next step is to assign an INT 10h handler for each device that needs access. This is done by calling the BIOS INT 15h service with several parameters, including the number of IRQ lines available (16 is a common value), the maximum number of devices that can be connected to each line, and so on