Operating System

Operating System

An Operating System(OS) is software that manages and handles the hardware and software resources of a computer system. It provides interaction between users of computers and computer hardware. An operating system is responsible for managing and controlling all the activities and sharing of computer resources. An operating system is a low-level Software that includes all the basic functions like processor management, memory management, Error detection, etc.

Functions of Operating System

1. Memory Management:

The operating system manages the Primary Memory or Main Memory. An operating system manages the allocation and deallocation of memory to various processes and ensures that the other process does not consume the memory allocated to one process.

In multiprogramming, the OS decides the order in which processes are granted memory access, and for how long.

Processor Management

An operating system manages the processor's work by allocating various jobs to it and ensuring that each process receives enough time from the processor to function properly.

Keeps track of the status of processes. The program which performs this task is known as a traffic controller. Allocates the CPU that is a processor to a process. De-allocates processor when a process is no longer required.

In a multi-programming environment, the OS decides the order in which processes have access to the processor, and how much processing time each process has. This function of OS is called Process Scheduling.

Device Management

An OS manages device communication via its respective drivers.

Keeps track of all devices connected to the system.

Decide which process gets access to a certain device and for how long.

It receives the requests from these devices, performs a specific task, and communicates back to the requesting process.

File Management

An OS keeps track of information regarding the creation, deletion, transfer, copy, and storage of files in an organized way. It also maintains the integrity of the data stored in these files, including the file directory structure, by protecting against unauthorized access.

Booting the Computer

The process of starting or restarting the computer is known as booting. If the computer is switched off completely and if turned on then it is called cold booting. Warm booting is a process of using the operating system to restart the computer.

Error-Detecting Aids

The operating system constantly monitors the system to detect errors and avoid malfunctioning computer systems. From time to time, the operating system checks the system for any external threat or malicious software activity. It also checks the hardware for any type of damage.

Security

The operating system uses password protection to protect user data and similar other techniques. it also prevents unauthorized access to programs and user data.

Process: An active program which running now on the Operating System is known as the process. The Process is the base of all computing things.

Files: A computer file is defined as a medium used for saving and managing data in the computer system. The data stored in the computer system is completely in digital format,there can be various types of files that help us to store the data. Like Text File, Image File, Video File, PDF file etc

System Calls:A system call is a mechanism used by programs to request services from the operating system (OS).It is a way for a program to interact with the underlying system, such as accessing hardware resources or performing operations.

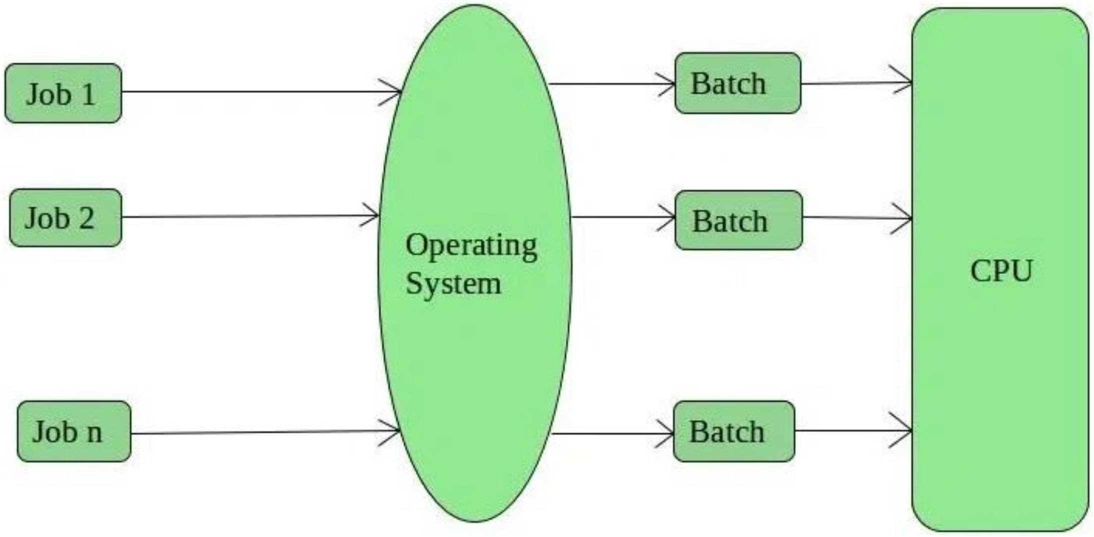
Shell: A shell is a program that provides a user interface to an operating system (OS). It allows you to interact with the computer by typing commands and executing them. It acts as a command-line interpreter, taking your input, interpreting it, and executing the corresponding actions.

Types of OS

1. Batch OS
2. Multi-Processing OS
3. Multi-Tasking OS
4. Time Sharing OS
5. Distributed OS
6. Network OS
7. Real Time OS

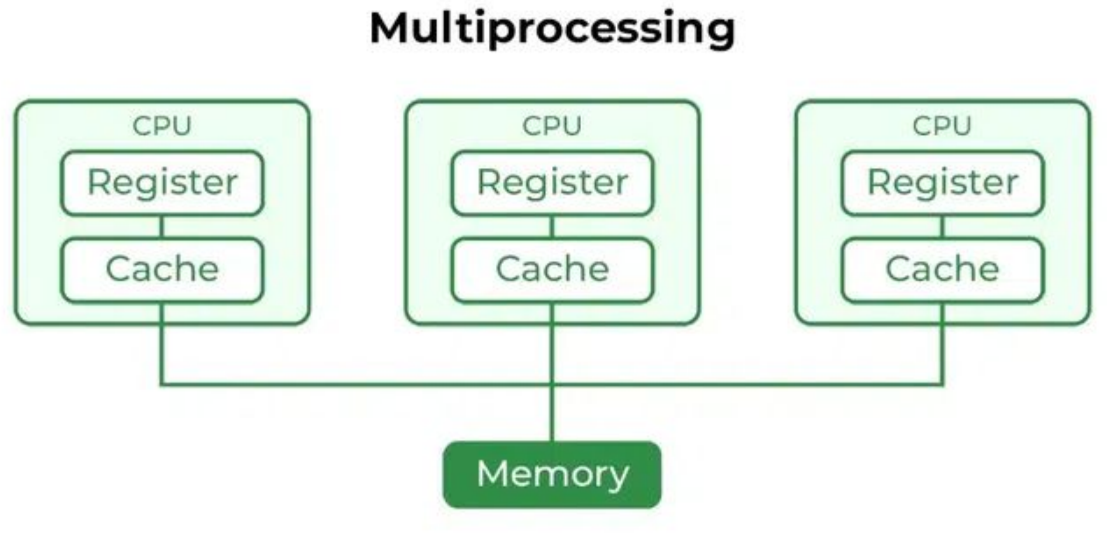
Batch OS

This type of operating system does not interact with the computer directly. There is an operator which takes similar jobs having the same requirements and groups them into batches. Batch Operating System is designed to manage and execute a large number of jobs efficiently by processing them in groups.



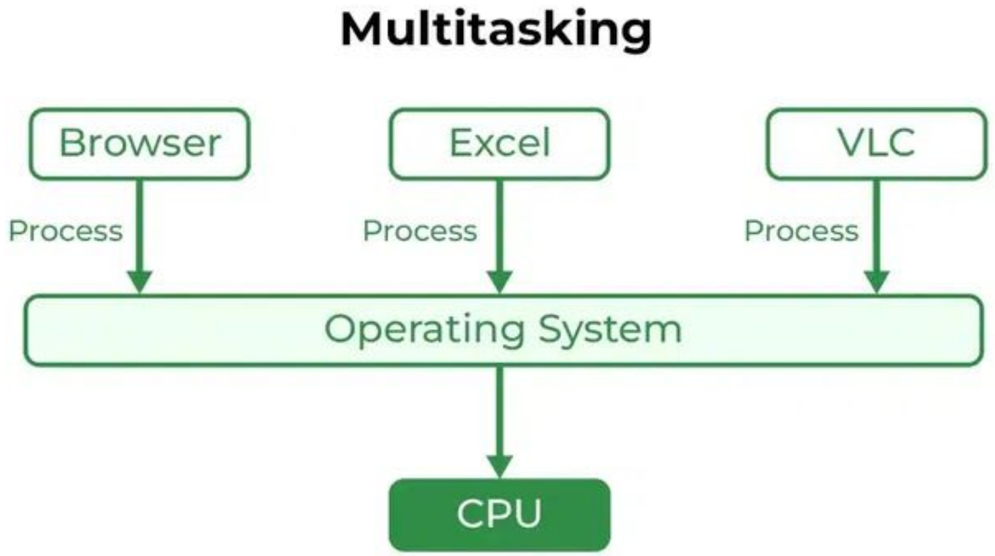
Multi-Processing OS

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



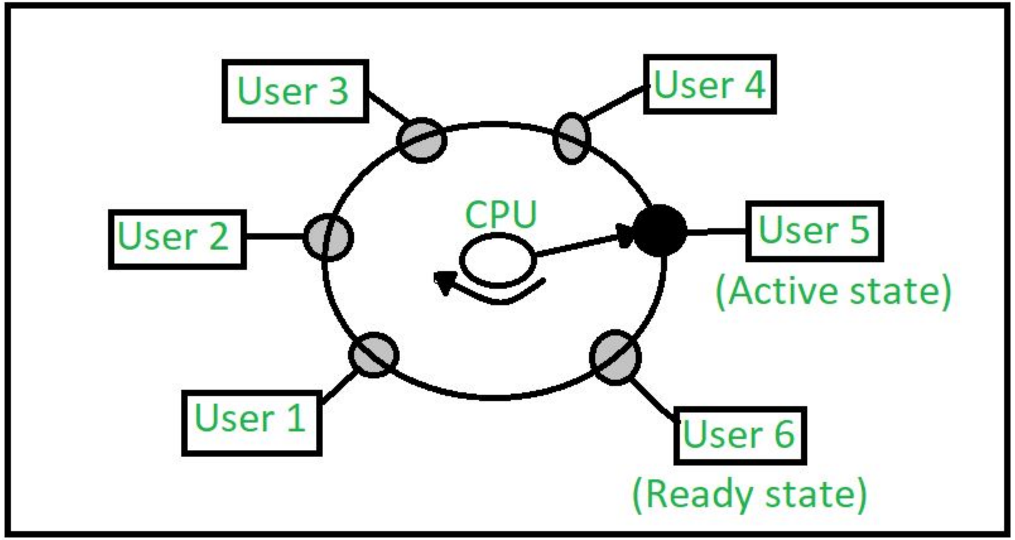
Multi-Tasking OS

A OS Capable of processing more than one task at time is known as multi-tasking OS. It has the facility of a Round-Robin Scheduling Algorithm.



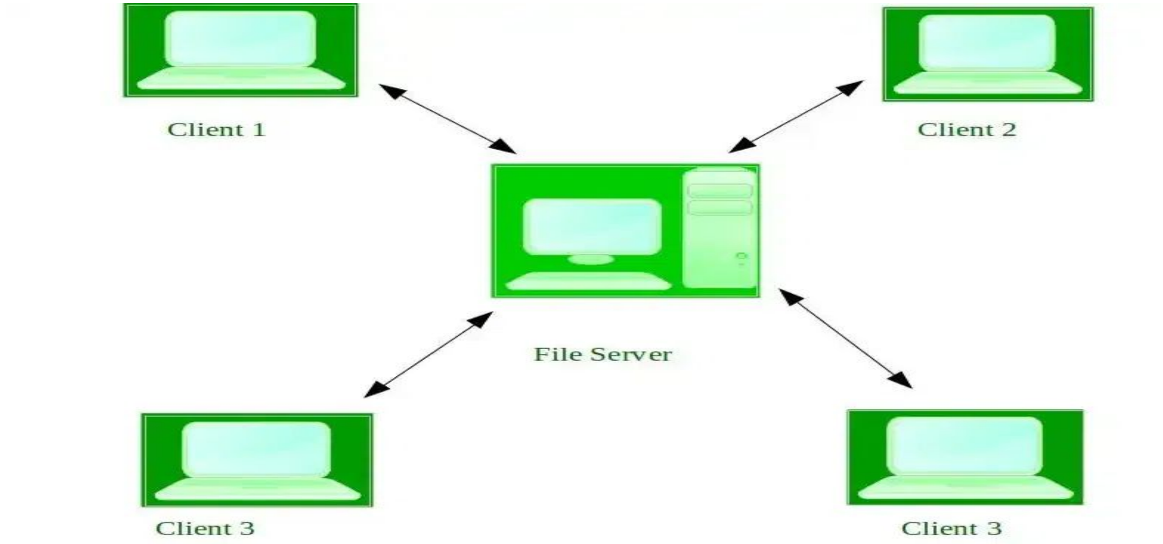
Time Sharing OS

In Time Sharing OS each task is given some time to execute so that all the tasks work smoothly. Each user gets the time of the CPU as they use a single system. The task can be from a single user or different users also. The time that each task gets to execute is called quantum. After this time interval is over OS switches over to the next task.



Network OS

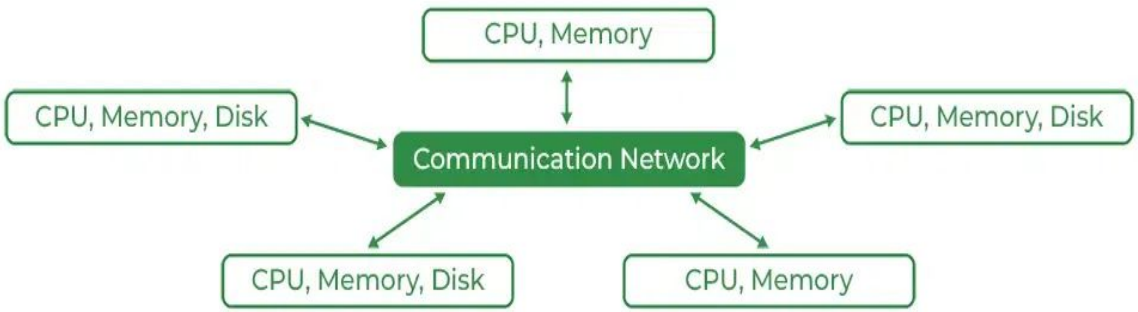
These systems run on a server and provide the capability to manage data, users, groups, security, applications, and other networking functions. These types of operating systems allow shared access to files, printers, security, applications, and other networking functions over a small private network.



Distributed Operating System

Various autonomous interconnected computers communicate with each other using a shared communication network. Independent systems possess their own memory unit and CPU.The major benefit of working with these types of the operating system is that it is always possible that one user can access the files or software which are not actually present on his system but some other system connected within this network

Architecture of Distributed OS



Real Time OS

A RTOS is a type of operating system that is designed to meet strict time constraints, with a guaranteed response time for critical tasks.

Hard Real-Time Systems: Hard Real-Time OSs are meant for applications where time constraints are very strict and even the shortest possible delay is not acceptable. These systems are built for saving life like automatic parachutes or airbags which are required to be readily available in case of an accident.

Soft Real-Time Systems: These OSs are for applications where time-constraint is less strict.A Soft Real-Time System is a variant of real-time system where timely processing is desirable but late processing do not cause the system to fail or can cause dramatic loss of life and property. However, failing to meet the time expectation, lead to a decline of service delivery or limited quality output. For Gaming, File Transfer, communication system.

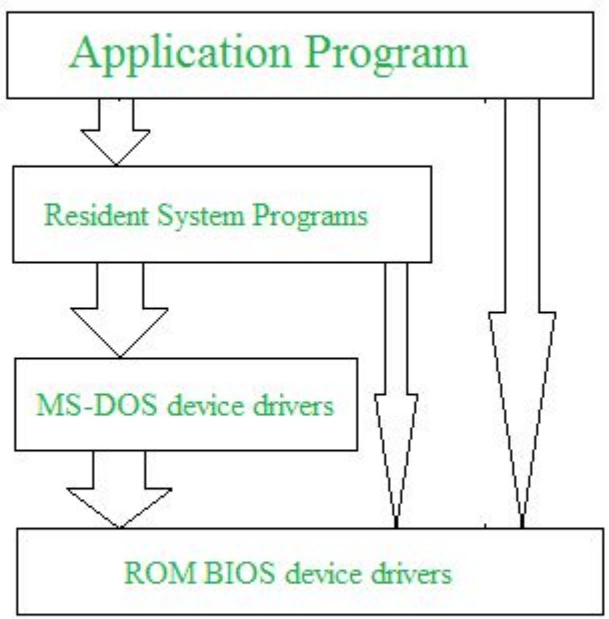
Operating System Structure

A system structure for an operating system is like the blueprint of how an OS is organized and how its different parts interact with each other. Because operating systems have complex structures, we want a structure that is easy to understand so that we can adapt an operating system to meet our specific needs.

1. Monolithic Structure
2. Layered Structure
3. Virtual Machines
4. Micro-Kernel Structure
5. Exo-Kernel Structure
6. Client-Server Structure

Monolithic structure

Monolithic operating systems do not have well-defined structures and are small, simple, and limited. The interfaces and levels of functionality are not well separated. MS-DOC is an example of such an operating system.



Micro-Kernel Structure

This structure designs the operating system by removing all non-essential components from the kernel and implementing them as system and user programs. This results in a smaller kernel called the micro-kernel.

- It makes the operating system portable to various platforms.
- As microkernels are small so these can be tested effectively.

Mac OS is an example of this type of OS.

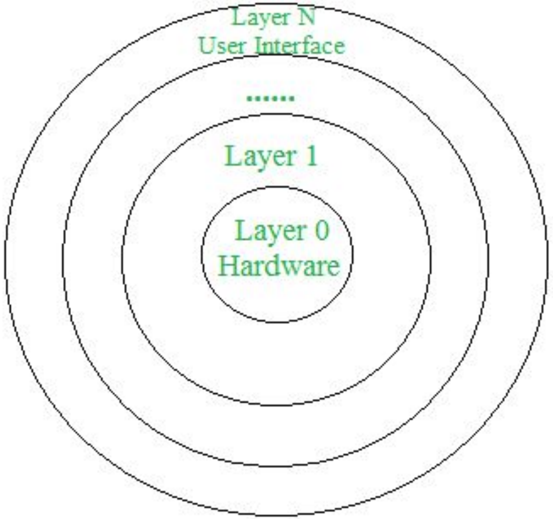
Exo-Kernel Structure

Exokernel is an operating system developed at MIT to provide application-level management of hardware resources. By separating resource management from protection, the exokernel architecture aims to enable application-specific customization. Due to its limited operability, exokernel size typically tends to be minimal.

- Support for improved application control.
- Separates management from security.
- It improves the performance of the application.

Layered Structure

An OS can be broken into pieces and retain much more control over the system. In this structure, the OS is broken into a number of layers (levels). The bottom layer (layer 0) is the hardware, and the topmost layer (layer N) is the user interface. These layers are so designed that each layer uses the functions of the lower-level layers. This simplifies the debugging process, if lower-level layers are debugged and an error occurs during debugging, then the error must be on that layer only, as the lower-level layers have already been debugged.



VMs (Virtual Machines)

A Virtual Machine is an Operating System or application environment that runs on software that replicates special hardware. The end-user experience when utilizing a virtual machine is identical to that of special hardware.

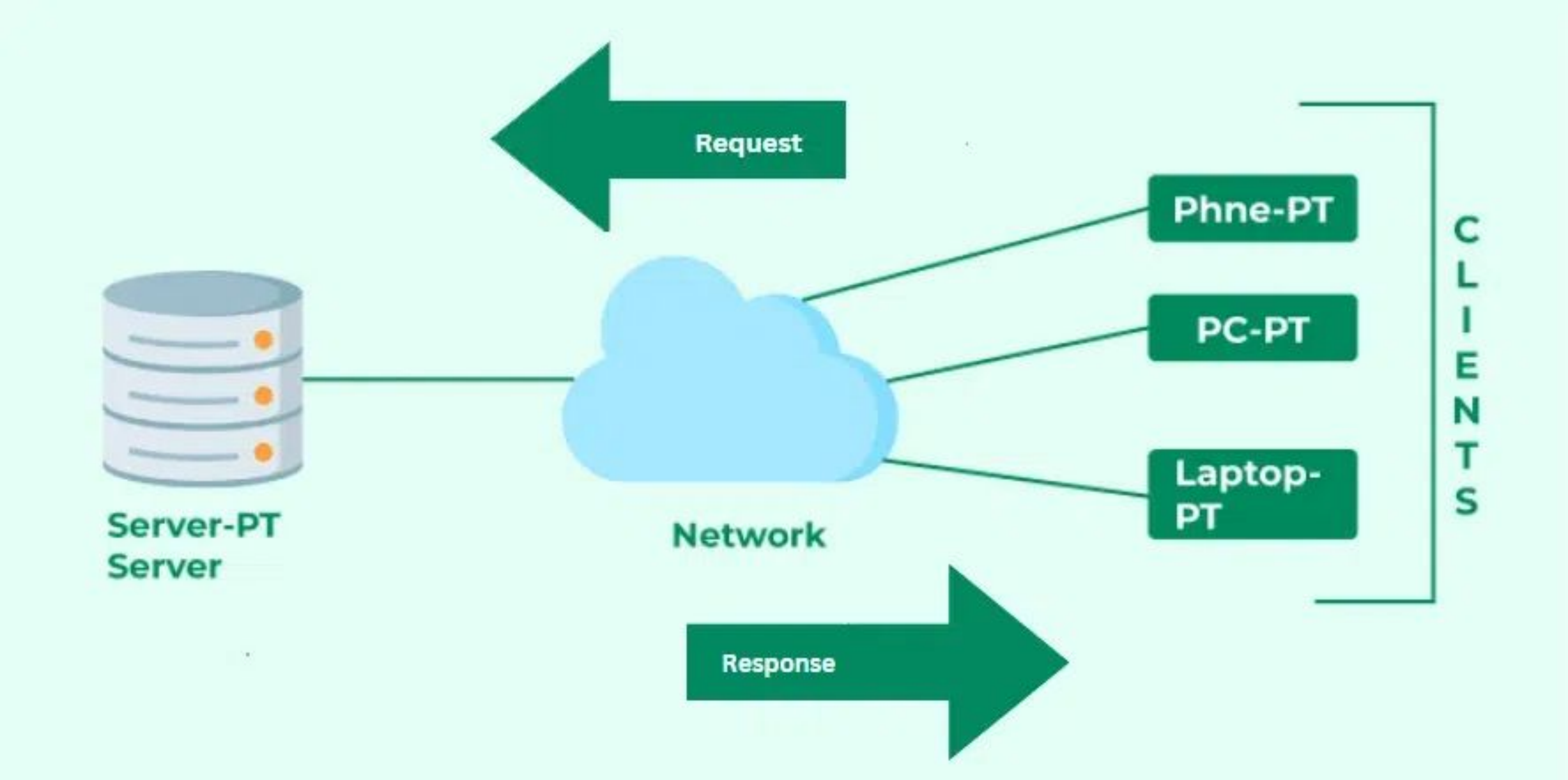
Virtual Machine abstracts the hardware of our personal computer such as CPU, disk drives, memory etc, into many different execution environments as per our requirements.. For example, VirtualBox. When we run different processes on an operating system, it creates an illusion that each process is running on a different processor having its own virtual memory, with the help of CPU scheduling and virtual-memory techniques.

Client-Server Model

The Client-server model is a distributed application structure that partitions tasks or workloads between the providers of a resource or service, called servers, and service requesters called clients. In the client-server architecture, when the client computer sends a request for data to the server through the internet, the server accepts the requested process and delivers the data packets requested back to the client. Clients do not share any of their resources. Examples of the Client-Server Model are Email, World Wide Web, etc.

Client is a computer (**Host**) i.e. capable of receiving information or using a particular service from the service providers (**Servers**).

Server is a remote computer that provides information (data) or access to particular services.


Request
Phne-PT
PC-PT
Laptop-PT
CLIENTS
Server-PT
Server
Network
Response

Evolution of Operating Systems

The operating system can be classified into four generations, as follows:

- First Generation(1945-1955)

  Serial Processing

  The evolution of operating systems began with serial processing. It marks the start of the development of electronic computing systems as alternatives to mechanical computers.

- Second Generation(1955-1965)

  Batch System

  The batched systems marked the second generation in the evolution of operating systems. In the second generation, the batch processing system was implemented, which allows a job or task to be done in a series and then completed sequentially.

Third Generation (1965-1980)

Multi-Programmed Batched System

The evolution of operating systems embarks the third generation with multi-programmed batched systems. In the third generation, the operating system was designed to serve numerous users simultaneously. Interactive users can communicate with a computer via an online terminal, making the operating system multi-user and multiprogramming.

Fourth Generation (1980-Now)

Graphical User Interface (GUI) was introduced. the time-sharing operating system and the Macintosh operating system came into existence.

Unit 2: Processes and Threads

A process is like a program that is running in a computer. A process is when that program is actually running and using the computer resources, like memory and CPU. In other words, the active state of a program is called the process. For example, when you open an app, like a browser or a game, the computer takes the program instructions and runs them. This running version of the program is called a process.

The components of a process are:

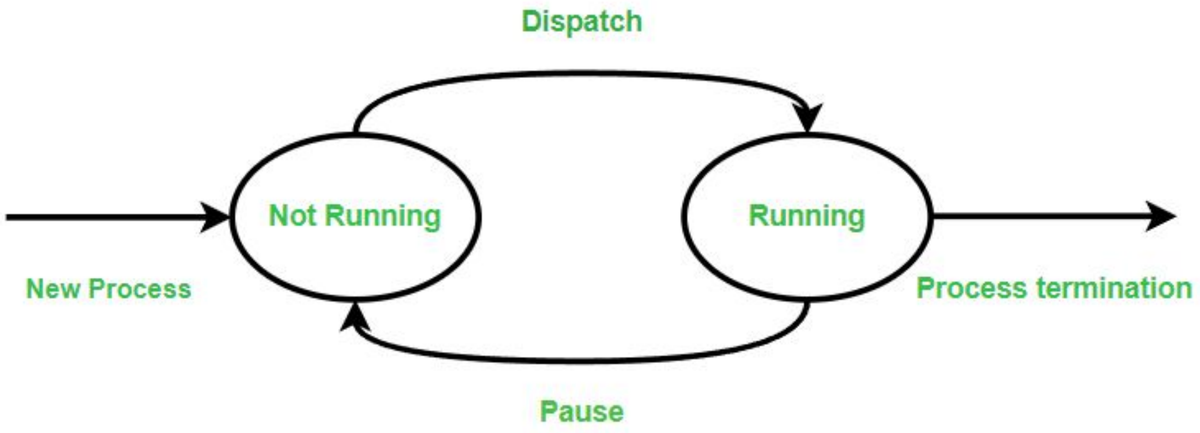
- Program code/Text: The instructions that the process will execute.
- Data: The data that the process will use during its execution.
- Stack: A data structure that is used to store temporary data, such as function parameters and return addresses.
- Heap: A data structure that is used to store dynamically allocated memory.
- Process control block (PCB): A data structure that contains information about the process, such as its state, priority, and memory usage.

States

When you run a program, it goes through different phases before it completion. These phases are called states, can vary depending on the operating system, but the most common process lifecycle includes two, five, or seven states.

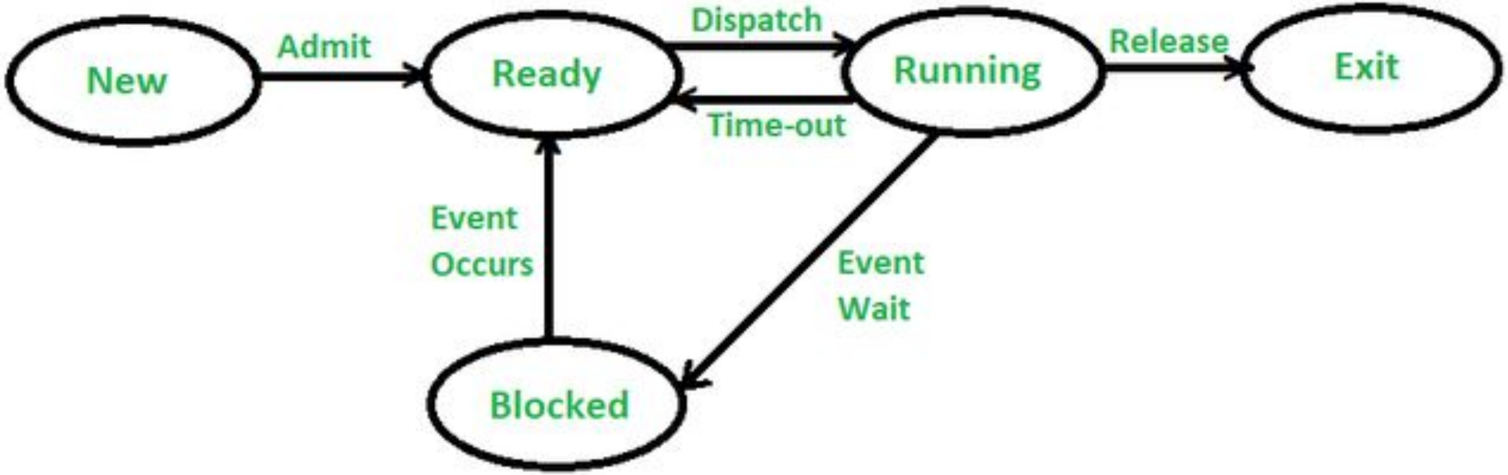
Two-State Model

1. **Running**: This means the process is actively using the CPU to do its work.
2. **Not Running**: This means the process is not currently using the CPU. It could be waiting for something, like user input or data, or it might just be paused.

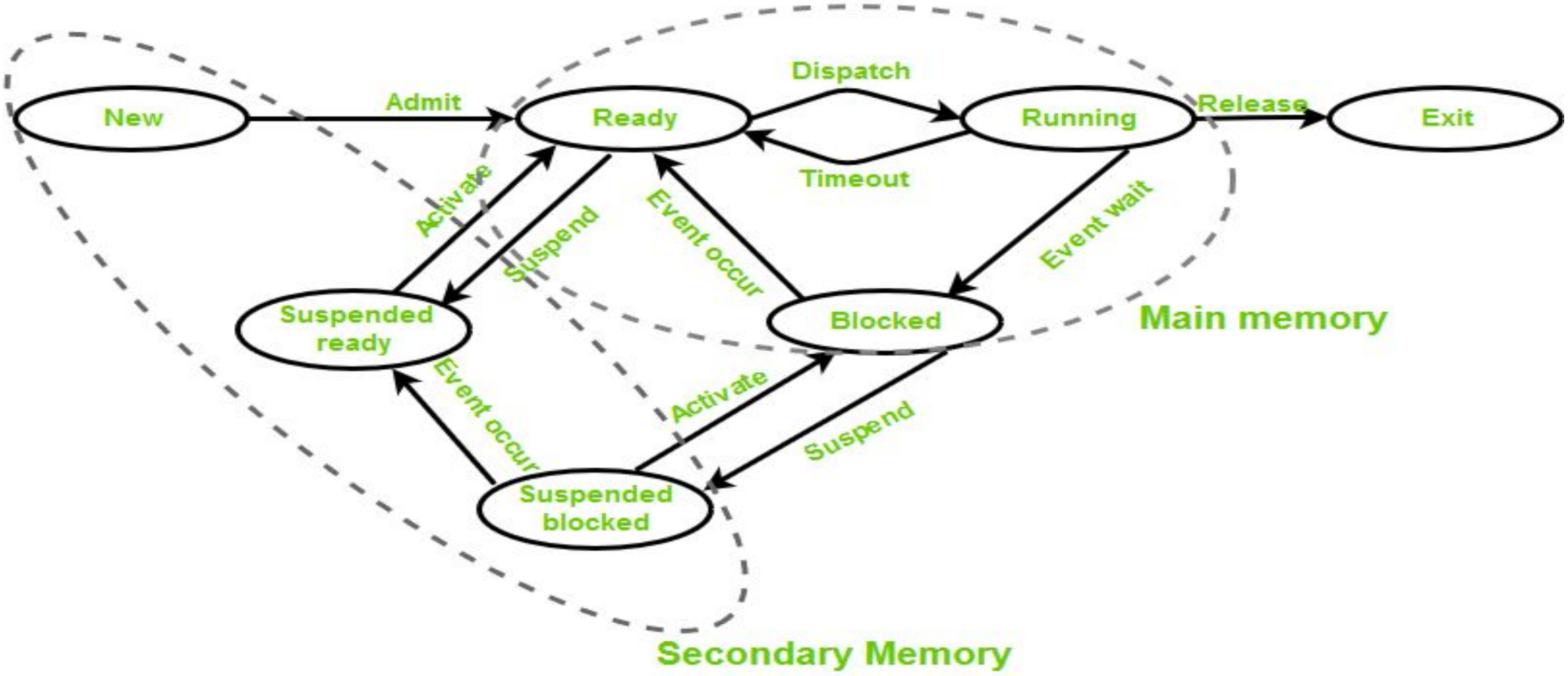


The Five-State Model

- **New:** This state represents a newly created process that hasn't started running yet. It has not been loaded into the main memory, but its process control block (PCB) has been created, which holds important information about the process.
- **Ready:** A process in this state is ready to run as soon as the CPU becomes available. It is waiting for the operating system to give it a chance to execute.
- **Running**: This state means the process is currently being executed by the CPU. Since we're assuming there is only one CPU, at any time, only one process can be in this state.
- **Blocked/Waiting:** This state means the process cannot continue executing right now. It is waiting for some event to happen, like the completion of an input/output operation (for example, reading data from a disk).
- **Exit/Terminate:** A process in this state has finished its execution or has been stopped by the user for some reason. At this point, it is released by the operating system and removed from memory.


New
Admit
Ready
Dispatch
Running
Release
Exit
Time-out
Event
Occurs
Event
Wait
Blocked

Seven-State Model

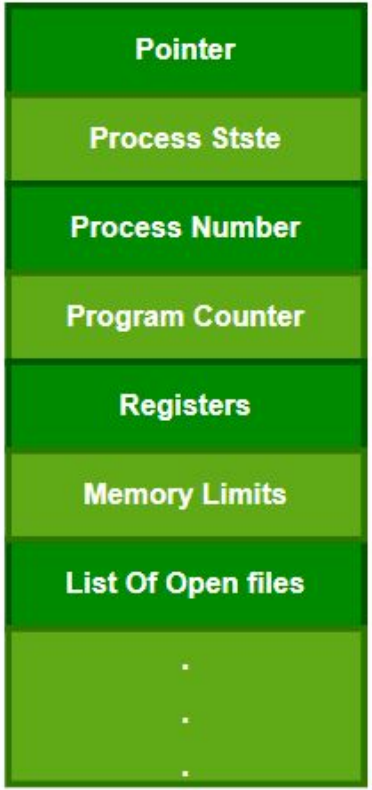


Process Control Block(PCB)

A Process Control Block (PCB) is a data structure used by the operating system to manage information about a process. The process control keeps track of many important pieces of information needed to manage processes efficiently.

While creating a process, the operating system performs several operations. To identify the processes, it assigns a process identification number (PID) to each process. As the operating system supports multi-programming, it needs to keep track of all the processes. For this task, the process control block (PCB) is used to track the process's execution status. Each block of memory contains information about the process state, program counter, stack pointer, status of opened files, scheduling algorithms, etc

Process Control Block



Pointer

Process Stste

Process Number

Program Counter

Registers

Memory Limits

List Of Open files

.

.

.

- **Pointer:** It is a stack pointer that is required to be saved when the process is switched from one state to another to retain the current position of the process.
- **Process state:** It stores the respective state of the process.
- **Process number:** Every process is assigned a unique id known as process ID or PID which stores the process identifier.
- **Program counter:** Program Counter stores the counter, which contains the address of the next instruction that is to be executed for the process.
- **Register:** Registers in the PCB, it is a data structure. When a processes is running and it's time slice expires, the current value of process specific registers would be stored in the PCB and the process would be swapped out. When the process is scheduled to be run, the register values is read from the PCB and written to the CPU registers. This is the main purpose of the registers in the PCB.
- **Memory limits:** This field contains the information about memory management system used by the operating system. This may include page tables, segment tables, etc.
- **List of Open files:** This information includes the list of files opened for a process.

Advantages

- **Efficient Process Management:** The process table and PCB provide an efficient way to manage processes in an operating system. The process table contains all the information about each process, while the PCB contains the current state of the process, such as the program counter and CPU registers.
- **Resource Management:** The process table and PCB allow the operating system to manage system resources, such as memory and CPU time, efficiently. By keeping track of each process’s resource usage, the operating system can ensure that all processes have access to the resources they need.
- **Process Synchronization:** The process table and PCB can be used to synchronize processes in an operating system. The PCB contains information about each process’s synchronization state, such as its waiting status and the resources it is waiting for.
- **Process Scheduling:** The process table and PCB can be used to schedule processes for execution. By keeping track of each process’s state and resource usage, the operating system can determine which processes should be executed next.

Concurrent processing is a computing model in which multiple processors execute instructions simultaneously for better performance. Concurrent means, which occurs when something else happens. The tasks are broken into subtypes, which are then assigned to different processors to perform simultaneously, sequentially instead, as they would have to be performed by one processor.

Process scheduling

Process scheduling is the activity of the process manager that handles the removal of the running process from the CPU and the selection of another process based on a particular strategy.

Process scheduling is an essential part of a Multiprogramming operating system. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

- **Non-Preemptive:** In this case, a process's resource cannot be taken before the process has finished running. When a running process finishes and transitions to a waiting state, resources are switched.
- **Preemptive:** In this case, the OS assigns resources to a process for a predetermined period. The process switches from running state to ready state or from waiting state to ready state during resource allocation. This switching happens because the CPU may give other processes priority and substitute the currently active process for the higher priority process.

Batch scheduling is a manufacturing approach wherein products are assembled in groups, referred to as “batches”. In this technique, each step in the production process is simultaneously applied to a group of items, and the batch progresses to the next stage only after the entire batch is completed.

Interactive Process Scheduling

The main goal of interactive process scheduling is to ensure quick responses to user inputs. To achieve this, preemptive scheduling is often used, which allows the system to switch between processes as needed to keep everything running smoothly.

An interactive process allows users to engage with the system. It accepts commands or data through devices like keyboards and mice, and provides output on screens.

Real Time Scheduling

Real-time systems are systems that carry real-time tasks. These tasks need to be performed immediately with a certain degree of urgency. In particular, these tasks are related to control of certain events (or) reacting to them. Real-time tasks can be classified as hard real-time tasks and soft real-time tasks.

A hard real-time task must be performed at a specified time which could otherwise lead to huge losses. In soft real-time tasks, a specified deadline can be missed. This is because the task can be rescheduled (or) can be completed after the specified time, In real-time systems, the scheduler is considered as the most important component which is typically a short-term task scheduler. The main focus of this scheduler is to reduce the response time associated with each of the associated processes instead of handling the deadline.

Criteria of CPU Scheduling

1. CPU utilization

The main objective of any CPU scheduling algorithm is to keep the CPU as busy as possible.

2. Throughput

A measure of the work done by the CPU is the number of processes being executed and completed per unit of time. This is called throughput. The throughput may vary depending on the length or duration of the processes.

3. Turnaround Time

For a particular process, an important criterion is how long it takes to execute that process. The time elapsed from the time of submission of a process to the time of completion is known as the turnaround time. Turn-around time is the sum of times spent waiting to get into memory, waiting in the ready queue, executing in CPU, and waiting for I/O.

4. Waiting Time

A scheduling algorithm does not affect the time required to complete the process once it starts execution. It only affects the waiting time of a process i.e. time spent by a process waiting in the ready queue.

5. Response Time

In an interactive system, turn-around time is not the best criterion. A process may produce some output fairly early and continue computing new results while previous results are being output to the user. Thus another criterion is the time taken from submission of the process of the request until the first response is produced. This measure is called response time.

6. Completion Time

The completion time is the time when the process stops executing, which means that the process has completed its burst time and is completely executed.

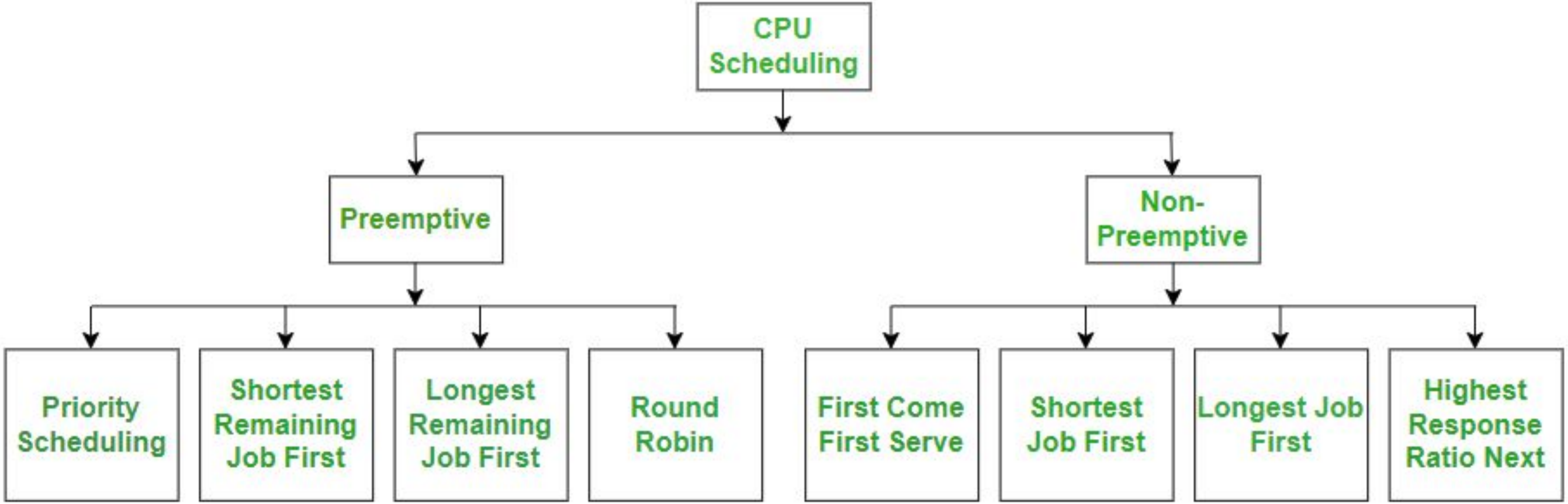
7. Priority

If the operating system assigns priorities to processes, the scheduling mechanism should favor the higher-priority processes.

8. Predictability

A given process always should run in about the same amount of time under a similar system load.

Scheduling Algorithms



FCFS considered to be the simplest of all operating system scheduling algorithms. First come first serve scheduling algorithm states that the process that requests the CPU first is allocated the CPU first and is implemented by using FIFO queue.

Shortest job first (SJF) is a scheduling process that selects the waiting process with the smallest execution time to execute next. This scheduling method may or may not be preemptive. Significantly reduces the average waiting time for other processes waiting to be executed.

Shortest remaining time first is the preemptive version of the Shortest job first which we have discussed earlier where the processor is allocated to the job closest to completion. In SRTF the process with the smallest amount of time remaining until completion is selected to execute.

Priority CPU Scheduling Algorithm is a pre-emptive method of CPU scheduling algorithm that works based on the priority of a process. In this algorithm, the editor sets the functions to be as important, meaning that the most important process must be done first. In the case of any conflict, that is, where there is more than one process with equal value, then the most important CPU planning algorithm works on the basis of the FCFS (First Come First Serve) algorithm.

Round Robin

Round Robin is a CPU scheduling algorithm where each process is cyclically assigned a fixed time slot. It is the preemptive version of First come First Serve CPU Scheduling algorithm. Round Robin CPU Algorithm generally focuses on Time Sharing technique.

Shortest remaining time first is the preemptive version of the Shortest job first which we have discussed earlier where the processor is allocated to the job closest to completion. In SRTF the process with the smallest amount of time remaining until completion is selected to execute.

Inter Process Communication (IPC)

Processes can coordinate and interact with one another using a method called **inter-process communication (IPC)** . Through facilitating process collaboration, it significantly contributes to improving the effectiveness, modularity, and ease of software systems.

The communication between these processes can be seen as a method of cooperation between them. Processes can communicate with each other through both: Methods of IPC

- **Shared Memory**
- **Message Passing**

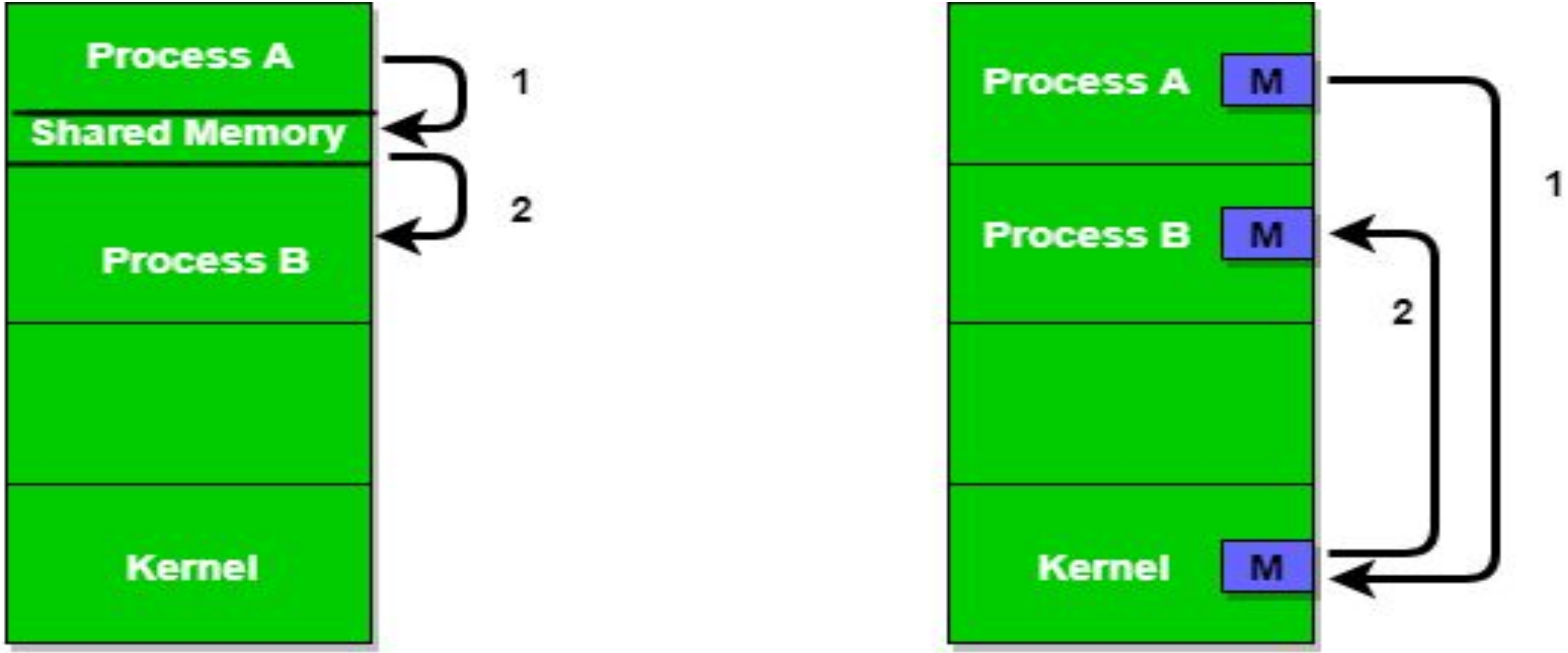


Figure 1 - Shared Memory and Message Passing

Communication between processes using shared memory requires processes to share some variable, and it completely depends on how the programmer will implement it. One way of communication using shared memory can be imagined like this: Suppose process1 and process2 are executing simultaneously, and they share some resources or use some information from another process. Process1 generates information about certain computations or resources being used and keeps it as a record in shared memory. When process2 needs to use the shared information, it will check in the record stored in shared memory and take note of the information generated by process1 and act accordingly. Processes can use shared memory for extracting information as a record from another process as well as for delivering any specific information to other processes.

Message Passing Method

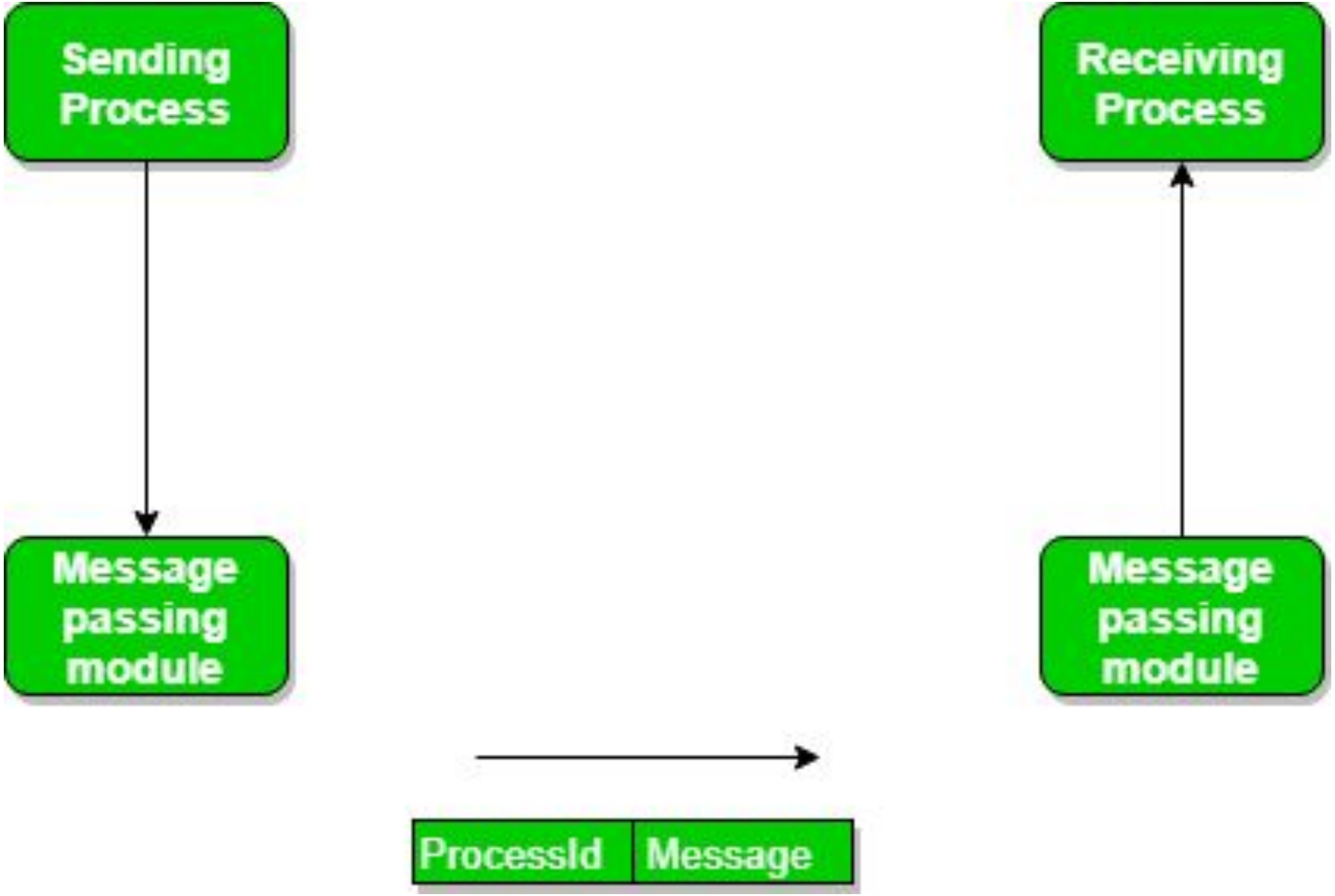
In this method, processes communicate with each other without using any kind of shared memory. If two processes p1 and p2 want to communicate with each other, they proceed as follows:

- Establish a communication link (if a link already exists, no need to establish it again.)
- Start exchanging messages using basic primitives.

  We need at least two primitives:

  – **send** (message, destination) or **send** (message)

  – **receive** (message, host) or **receive** (message)


Sending Process
Receiving Process
Message passing module
Message passing module
ProcessId
Message

The message size can be of fixed size or of variable size. If it is of fixed size, it is easy for an OS designer but complicated for a programmer and if it is of variable size then it is easy for a programmer but complicated for the OS designer. A standard message can have two parts: header and body. The header part is used for storing message type, destination id, source id, message length, and control information.

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.

The main objective of process synchronization is to ensure that multiple processes access shared resources without interfering with each other and to prevent the possibility of inconsistent data due to concurrent access. To achieve this, various synchronization techniques such as semaphores, monitors, and critical sections are used.

In a multi-process system, synchronization is necessary to ensure data consistency and integrity, and to avoid the risk of deadlocks and other synchronization problems. Process synchronization is an important aspect of modern operating systems, and it plays a crucial role in ensuring the correct and efficient functioning of multi-process systems.

What is Race Condition?

When more than one process is executing the same code or accessing the same memory or any shared variable in that condition there is a possibility that the output or the value of the shared variable is wrong so for that all the processes doing the race to say that my output is correct this condition known as a race condition.

How can race conditions be avoided?

Race conditions can be avoided by implementing mutual exclusion techniques, such as locks, semaphores, or monitors. These techniques ensure that only one process or thread can access a shared resource at a time, preventing simultaneous conflicting accesses and maintaining data integrity.

Critical Regions

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.

What is a deadlock?

A deadlock is a situation in concurrent programming where two or more processes or threads are unable to proceed because each is waiting for the other to release a resource. As a result, the processes or threads are stuck in a state of waiting indefinitely, leading to a system freeze or unresponsiveness.

Critical Region Characteristics and Requirements

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.

What is Mutual Exclusion?

Mutual Exclusion is a property of process synchronization that states that “no two processes can exist in the critical section at any given point of time“. Any process synchronization technique being used must satisfy the property of mutual exclusion, without which it would not be possible to get rid of a race condition.

The need for mutual exclusion comes with concurrency. There are several kinds of concurrent execution:

- Interrupt handlers
- Interleaved, preemptively scheduled processes/threads
- Multiprocessor clusters, with shared memory
- Distributed systems

Mutual exclusion methods are used in concurrent programming to avoid the simultaneous use of a common resource, such as a global variable, by pieces of computer code called critical sections.

The requirement of mutual exclusion is that when process P1 is accessing a shared resource R1, another process should not be able to access resource R1 until process P1 has finished its operation with resource R1.

Examples of such resources include files, I/O devices such as printers, and shared data structures.

Conditions Required for Mutual Exclusion

According to the following four criteria, mutual exclusion is applicable:

- When using shared resources, it is important to ensure mutual exclusion between various processes. There cannot be two processes running simultaneously in either of their critical sections.
- It is not advisable to make assumptions about the relative speeds of the unstable processes.
- For access to the critical section, a process that is outside of it must not obstruct another process.
- Its critical section must be accessible by multiple processes in a finite amount of time; multiple processes should never be kept waiting in an infinite loop.

Approaches To Implementing Mutual Exclusion

- **Software Method:** Leave the responsibility to the processes themselves. These methods are usually highly error-prone and carry high overheads.
- **Hardware Method:** Special-purpose machine instructions are used for accessing shared resources. This method is faster but cannot provide a complete solution. Hardware solutions cannot give guarantee the absence of deadlock and starvation.
- **Programming Language Method:** Provide support through the operating system or through the programming language.

Requirements of Mutual Exclusion

- At any time, only one process is allowed to enter its critical section.
- The solution is implemented purely in software on a machine.
- A process remains inside its critical section for a bounded time only.
- No assumption can be made about the relative speeds of asynchronous concurrent processes.
- A process cannot prevent any other process from entering into a critical section.
- A process must not be indefinitely postponed from entering its critical section.

Types of Mutual Exclusion

1. Semaphores are just normal variables used to coordinate the activities of multiple processes in a computer system. They are used to enforce mutual exclusion, avoid race conditions, and implement synchronization between processes.

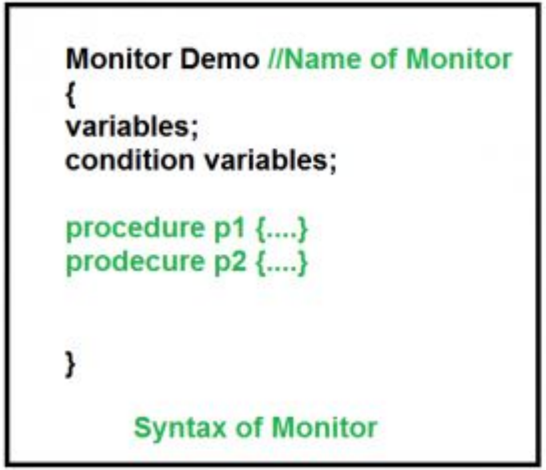
The process of using Semaphores provides two operations: wait (P) and signal (V). The wait operation decrements the value of the semaphore, and the signal operation increments the value of the semaphore. When the value of the semaphore is zero, any process that performs a wait operation will be blocked until another process performs a signal operation.

When a process performs a wait operation on a semaphore, the operation checks whether the value of the semaphore is >0. If so, it decrements the value of the semaphore and lets the process continue its execution; otherwise, it blocks the process on the semaphore. A signal operation on a semaphore activates a process blocked on the semaphore if any, or increments the value of the semaphore by 1. Due to these semantics, semaphores are also called counting semaphores. The initial value of a semaphore determines how many

```
Wait(){
while(1){
Atomic {
If (v>0) {
V - -;
Return;
}}}}
```

```
Post(){
atomic{
v=v++;
return;
}}
```

2. Monitors are a higher-level synchronization construct that simplifies process synchronization by providing a high-level abstraction for data access and synchronization. Monitors are implemented as programming language constructs, typically in object-oriented languages, and provide mutual exclusion, condition variables, and data encapsulation in a single construct.



```
Monitor Demo //Name of Monitor
{
variables;
condition variables;

procedure p1 {....}
prodecure p2 {....}


}
```

Syntax of Monitor

3. mutual exclusion by using interrupt disabling

Whenever the interrupts are disabled, it effectively stops scheduling other processes. Whenever disabling interrupts, the CPU will be unable to switch processes and processes can use shared variables without another process accessing it.

The most obvious way to achieve mutual exclusion is to allow a process to disable interrupts before it enters critical sections. Uniprocessor refers to the operation being atomic as long as context switches do not occur.

There are basically two ways to control dispatcher which are as follows −

Internal events − Thread does something to relinquish the CPU.
External events − Interrupts can cause dispatchers to take the CPU away.

So, it is needed to prevent internal and external events as follows −

To prevent internal events is easy.
To prevent external events requires disabling interrupts.

4. lock variable provides the simplest synchronization mechanism for processes. Some noteworthy points regarding Lock Variables are-

1. Its a **software mechanism** implemented in user mode, i.e. no support required from the Operating System.
2. Its a busy waiting solution (keeps the CPU busy even when its technically waiting).
3. It can be used for more than two processes.

When Lock = 0 implies critical section is vacant (initial value) and Lock = 1 implies critical section occupied.
The pseudocode looks something like this –

```
Entry section - while(lock != 0);
                Lock = 1;
//critical section
  Exit section - Lock = 0;
```

Turn Variable or Strict Alternation Approach

Turn Variable or Strict Alternation Approach is the software mechanism implemented at user mode. It is a busy waiting solution which can be implemented only for two processes. In this approach, A turn variable is used which is actually a lock.

This approach can only be used for only two processes. In general, let the two processes be Pi and Pj. They share a variable called turn variable. The pseudo code of the program can be given as following.

For Process Pi

```
    Non - CS
    while (turn ! = i);
    Critical Section
    turn = j;
    Non - CS
```

For Process Pj

```
Non - CS
while (turn ! = j);
Critical Section
turn = i ;
Non - CS
```

Dekker’s algorithm in Process Synchronization

Dekker’s algorithm is the first solution of critical section problem. There are many versions of this algorithms, the 5th or final version satisfies the all the conditions below and is the most efficient among all of them.

The solution to critical section problem must ensure the following three conditions:

Mutual Exclusion
Progress
Bounded Waiting

Serializability

Serializability is a concept in operating systems (OS) that ensures the consistency of a database's final state. It's used to prevent data inconsistencies and anomalies that can occur when multiple transactions try to access the same data simultaneously.

How it works

- Serializability schedules transactions in the order they are executed.
- It ensures that multiple transactions can access and modify data without interfering with each other.
- It prevents data race conditions and concurrency problems like lost updates.

Benefits

- Improves database reliability
- Allows databases to scale to handle large amounts of data
- Ensures data accuracy, privacy, and security

1. Conflict Serializability

Conflict serializability is a type of conflict operation in serializability that operates the same data item that should be executed in a particular order and maintains the consistency of the database. In DBMS, each transaction has some unique value, and every transaction of the database is based on that unique value of the database.

This unique value ensures that no two operations having the same conflict value are executed concurrently. There is some condition for the conflict serializability of the database. These are as below.

- Both operations should have different transactions.
- Both transactions should have the same data item.
- There should be at least one write operation between the two operations.

2. View Serializability

View serializability is a type of operation in the serializable in which each transaction should produce some result and these results are the output of proper sequential execution of the data item. Unlike conflict serialized, the view serializability focuses on preventing inconsistency in the database. In DBMS, the view serializability provides the user to view the database in a conflicting way.

Lock-Based Protocol

In this type of protocol, any transaction cannot read or write data until it acquires an appropriate lock on it. There are two types of lock:

1. Shared lock:

- It is also known as a Read-only lock. In a shared lock, the data item can only read by the transaction.
- It can be shared between the transactions because when the transaction holds a lock, then it can't update the data on the data item.

2. Exclusive lock:

- In the exclusive lock, the data item can be both reads as well as written by the transaction.
- This lock is exclusive, and in this lock, multiple transactions do not modify the same data simultaneously.

There are four types of lock protocols available:

1. Simplistic lock protocol

It is the simplest way of locking the data while transaction. Simplistic lock-based protocols allow all the transactions to get the lock on the data before insert or delete or update on it. It will unlock the data item after completing the transaction.

2. Pre-claiming Lock Protocol

- Pre-claiming Lock Protocols evaluate the transaction to list all the data items on which they need locks.
- Before initiating an execution of the transaction, it requests DBMS for all the lock on all those data items.
- If all the locks are granted then this protocol allows the transaction to begin. When the transaction is completed then it releases all the lock.
- If all the locks are not granted then this protocol allows the transaction to rolls back and waits until all the locks are granted.

3. Two-phase locking (2PL)

- The two-phase locking protocol divides the execution phase of the transaction into three parts.
- In the first part, when the execution of the transaction starts, it seeks permission for the lock it requires.
- In the second part, the transaction acquires all the locks. The third phase is started as soon as the transaction releases its first lock.
- In the third phase, the transaction cannot demand any new locks. It only releases the acquired locks.

There are two phases of 2PL:

Growing phase: In the growing phase, a new lock on the data item may be acquired by the transaction, but none can be released.

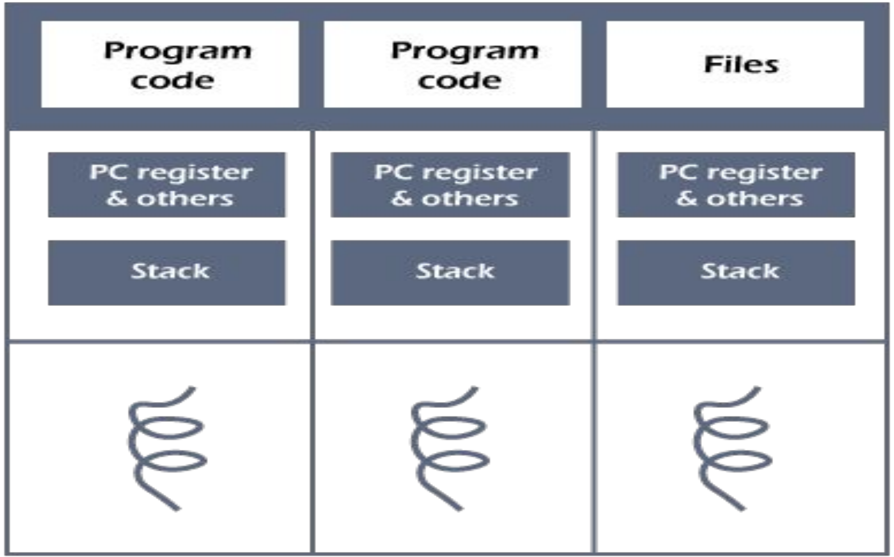
Shrinking phase: In the shrinking phase, existing lock held by the transaction may be released, but no new locks can be acquired.

4. Strict Two-phase locking (Strict-2PL)

- The first phase of Strict-2PL is similar to 2PL. In the first phase, after acquiring all the locks, the transaction continues to execute normally.
- The only difference between 2PL and strict 2PL is that Strict-2PL does not release a lock after using it.
- Strict-2PL waits until the whole transaction to commit, and then it releases all the locks at a time.
- Strict-2PL protocol does not have shrinking phase of lock release.

What is Thread

A thread is a single sequence stream within a process. Threads are also called lightweight processes as they possess some of the properties of processes. Each thread belongs to exactly one process. In an operating system that supports multithreading, the process can consist of many threads. But threads can be effective only if the CPU is more than 1 otherwise two threads have to context switch for that single CPU.



For example, in a browser, many tabs can be viewed as threads. MS Word uses many threads - formatting text from one thread, processing input from another thread, etc.

Why Do We Need Thread?

- Threads run in parallel improving the application performance. Each such thread has its own CPU state and stack, but they share the address space of the process and the environment.
- Threads can share common data so they do not need to use inter-process communication. Like the processes, threads also have states like ready, executing, blocked, etc.
- Priority can be assigned to the threads just like the process, and the highest priority thread is scheduled first.
- Each thread has its own Thread Control Block (TCB). Like the process, a context switch occurs for the thread, and register contents are saved in (TCB). As threads share the same address space and resources, synchronization is also required for the various activities of the thread.

Types of Thread in Operating System

1. User Level Threads

User Level Thread is a type of thread that is not created using system calls. The kernel has no work in the management of user-level threads. User-level threads can be easily implemented by the user. In case when user-level threads are single-handed processes, kernel-level thread manages them.

- User-level threads are implemented using user-level libraries and the OS does not recognize these threads.
- User-level thread is faster to create and manage compared to kernel-level thread.
- Context switching in user-level threads is faster.
- If one user-level thread performs a blocking operation then the entire process gets blocked. Eg: POSIX threads, Java threads, etc.

Advantages of User-level threads

1. The user threads can be easily implemented than the kernel thread.
2. User-level threads can be applied to such types of operating systems that do not support threads at the kernel-level.
3. It is faster and efficient.
4. Context switch time is shorter than the kernel-level threads.
5. It does not require modifications of the operating system.
6. User-level threads representation is very simple. The register, PC, stack, and mini thread control blocks are stored in the address space of the user-level process.
7. It is simple to create, switch, and synchronize threads without the intervention of the process.

Disadvantages of User-level threads

1. User-level threads lack coordination between the thread and the kernel.
2. If a thread causes a page fault, the entire process is blocked.

2. Kernel Level Threads

A kernel Level Thread is a type of thread that can recognize the Operating system easily. Kernel Level Threads has its own thread table where it keeps track of the system. The operating System Kernel helps in managing threads. Kernel Threads have somehow longer context switching time. Kernel helps in the management of threads.

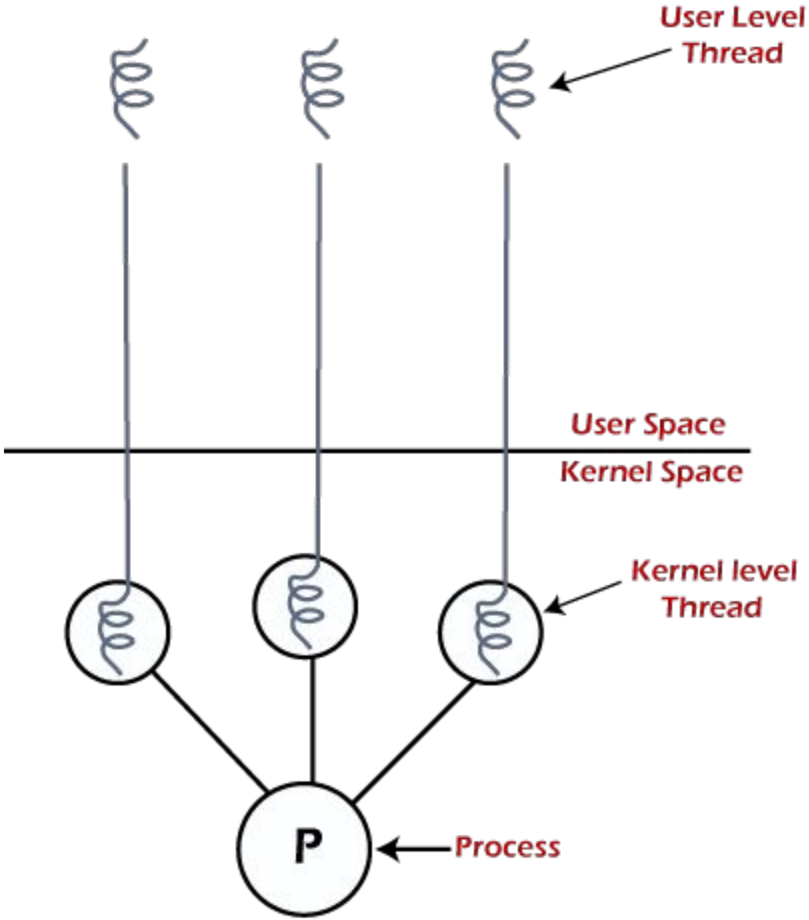
- Kernel level threads are implemented using system calls and Kernel level threads are recognized by the OS.
- Kernel-level threads are slower to create and manage compared to user-level threads.
- Context switching in a kernel-level thread is slower.
- Even if one kernel-level thread performs a blocking operation, it does not affect other threads. Eg: Window Solaris.

Advantages of Kernel-level threads

1. The kernel-level thread is fully aware of all threads.
2. The scheduler may decide to spend more CPU time in the process of threads being large numerical.
3. The kernel-level thread is good for those applications that block the frequency.

Disadvantages of Kernel-level threads

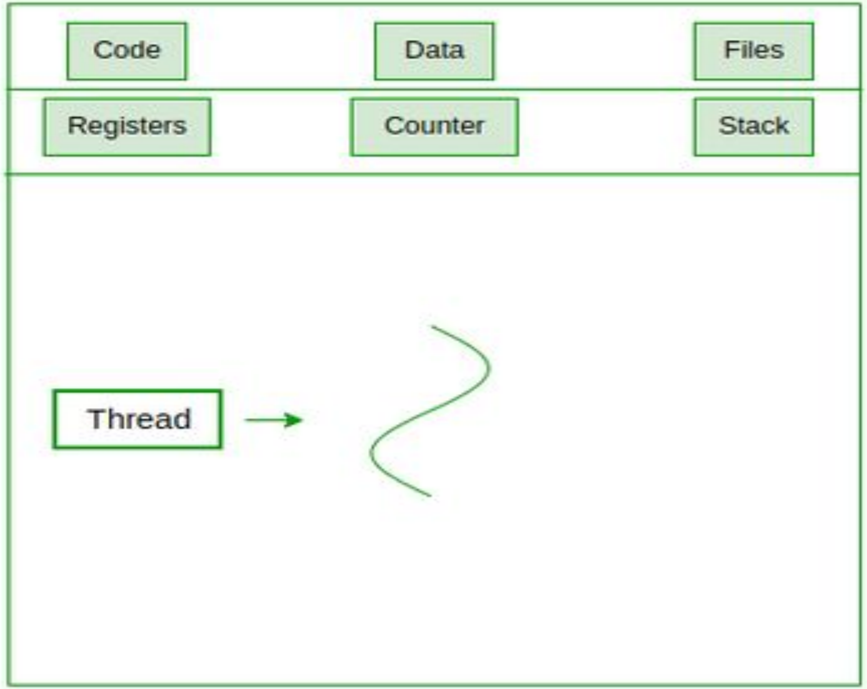
1. The kernel thread manages and schedules all threads.
2. The implementation of kernel threads is difficult than the user thread.
3. The kernel-level thread is slower than user-level threads.


User Level Thread
User Space
Kernel Space
Kernel level Thread
P
Process

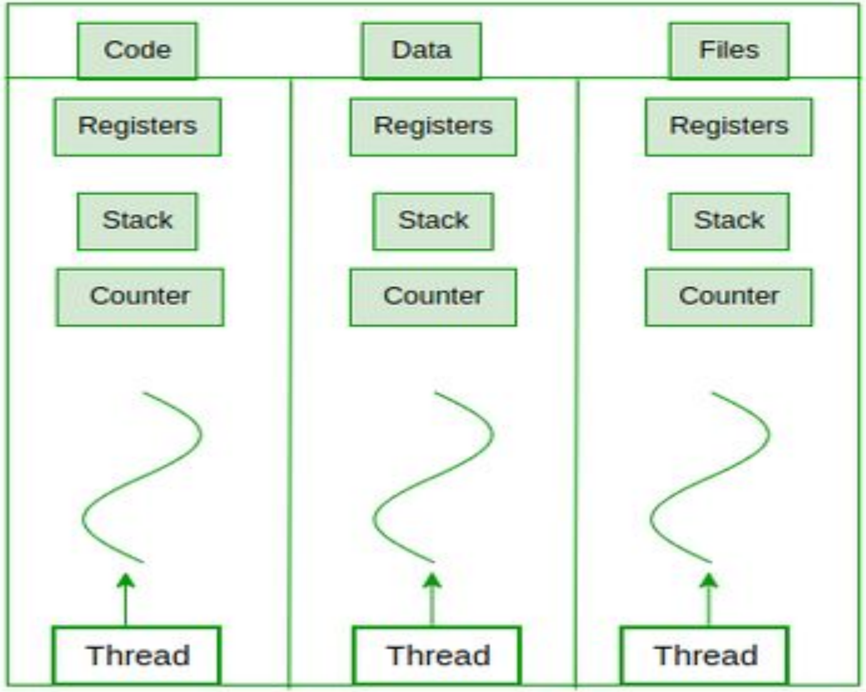
Benefits of Threads

- **Enhanced throughput of the system:** When the process is split into many threads, and each thread is treated as a job, the number of jobs done in the unit time increases. That is why the throughput of the system also increases.
- **Effective Utilization of Multiprocessor system:** When you have more than one thread in one process, you can schedule more than one thread in more than one processor.
- **Faster context switch:** The context switching period between threads is less than the process context switching. The process context switch means more overhead for the CPU.
- **Responsiveness:** When the process is split into several threads, and when a thread completes its execution, that process can be responded to as soon as possible.
- **Communication:** Multiple-thread communication is simple because the threads share the same address space, while in process, we adopt just a few exclusive communication strategies for communication between two processes.
- **Resource sharing:** Resources can be shared between all threads within a process, such as code, data, and files.

Note: The stack and register cannot be shared between threads. There is a stack and register for each thread.

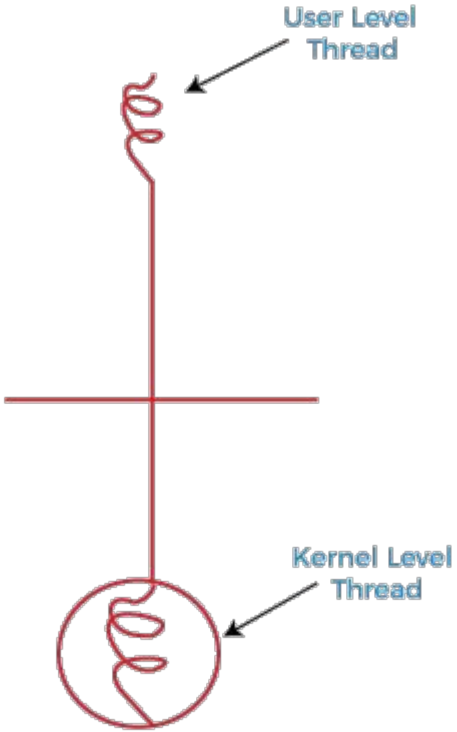


Single Threaded Process

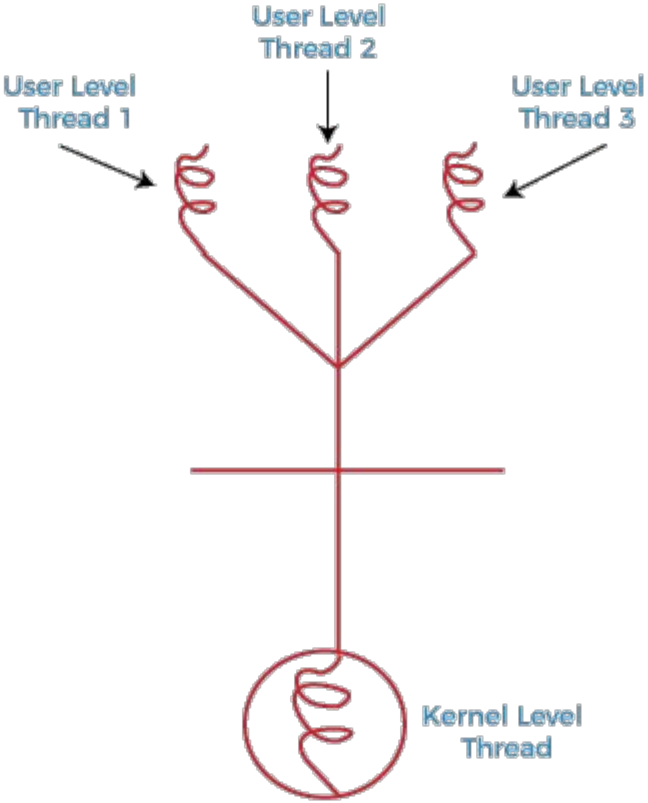


Multi Threaded Process

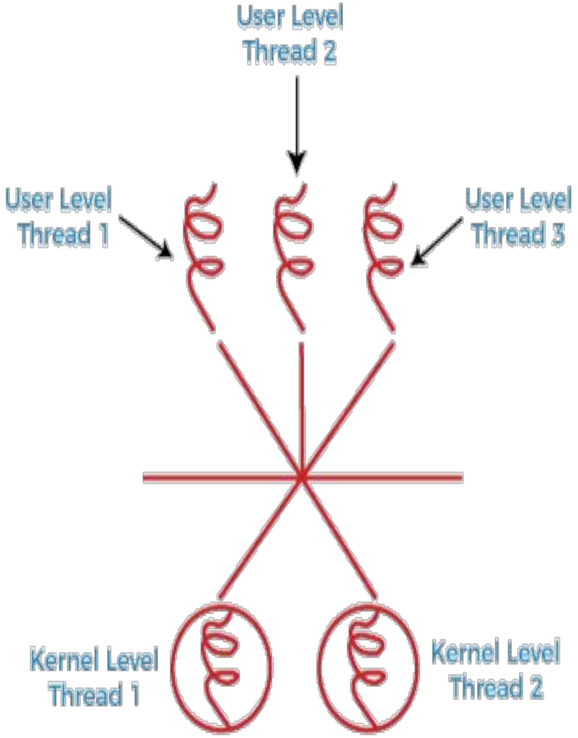
Multithreading Model



One-to-one model



Many-to-one model



Many-to-Many model

Many to one multithreading model:

The many to one model maps many user levels threads to one kernel thread. This type of relationship facilitates an effective context-switching environment, easily implemented even on the simple kernel with no thread support.

The disadvantage of this model is that since there is only one kernel-level thread schedule at any given time, this model cannot take advantage of the hardware acceleration offered by multithreaded processes or multi-processor systems. In this, all the thread management is done in the userspace. If blocking comes, this model blocks the whole system.

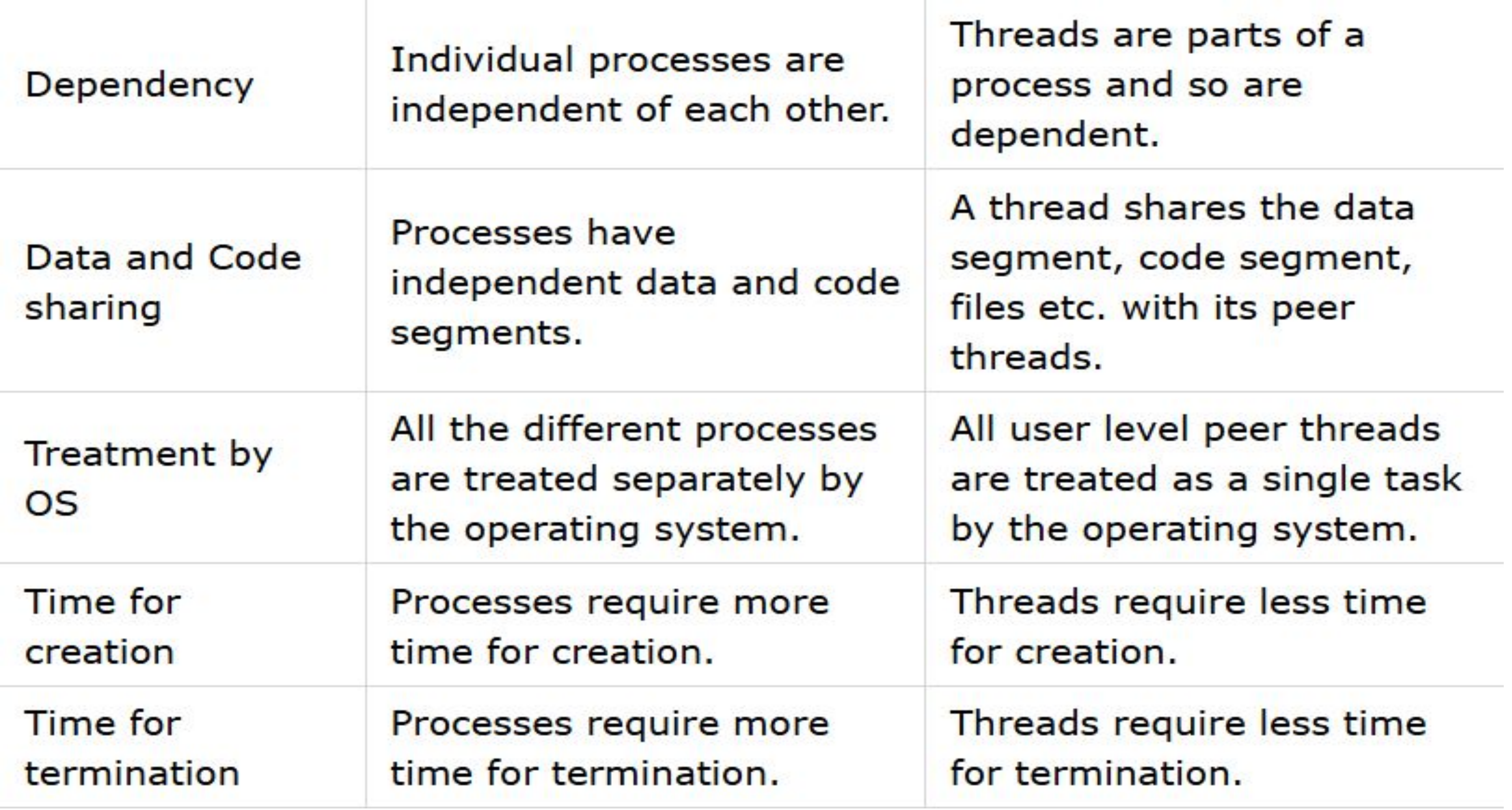
One to one multithreading model

The one-to-one model maps a single user-level thread to a single kernel-level thread. This type of relationship facilitates the running of multiple threads in parallel. However, this benefit comes with its drawback. The generation of every new user thread must include creating a corresponding kernel thread causing an overhead, which can hinder the performance of the parent process. Windows series and Linux operating systems try to tackle this problem by limiting the growth of the thread count.

Many to Many Model multithreading model

In this type of model, there are several user-level threads and several kernel-level threads. The number of kernel threads created depends upon a particular application. The developer can create as many threads at both levels but may not be the same. The many to many model is a compromise between the other two models. In this model, if any thread makes a blocking system call, the kernel can schedule another thread for execution. Also, with the introduction of multiple threads, complexity is not present as in the previous models. Though this model allows the creation of multiple kernel threads, true concurrency cannot be achieved by this model. This is because the kernel can schedule only one process at a time.

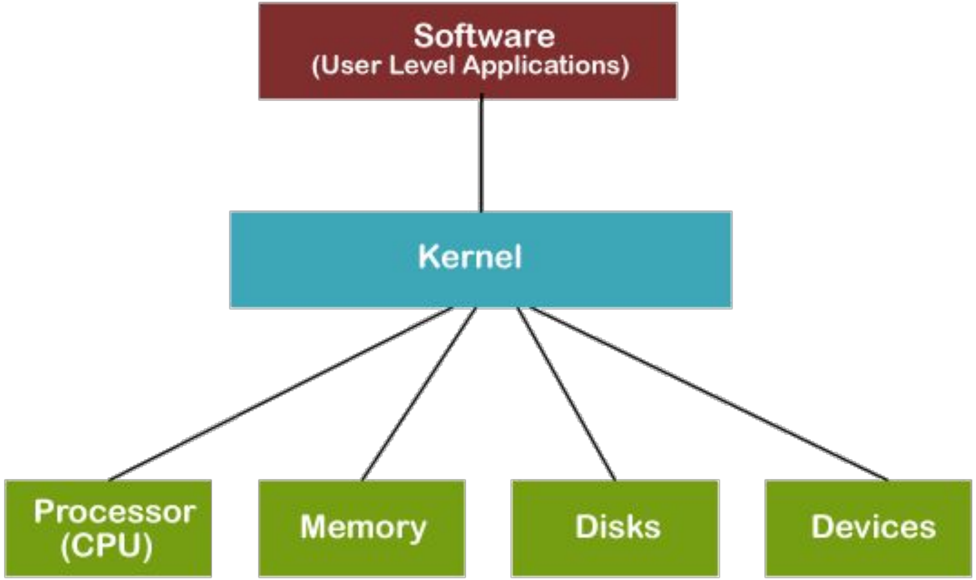
Comparison Basis	Process	Thread
Definition	A process is a program under execution i.e. an active program.	A thread is a lightweight process that can be managed independently by a scheduler
Context switching time	Processes require more time for context switching as they are heavier.	Threads require less time for context switching as they are lighter than processes.
Memory Sharing	Processes are totally independent and don't share memory.	A thread may share some memory with its peer threads.
Communication	Communication between processes requires more time than between threads.	Communication between threads requires less time than between processes.
Blocked	If a process gets blocked, remaining processes can continue execution.	If a user level thread gets blocked, all of its peer threads also get blocked.
Resource Consumption	Processes require more resources than threads.	Threads generally need less resources than processes.

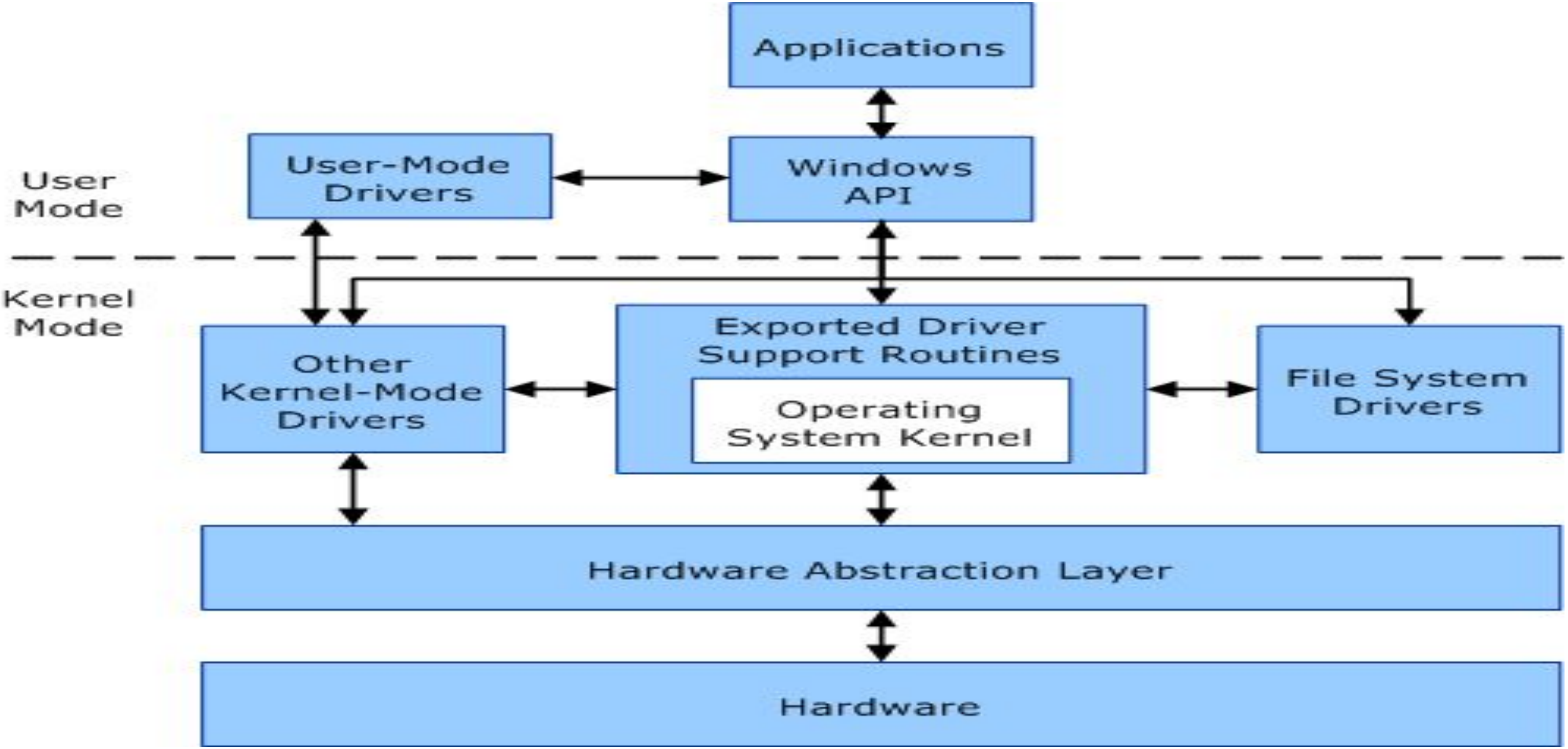


Dependency	Individual processes are independent of each other.	Threads are parts of a process and so are dependent.
Data and Code sharing	Processes have independent data and code segments.	A thread shares the data segment, code segment, files etc. with its peer threads.
Treatment by OS	All the different processes are treated separately by the operating system.	All user level peer threads are treated as a single task by the operating system.
Time for creation	Processes require more time for creation.	Threads require less time for creation.
Time for termination	Processes require more time for termination.	Threads require less time for termination.

Unit 4: Kernel

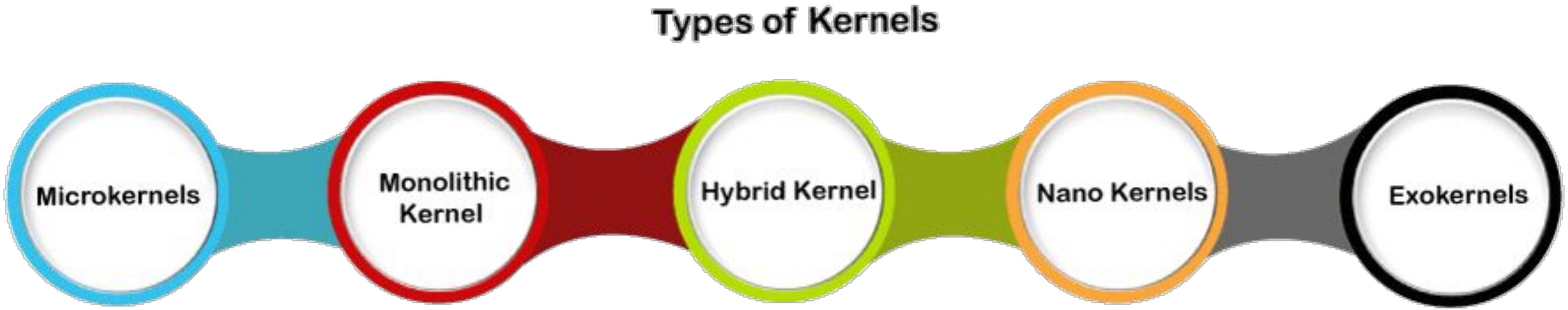
A kernel is the core part of an operating system. It acts as a bridge between software applications and the hardware of a computer. The kernel manages system resources, such as the CPU, memory, and devices, ensuring everything works together smoothly and efficiently. It handles tasks like running programs, accessing files, and connecting to devices like printers and keyboards.



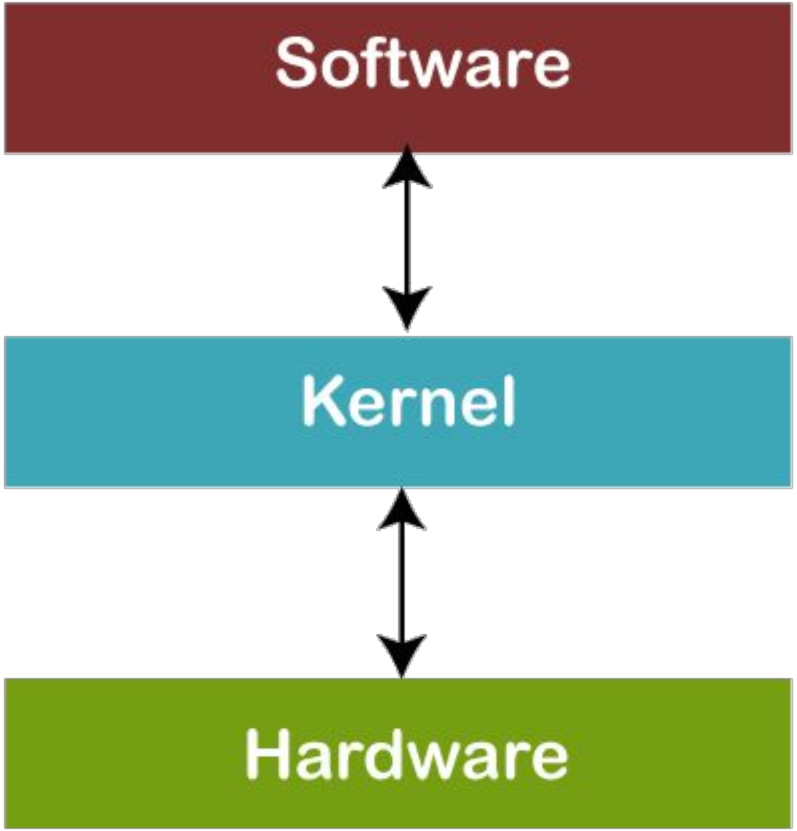

Applications
User Mode
User-Mode Drivers
Windows API
Kernel Mode
Other Kernel-Mode Drivers
Exported Driver Support Routines
Operating System Kernel
File System Drivers
Hardware Abstraction Layer
Hardware

- It is the computer program that first loaded on start-up the system (After the bootloader). Once it is loaded, it manages the remaining start-ups. It also manages memory, peripheral, and I/O requests from software. Moreover, it translates all I/O requests into data processing instructions for the CPU. It manages other tasks also such as memory management, task management, and disk management.
- A kernel is kept and usually loaded into separate memory space, known as protected Kernel space. It is protected from being accessed by application programs or less important parts of OS.
- Other application programs such as browser, word processor, audio & video player use separate memory space known as user-space.
- Due to these two separate spaces, user data and kernel data don't interfere with each other and do not cause any instability and slowness.

Types of Kernel



1. Monolithic Kernels



In a monolithic kernel, the same memory space is used to implement user services and kernel services.

It means, in this type of kernel, there is no different memory used for user services and kernel services.

As it uses the same memory space, the size of the kernel increases, increasing the overall size of the OS.

The execution of processes is also faster than other kernel types as it does not use separate user and kernel space.

Examples of Monolithic Kernels are Unix, Linux, Open VMS, XTS-400, etc.

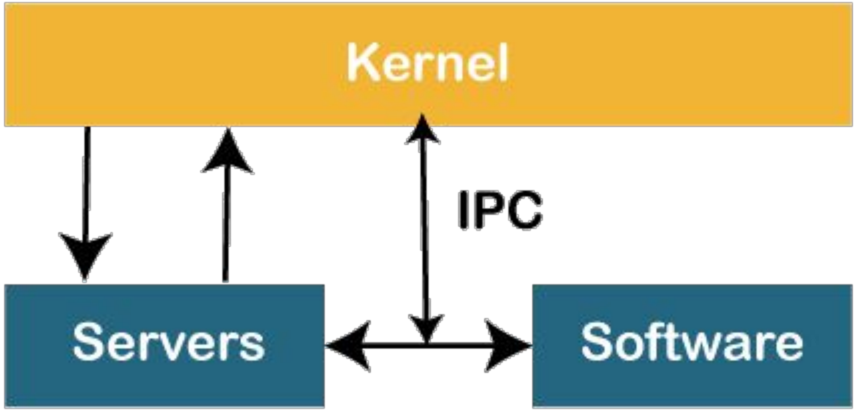
Advantages:

- The execution of processes is also faster as there is no separate user space and kernel space and less software involved.
- As it is a single piece of software hence, it's both sources and compiled forms are smaller.

Disadvantages:

- If any service generates any error, it may crash down the whole system.
- These kernels are not portable, which means for each new architecture, they must be rewritten.
- Large in size and hence become difficult to manage.
- To add a new service, the complete operating system needs to be modified.

2. Microkernel



In this, *user services and kernel services are implemented into two different address spaces: user space and kernel space*. Since it uses different spaces for both the services, so, the size of the microkernel is decreased, and which also reduces the size of the OS.

These kernels use a message passing system for handling the request from one server to another server.

Examples of Microkernel are L4, AmigaOS, Minix, K42, etc.

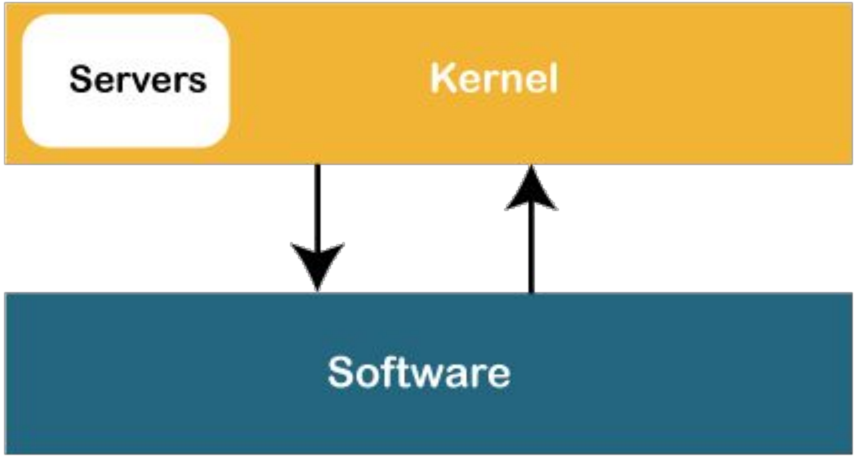
Advantages

- Microkernels can be managed easily.
- A new service can be easily added without modifying the whole OS.
- In a microkernel, if a kernel process crashes, it is still possible to prevent the whole system from crashing.

Disadvantages

- There is more requirement of software for interfacing, which reduces the system performance.
- Process management is very complicated.
- The messaging bugs are difficult to fix.

3. Hybrid Kernel



A hybrid kernel can be understood as the extended version of a microkernel with additional properties of a monolithic kernel. These kernels are widely used in commercial OS, such as different versions of MS Windows.

It is much similar to a microkernel, but it also includes some additional code in kernel space to enhance the performance of the system.

Hybrid kernels allow to run some services such as network stack in kernel space to reduce the performance compared to a traditional microkernel, but it still allows to run kernel code (such as device drivers) as servers in user-space.

Examples of Hybrid Kernel are Windows NT, Netware, BeOS, etc.

Advantages:

- There is no requirement for a reboot for testing.
- Third-party technology can be integrated rapidly.

Disadvantages:

- There is a possibility of more bugs with more interfaces to pass through.
- It can be a confusing task to maintain the modules for some administrators, especially when dealing with issues such as symbol differences.

4. Nanokernel

As the name suggests, *in Nanokernel, the complete code of the kernel is very small, which means the code executing in the privileged mode of the hardware is very small*. Here the term nano defines a kernel that supports a nanosecond clock resolution.

Examples of Nanokernel are EROS etc.

Advantages

- It provides hardware abstractions even with a very small size.

Disadvantages

- Nanokernel lacks system services.

5. Exokernel

Exokernel is still developing and is the experimental approach for designing OS.

This type of kernel is different from other kernels as in this; resource protection is kept separated from management, which allows us to perform application-specific customization.

Advantages:

○ The exokernel-based system can incorporate multiple library operating systems. Each library exports a different API, such as one can be used for high-level UI development, and the other can be used for real-time control.

Disadvantages:

○ The design of the exokernel is very complex.

Functions of Kernel

1. Process Management

- Scheduling and execution of processes.
- Context switching between processes.
- Process creation and termination.

2. Memory Management

- Allocation and deallocation of memory space.
- Managing virtual memory.
- Handling memory protection and sharing.

3. Device Management

- Managing input/output devices.
- Providing a unified interface for hardware devices.
- Handling device driver communication.

4. File System Management

- Managing file operations and storage.
- Handling file system mounting and unmounting.
- Providing a file system interface to applications.

5. Resource Management

- Managing system resources (CPU time, disk space, network bandwidth)
- Allocating and deallocating resources as needed
- Monitoring resource usage and enforcing resource limits

6. Security and Access Control

- Enforcing access control policies.
- Managing user permissions and authentication.
- Ensuring system security and integrity.

7. Inter-Process Communication

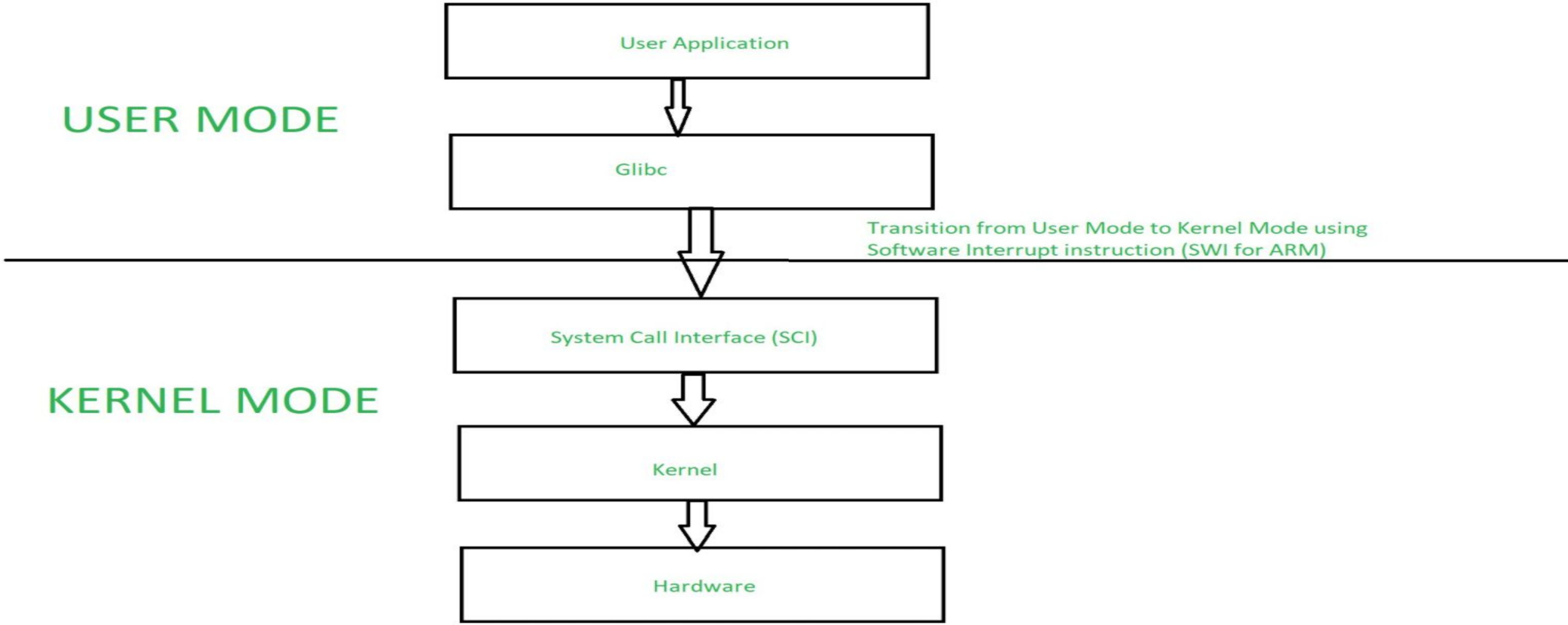
- Facilitating communication between processes.
- Providing mechanisms like message passing and shared memory.

Context Switching(Kernel mode and User mode)

- The **User mode** is normal mode where the process has limited access. While the **Kernel mode** is the privileged mode where the process has unrestricted access to system resources like hardware, memory, etc.
- The kernel provides System Call Interface **(SCI)**, which are the entry points for kernel. System Calls are the only way through which a process can go into kernel mode from user mode.
- When a user-level application needs to perform an operation that requires kernel mode access, such as accessing hardware devices or modifying system settings, it must make a system call to the operating system kernel.

The operating system switches the processor from user mode to kernel mode to execute the system call, and then switches back to user mode once the operation is complete.

This switching between user mode and kernel mode is known as mode switching or context switching.



User Application
USER MODE
Glibc
Transition from User Mode to Kernel Mode using
Software Interrupt instruction (SWI for ARM)
System Call Interface (SCI)
KERNEL MODE
Kernel
Hardware

Conclusion : For any system, privilege mode and non-privilege mode is important for access protection. The processor must have hardware support for user/kernel mode. System Call Interfaces (SCI) are the only way to transit from User space to kernel space. Kernel space switching is achieved by Software Interrupt, which changes the processor mode and jump the CPU execution into interrupt handler, which executes corresponding System Call routine.

Objectives of Kernel

- To establish communication between user-level applications and hardware.
- To decide the state of incoming processes.
- To control disk management.
- To control memory management.
- To control task management.

Kernel implementation processes include:

- **Creating kernel processes**: Kernel processes are created using the creatp and initp kernel services.
- **Representing processes**: The kernel uses a process descriptor to represent each process and store information about its current state.
- **Managing processes**: The kernel manages processes by saving the contents of processor registers in the process descriptor when it stops a process's execution.
- **Accessing data**: Kernel processes use kernel services to access and manipulate data from the u-area.
- **Allocating memory**: Kernel processes can allocate and de-allocate memory from the kernel heaps.
- **Using containers**: Containers are lightweight virtual machines that share the same kernel instance. They are built on top of kernel features like namespaces, which isolate different resources.
- **Using shared memory**: Shared memory is a way for processes to exchange and share data.
- **Using message queues**: Message queues allow processes to exchange messages using system calls.

Unit 6: File System

A computer file is defined as a medium used for saving and managing data in the computer system. The data stored in the computer system is completely in digital format, although there can be various types of files that help us to store the data.

provide a structured way to store, organize, and manage data on storage devices such as hard drives, SSDs, and USB drives.

A file system acts as a bridge between the operating system and the physical storage hardware, allowing users and applications to create, read, update, and delete files in an organized and efficient manner.

What is a File System?

A file system is a method an operating system uses to store, organize, and manage files and directories on a storage device. Some common types of file systems include:

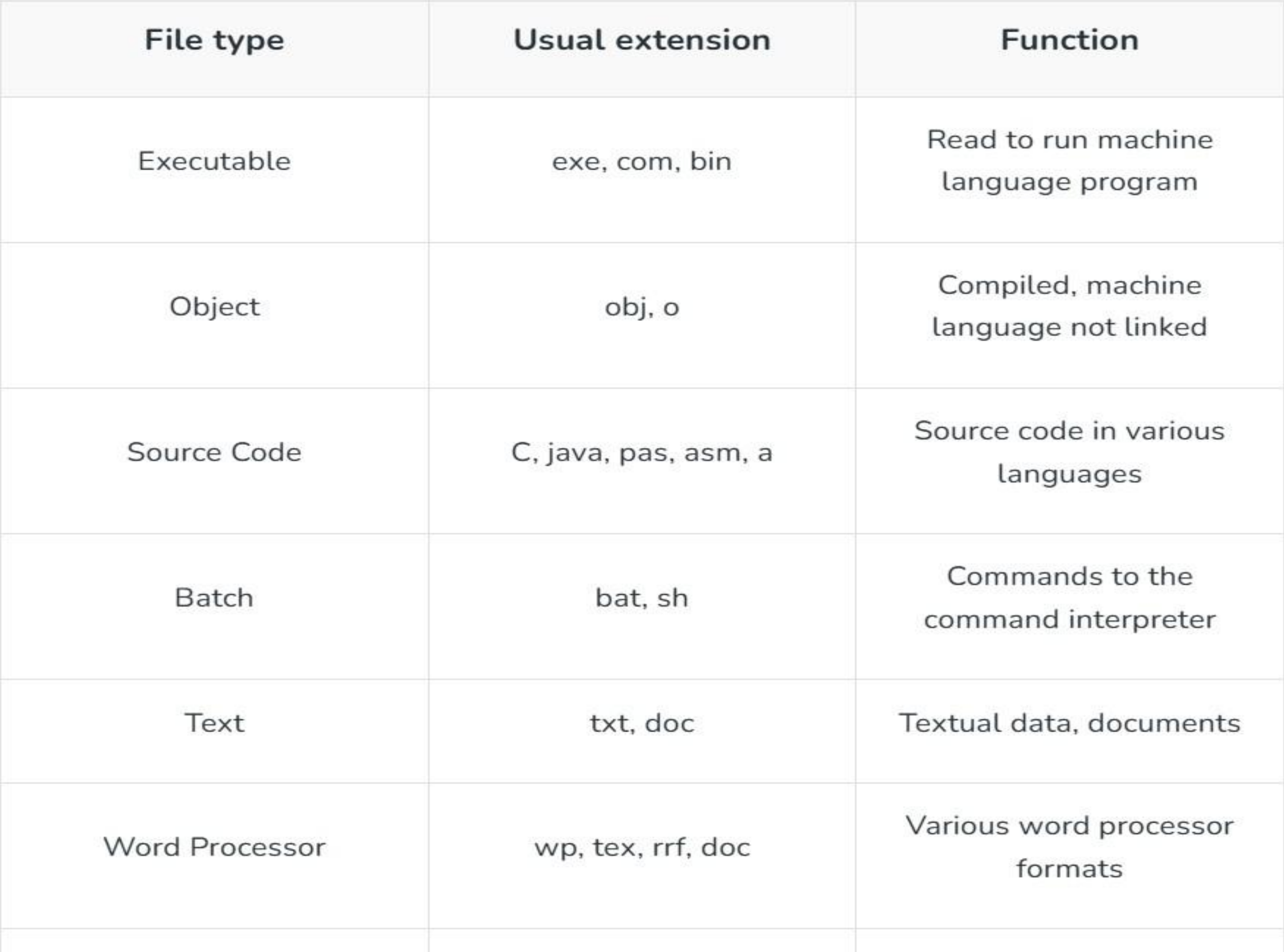
- **FAT (File Allocation Table):** An older file system used by older versions of Windows and other operating systems.
- **NTFS (New Technology File System):** A modern file system used by Windows. It supports features such as file and folder permissions, compression, and encryption.
- **ext (Extended File System):** A file system commonly used on Linux and Unix-based operating systems.
- **HFS (Hierarchical File System):** A file system used by macOS.
- **APFS (Apple File System):** A new file system introduced by Apple for their Macs and iOS devices

The name of the file is divided into two parts as shown below:.

- Name
- Extension, separated by a period.

Files Attributes And Their Operations

Attributes	Types	Operations
Name	Doc	Create
Type	Exe	Open
Size	Jpg	Read
Creation Data	Xis	Write
Author	C	Append
Last Modified	Java	Truncate
protection	class	Delete



File type	Usual extension	Function
Executable	exe, com, bin	Read to run machine language program
Object	obj, o	Compiled, machine language not linked
Source Code	C, java, pas, asm, a	Source code in various languages
Batch	bat, sh	Commands to the command interpreter
Text	txt, doc	Textual data, documents
Word Processor	wp, tex, rrf, doc	Various word processor formats

Archive	arc, zip, tar	Related files grouped into one compressed file
Multimedia	mpeg, mov, rm	For containing audio/video information
Markup	xml, html, tex	It is the textual data and documents
Library	lib, a ,so, dll	It contains libraries of routines for programmers
Print or View	gif, pdf, jpg	It is a format for printing or viewing an ASCII or binary file.

File Blocking allow you to identify specific file types that you want to want to block or monitor. For most block files that are known to carry threats or that have no real use case for upload/download. Currently, these include batch files, DLLs, Java class files, help files, Windows shortcuts (.lnk), and BitTorrent files.

In Operating Systems I/O operations are one of the most fundamental tasks that is needed to be carried out correctly and with the utmost efficiency. One of the techniques that we can use to ensure the utmost efficiency of the I/O Operations is Buffering. So, *Buffering* is a process in which the data is stored in a buffer or cache, which makes this stored data more accessible than the original source. Buffer is an area in memory that is used to hold the data that is being transmitted from one place to another and store the data temporarily.

What is a file descriptor?

A file descriptor is a unique identifier or reference that the operating system assigns to a file when it is opened. It allows programs to interact with files, sockets, or other input/output (I/O) resources. The file descriptor is used by the operating system to keep track of the file and perform operations on it.

A file structure is the way files are organized on a computer. It's important because it helps you find the files you need. Adopting a file structure makes getting to the files you need as easy as possible. Specifically, it's to minimize the number of trips to the disk to get the data you're looking for.

File Structures: Pile, Sequential, Indexed Sequential, Direct access, Inverted files; Indexing structures- B-tree and its variations.

File operations within an operating system (OS) encompass a set of essential tasks and actions directed at files and directories residing within a computer's file system. These operations are fundamental for the effective management and manipulation of data stored on various storage devices.

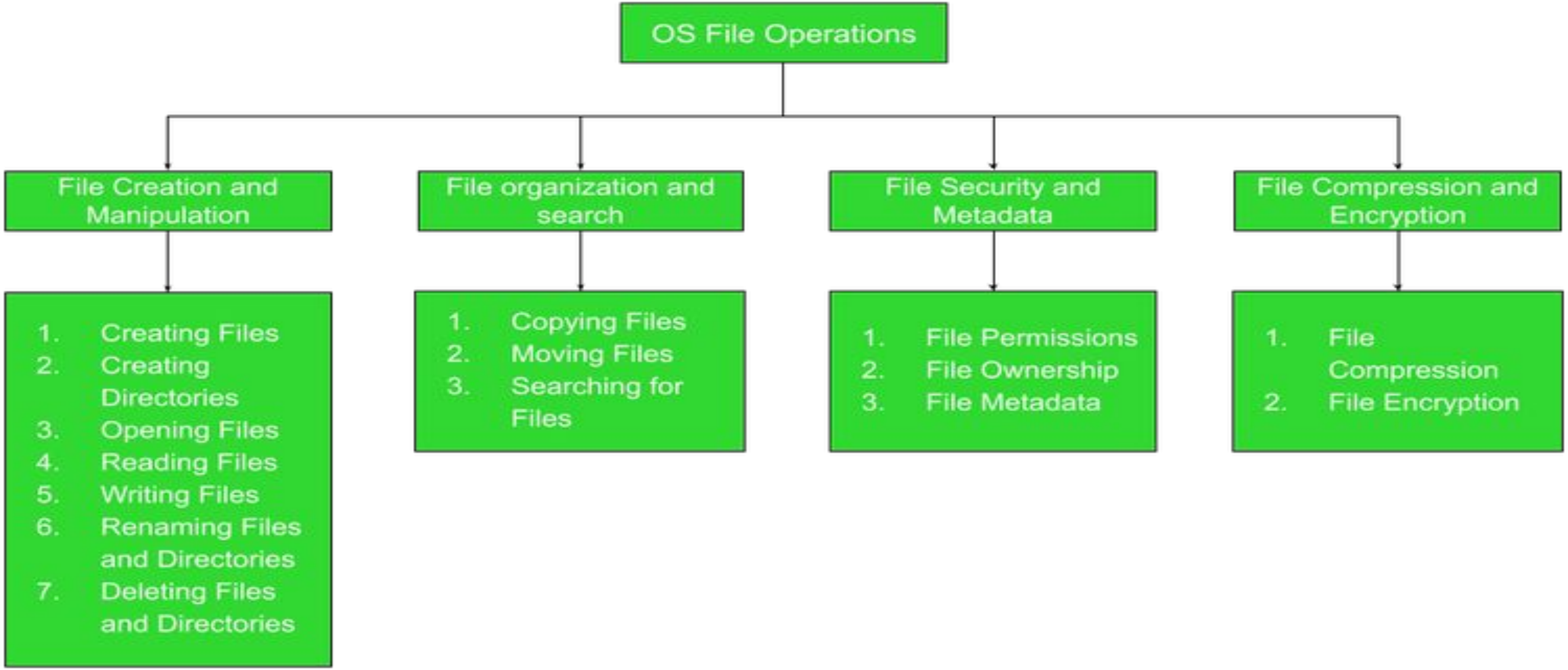


Fig: 1.1 OS File Operations

File Access Methods

File access methods in an operating system are the techniques and processes used to read from and write to files stored on a computer's storage devices.

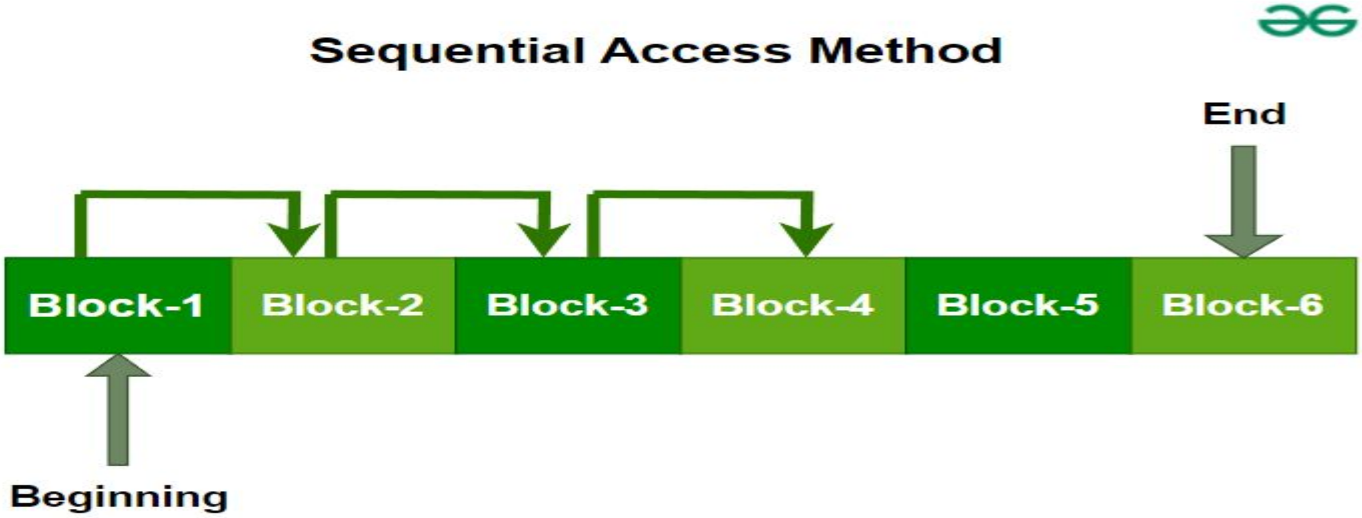
There are three ways to access a file in a computer system:

- Sequential-Access
- Direct Access
- Index sequential Method

Sequential Access

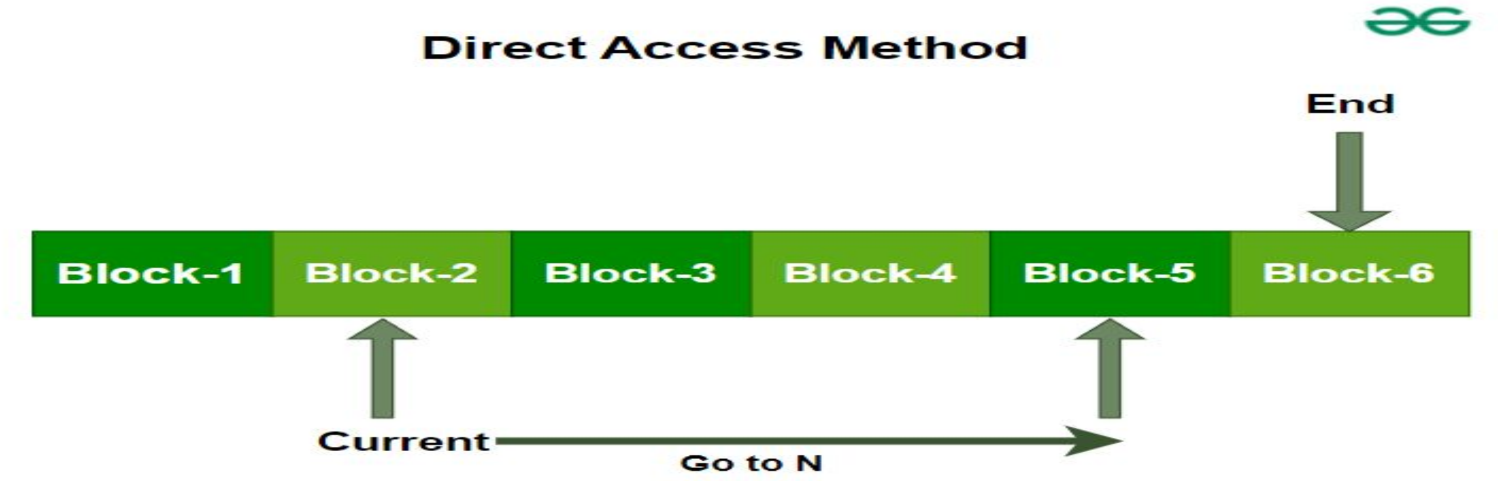
It is the simplest access method. Information in the file is processed in order, one record after the other.

Read and write make up the bulk of the operation on a file. A read operation *-read next-* reads the next position of the file and automatically advances a file pointer, which keeps track of the I/O location. Similarly, for the -write *next*- append to the end of the file and advance to the newly written material.



Direct Access Method

Another method is *direct access method* also known as *relative access method*. A fixed-length logical record that allows the program to read and write record rapidly. in no particular order. The direct access is based on the disk model of a file since disk allows random access to any file block. For direct access, the file is viewed as a numbered sequence of block or record. Thus, we may read block 14 then block 59, and then we can write block 17. There is no restriction on the order of reading and writing for a direct access file. A block number provided by the user to the operating system is normally a *relative block number*, the first relative block of the file is 0 and then 1 and so on.



Index Sequential method

It is the other method of accessing a file that is built on the top of the sequential access method. These methods construct an index for the file. The index, like an index in the back of a book, contains the pointer to the various blocks. To find a record in the file, we first search the index, and then by the help of pointer we access the file directly.

Key Points Related to Index Sequential Method

- It is built on top of Sequential access.
- It control the pointer by using index.

Other Way of File Access

1. Relative Record Access

Relative record access is a file access method used in operating systems where records are accessed relative to the current position of the file pointer. In this method, records are located based on their position relative to the current record, rather than by a specific address or key value.

2. Content Addressable Access

Content-addressable access (CAA) is a file access method used in operating systems that allows records or blocks to be accessed based on their content rather than their address. In this method, a hash function is used to calculate a unique key for each record or block, and the system can access any record or block by specifying its key.

What is Access Control?

Access Control is a method of limiting access to a system or resources. Access control refers to the process of determining who has access to what resources within a network and under what conditions. It is a fundamental concept in security that reduces risk to the business or organization. Access control systems perform identification, authentication, and authorization of users and entities by evaluating required login credentials that may include passwords, pins, bio-metric scans, or other authentication factors.

Components of Access Control

- **Authentication:** Authentication is the process of verifying the identity of a user. User authentication is the process of verifying the identity of a user when that user logs in to a computer system.
- **Authorization:** Authorization determines the extent of access to the network and what type of services and resources are accessible by the authenticated user. Authorization is the method of enforcing policies.
- **Access:** After the successful authentication and authorization, their identity becomes verified, This allows them to access the resource to which they are attempting to log in.
- **Manage:** Organizations can manage their access control system by adding and removing authentication and authorization for users and systems. Managing these systems can be difficult in modern IT setups that combine cloud services and physical systems.
- **Audit:** The access control audit method enables organizations to follow the principle. This allows them to collect data about user activities and analyze it to identify possible access violations

Types of Access Control

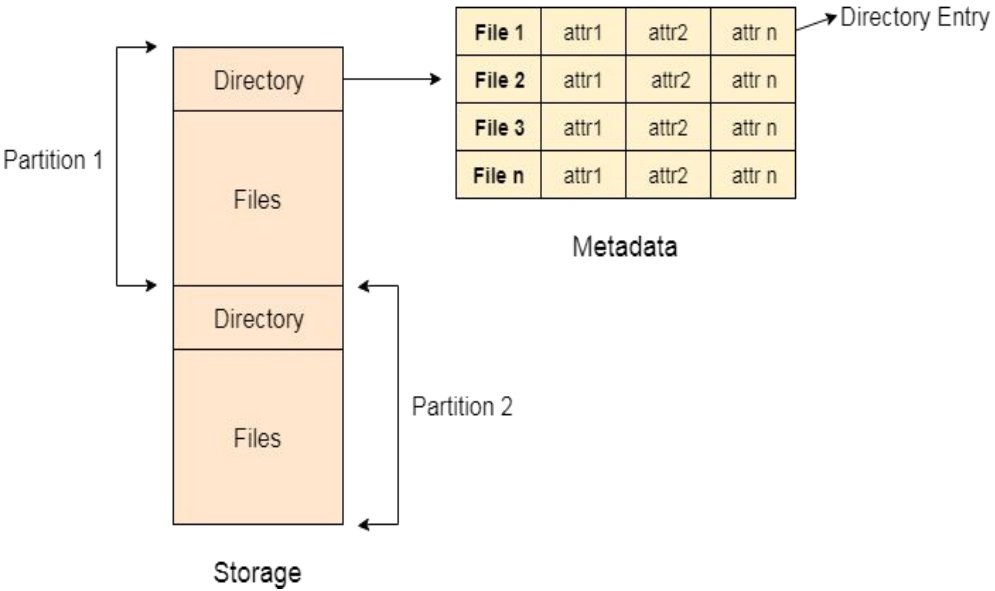
- **Attribute-based Access Control (ABAC):** In this model, access is granted or declined by evaluating a set of rules, policies, and relationships using the attributes of users, systems and environmental conditions.
- **Discretionary Access Control (DAC):** In DAC, the owner of data determines who can access specific resources.
- **History-Based Access Control (HBAC):** Access is granted or declined by evaluating the history of activities of the inquiring party that includes behavior, the time between requests and content of requests.
- **Identity-Based Access Control (IBAC):** By using this model network administrators can more effectively manage activity and access based on individual requirements.

- **Organization-Based Access control (OrBAC):** This model allows the policy designer to define a security policy independently of the implementation.
- **Role-Based Access Control (RBAC):** RBAC allows access based on the job title. RBAC eliminates discretion on a large scale when providing access to objects. For example, there should not be permissions for human resources specialist to create network accounts.
- **Rule-Based Access Control (RAC):** RAC method is largely context based. Example of this would be only allowing students to use the labs during a certain time of day.
- **Mandatory Access Control (MAC):** A control model in which access rights are regulated by a central authority based on multiple levels of security. Security Enhanced Linux is implemented using MAC on the Linux operating system.

Directory

A **directory** is a container that is used to contain folders and files. It organizes files and folders in a hierarchical manner. In other words, directories are like folders that help organize files on a computer.

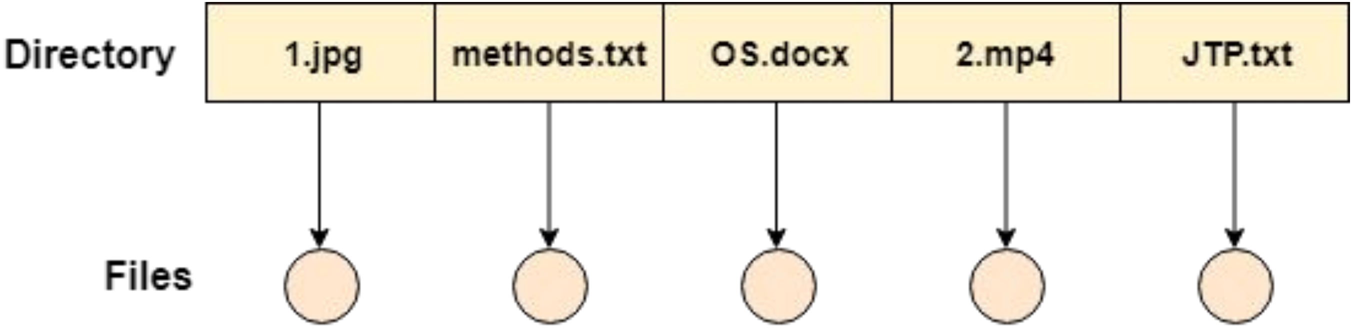
the operating system uses directories to keep track of files and where they are stored. Different structures of directories can be used to organize these files, making it easier to find and manage them.



File 1	attr1	attr2	attr n
File 2	attr1	attr2	attr n
File 3	attr1	attr2	attr n
File n	attr1	attr2	attr n

1) Single-Level Directory

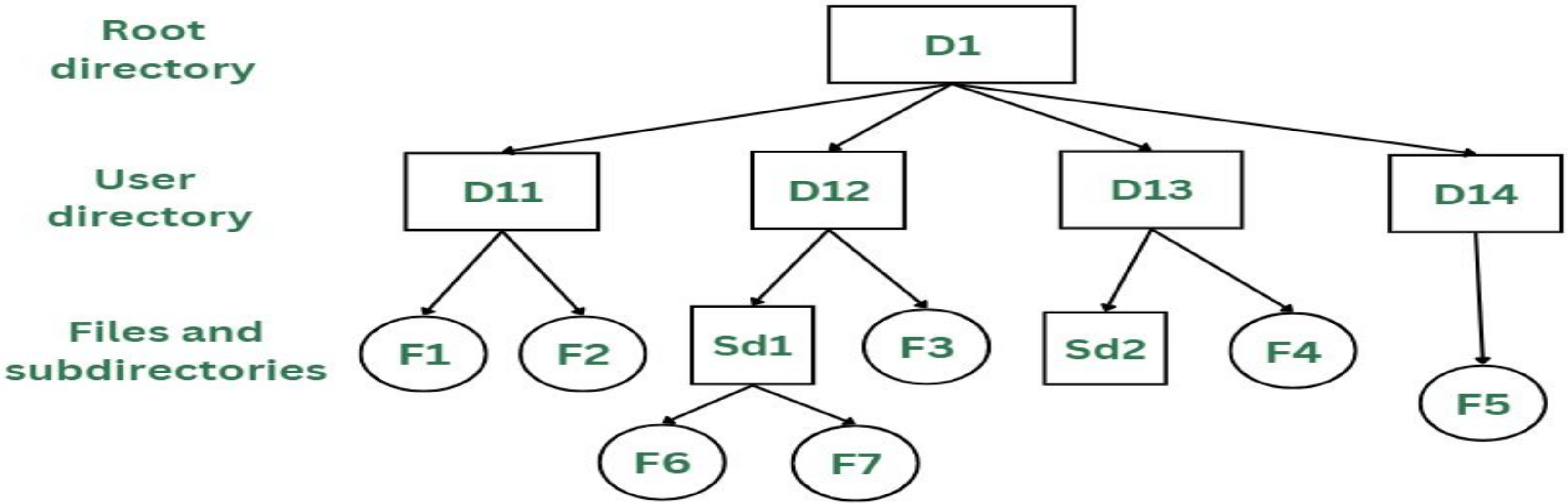
A single level directory is a file system where all files are stored in one directory, with no subdirectories. It's the simplest directory structure, but it has limitations when the number of files increases or when there are multiple users.



Single Level Directory

2) Tree Structure/ Hierarchical Structure

Tree directory structure of operating system is most commonly used in our personal computers. User can create files and subdirectories too, which was a disadvantage in the previous directory structures.



Every Directory supports a number of common operations on the file:

1. File Creation
2. Search for the file
3. File deletion
4. Renaming the file
5. Traversing Files
6. Listing of files

File Access Methods in Operating System

File access methods in an operating system are the techniques and processes used to read from and write to files stored on a computer’s storage devices. These methods determine how data is organized, retrieved, and modified within a file system.

File Access Methods

There are three ways to access a file in a computer system:

- Sequential-Access
- Direct Access
- Index sequential Method

Access control list (ACL)

An access control list (ACL) is a list of rules that specifies which users or systems are granted or denied access to a particular object or system resource. Access control lists are also installed in routers or switches, where they act as filters, managing which traffic can access the network.

Each system resource has a security attribute that identifies its access control list. The list includes an entry for every user who can access the system. The most common privileges for a file system ACL include the ability to read a file or all the files in a directory, to write to the file or files, and to execute the file if it is an executable file or program.

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

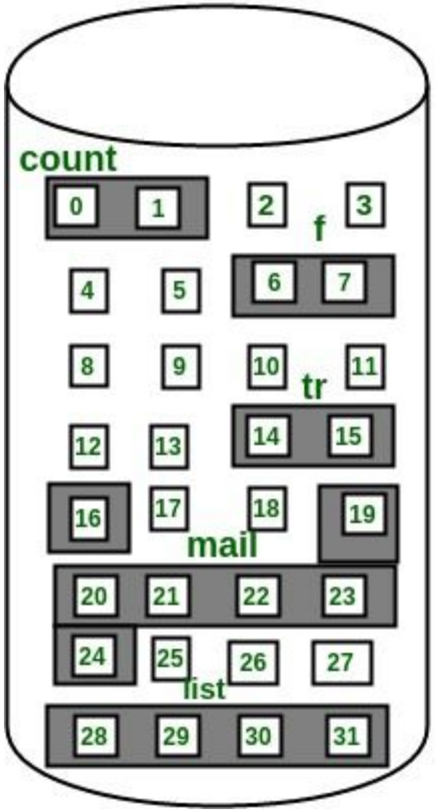
The main idea behind these methods is to provide:

- Efficient disk space utilization.
- Fast access to the file blocks.

1. Contiguous Allocation

In this scheme, each file occupies a contiguous set of blocks on the disk. For 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*. 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.



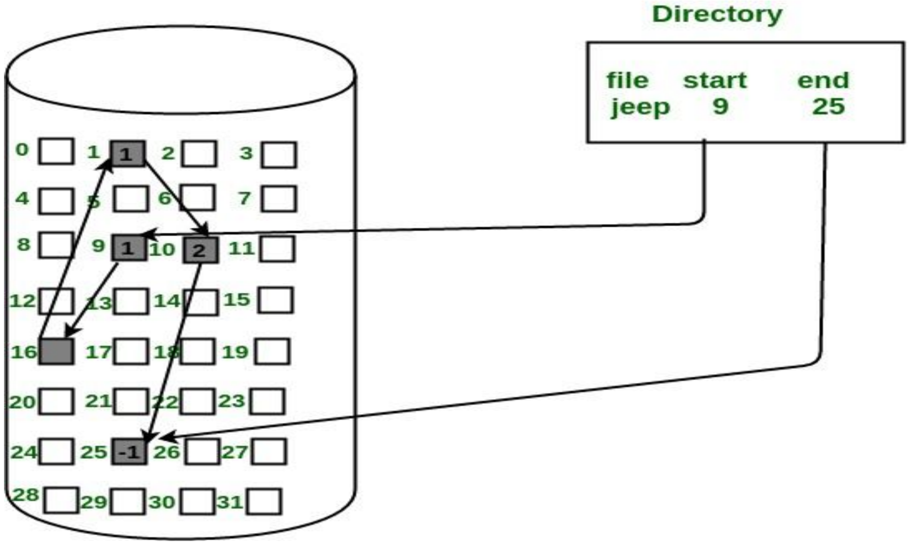
Directory

file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

2. Linked List Allocation

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

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:

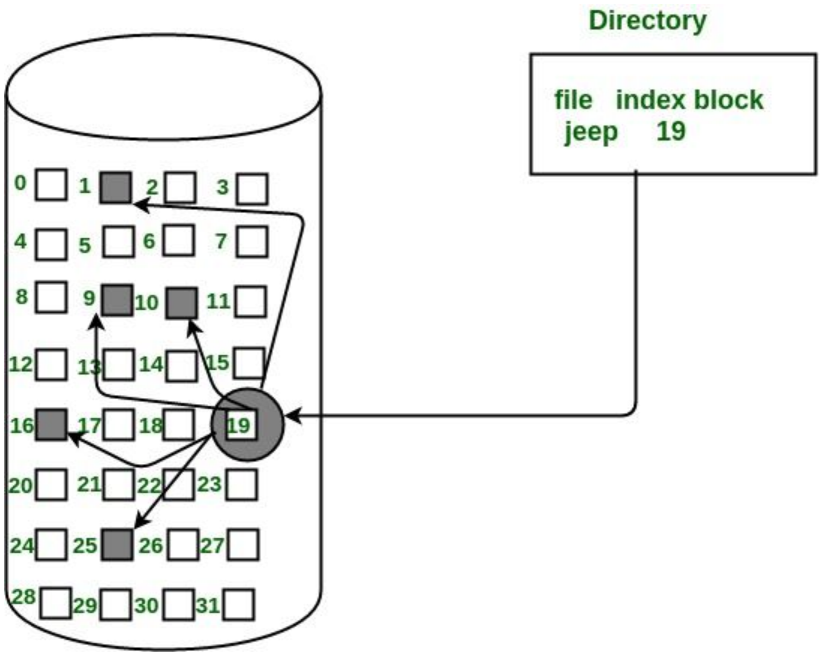
- This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for a contiguous chunk of memory.
- This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.

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.

3. Indexed Allocation

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

Unit 5: Input/Output Management

Humans interact with machines by providing information through I/O devices. In addition, much of whatever a computer system provides as on-line services is essentially made available through specialized devices such as screen displays, printers, keyboards, mouse, etc. Management of all these devices can affect the throughput of a system. Input/ Output management is one of the primary responsibilities of an Operating System.

Categories of Input/output devices

There are three categories of input/ output devices namely:-

1. Human readable: Devices used to communicate with the user. They include: Printers and terminals, Video display, Keyboard, Mouse etc.

2. Machine readable: devices used to communicate with electronic equipment. They include: Disk drives, USB keys, Sensors etc.

3. Communications: devices used to communicate with remote devices such as Modems.

Goals/ Objectives of Input /Output Management

1. Allow users to access all devices in a uniform manner.
2. Ensure that devices are named in a uniform manner.
3. Ensure that the OS, without the intervention of the user program, handles recoverable system errors.
4. Maintain security of the devices.
5. Optimize the performance of the I/O system.

Techniques for performing I/O

1. **Programmed mode:** In this mode of communication, an IO instruction is issued to an IO device and the program executes in “busy-waiting” (idling) mode till the IO is Completed. During the busy-wait period the processor is continually interrogating to check if the device has completed an IO activity.
2. **Polling mode:** In this mode of data transfer, a device is repeatedly checked for Readiness to communicate and if it is not the processor returns to a different task. If the device is ready, communication is initiated.
3. **Direct memory access (DMA) mode:** This is a mode of data transfer in which IO is performed in large data blocks. The direct memory access, or DMA ensures access to main memory without processor intervention or support. Such independence from processor makes this mode of transfer extremely efficient.

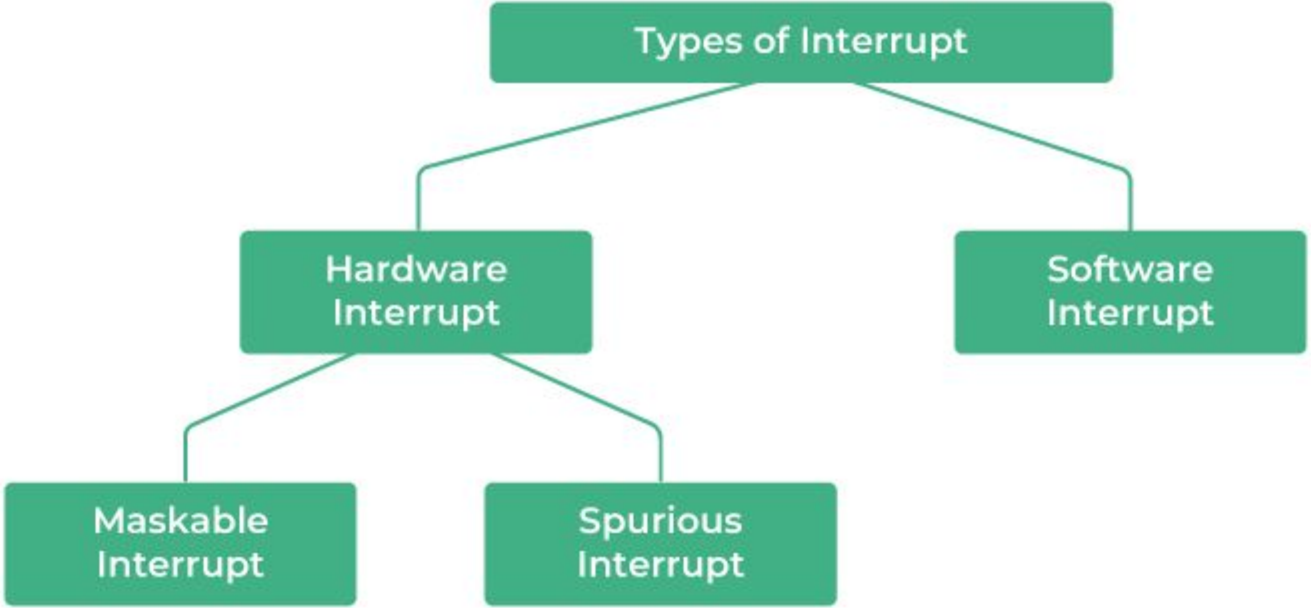
4. Interrupt mode: an interrupt is a signal from a device attached to a computer or from a program within the computer that causes the operating system to stop and figure out what to do next. An operating system usually has some code that is called an interrupt handler. The interrupt handler prioritizes the interrupts and saves them in a queue if more than one is waiting to be handled. The operating system has another little program, sometimes called a scheduler that figures out which program to give control to next.

Types of interrupts

1. Hardware interrupts: A hardware interrupt occurs, for example, when an I/O operation is completed such as reading some data into the computer from a tape drive.

2. Software interrupts: A software interrupt occurs when an application program terminates or requests certain services from the operating system.

Effective handling of interrupts calls for Context switch. Context Switch is the process of storing and restoring the execution context of a process or thread so that execution can be resumed from the same point at a later time. Context switch is carried out by the interrupt handler module of the operating system.



Types of Interrupt

Hardware Interrupt

Software Interrupt

Maskable Interrupt

Spurious Interrupt

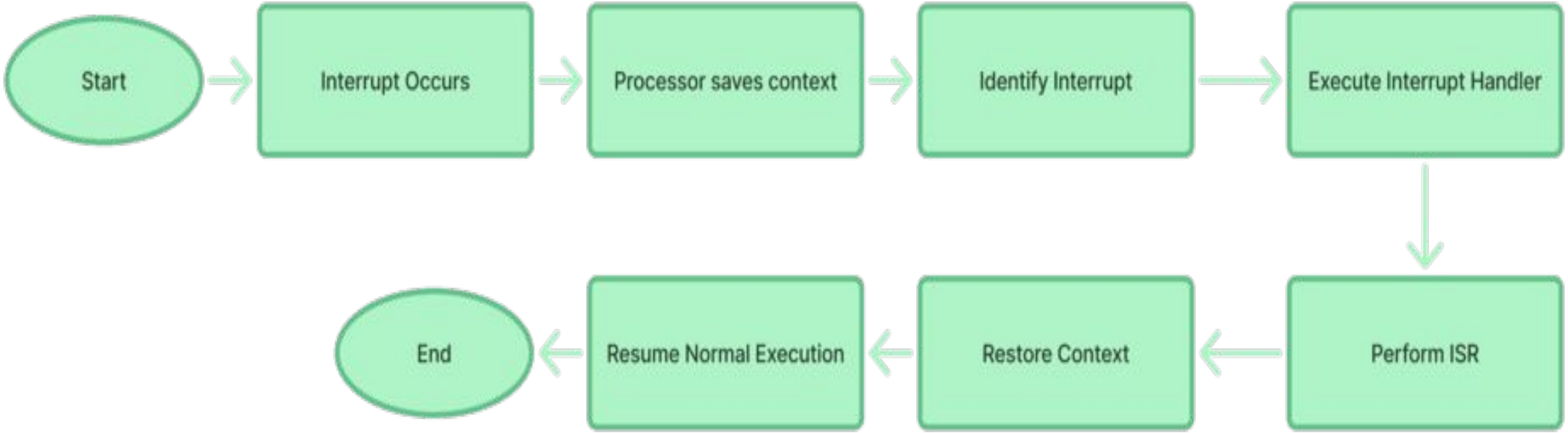
Hardware interrupts are further divided into two types of interrupt

- **Maskable Interrupt:** Hardware interrupts can be selectively enabled and disabled thanks to an inbuilt interrupt mask register that is commonly found in processors. A bit in the mask register corresponds to each interrupt signal; on some systems, the interrupt is enabled when the bit is set and disabled when the bit is clear, but on other systems, the interrupt is deactivated when the bit is set.
- **Spurious Interrupt:** A hardware interrupt for which there is no source is known as a spurious interrupt. This phenomenon might also be referred to as phantom or ghost interrupts. When a wired-OR interrupt circuit is connected to a level-sensitive processor input, spurious interruptions are typically an issue. When a system performs badly, it could be challenging to locate these interruptions.

Sequences of Events Involved in Handling an IRQ(Interrupt Request)

- Devices raise an IRQ.
- The processor interrupts the program currently being executed.
- The device is informed that its request has been recognized and the device deactivates the request signal.
- The requested action is performed.
- An interrupt is enabled and the interrupted program is resumed.

Interrupt Handling Mechanism



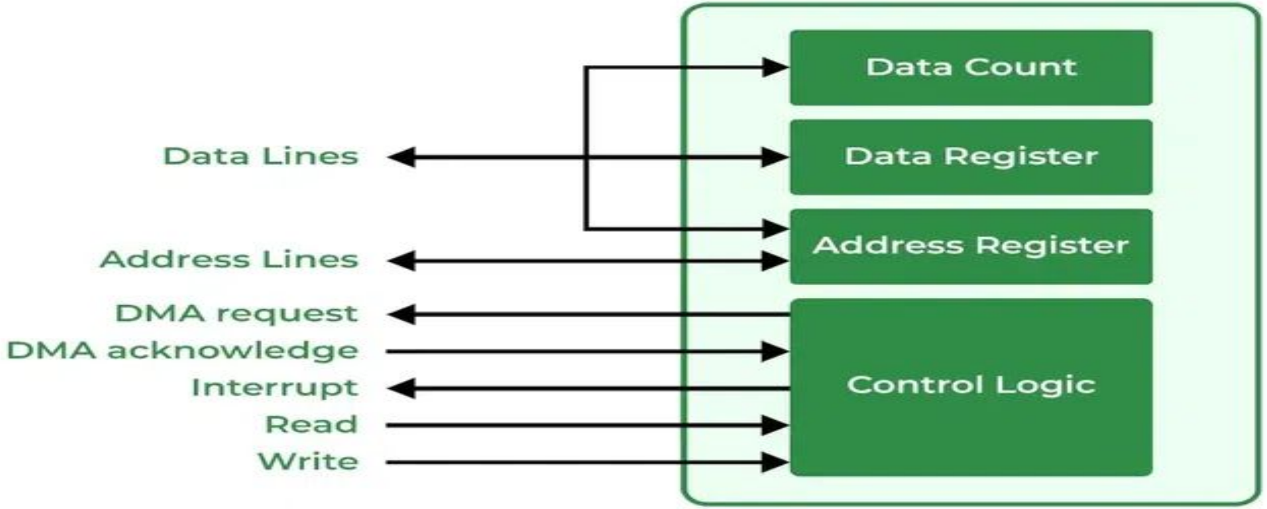
- Step 1:- Any time that an interrupt is raised, it may either be an I/O interrupt or a system interrupt.
- Step 2:- The current state comprising registers and the program counter is then stored in order to conserve the state of the process.
- Step 3:- The current interrupt and its handler is identified through the interrupt vector table in the processor.
- Step 4:- This control now shifts to the interrupt handler, which is a function located in the kernel space.
- Step 5:- Specific tasks are performed by Interrupt Service Routine (ISR) which are essential to manage interrupt.
- Step 6:- The status from the previous session is retrieved so as to build on the process from that point.
- Step 7:- The control is then shifted back to the other process that was pending and the normal process continues.

Direct Memory Access (DMA)

Transferring data between input/output devices and memory can be a slow process if the CPU is required to manage every step. To address this, a Direct Memory Access (DMA) Controller is utilized. A Direct Memory Access (DMA) Controller solves this by allowing I/O devices to transfer data directly to memory, reducing CPU involvement. This increases system efficiency and speeds up data transfers, freeing the CPU to focus on other tasks.

Direct Memory Access (DMA) uses hardware for accessing the memory, that hardware is called a DMA Controller. It has the work of transferring the data between Input Output devices and main memory with very less interaction with the processor. The direct Memory Access Controller is a control unit, which has the work of transferring data.

DMA Controller contains an address unit, which generates the address and selects an I/O device for the transfer of data. Here we are showing the block diagram of the DMA Controller.



Types of Direct Memory Access (DMA)

There are four popular types of DMA.

- Single-Ended DMA
- Dual-Ended DMA
- Arbitrated-Ended DMA
- Interleaved DMA

Single-Ended DMA: Single-Ended DMA Controllers operate by reading and writing from a single memory address. They are the simplest DMA.

Dual-Ended DMA: Dual-Ended DMA controllers can read and write from two memory addresses. Dual-ended DMA is more advanced than single-ended DMA.

Arbitrated-Ended DMA: Arbitrated-Ended DMA works by reading and writing to several memory addresses. It is more advanced than Dual-Ended DMA.

Interleaved DMA: Interleaved DMA are those DMA that read from one memory address and write from another memory address.

Advantages of DMA Controller

- Data Memory Access speeds up memory operations and data transfer.
- CPU is not involved while transferring data.
- DMA requires very few clock cycles while transferring data.
- DMA distributes workload very appropriately.
- DMA helps the CPU in decreasing its load.

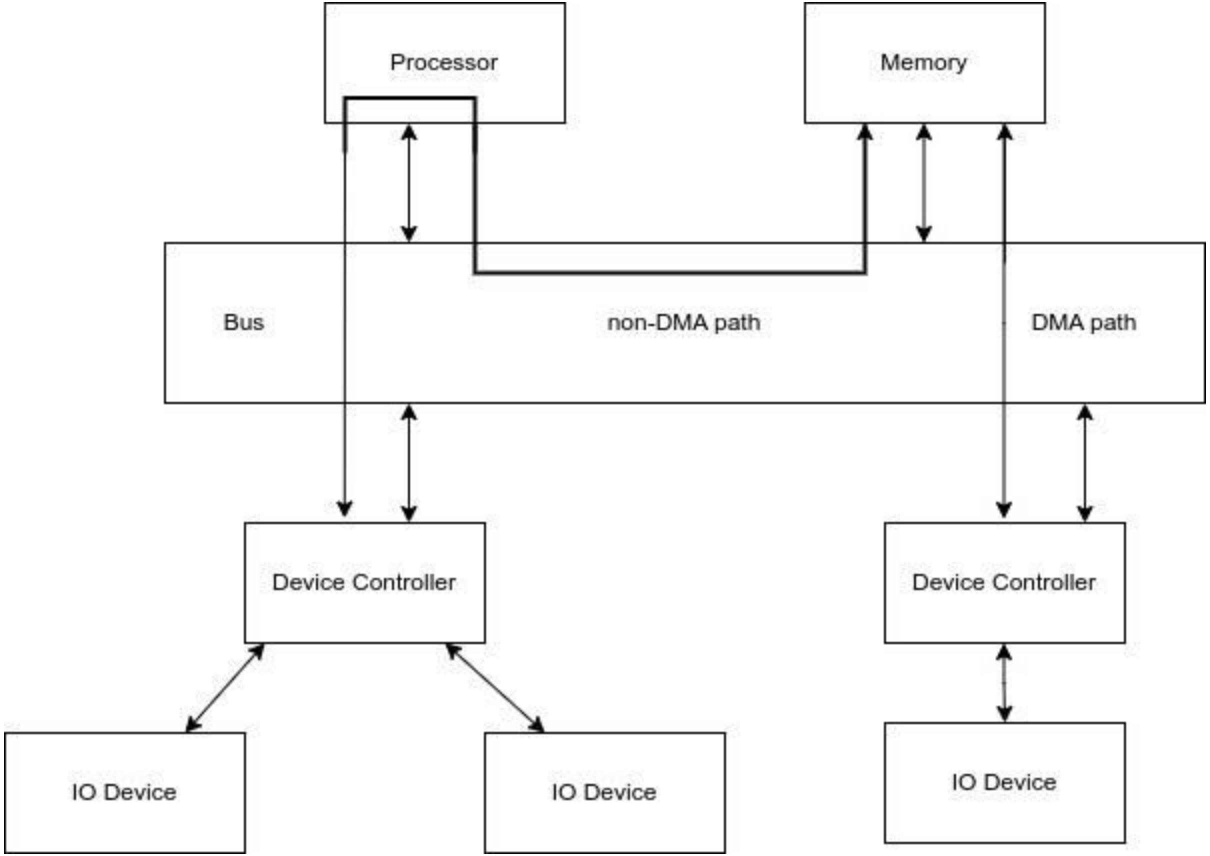
Disadvantages of DMA Controller

- Direct Memory Access is a costly operation because of additional operations.
- DMA Controller increases the overall cost of the system.
- DMA Controller increases the complexity of the software.

Device Controllers

In computer systems, I/O devices do not usually communicate with the operating system. The operating system manages their task with the help of one intermediate electronic device called a **device controller.**

The device controller knows how to communicate with the operating system as well as how to communicate with I/O devices. So device controller is an interface between the computer system (operating system) and I/O devices. The device controller communicates with the system using the system bus. So how the device controller, I/O devices, and the system bus is connected is shown below in the diagram


Processor
Memory
Bus
non-DMA path
DMA path
Device Controller
Device Controller
IO Device
IO Device
IO Device

I/O Software

I/O Software is used for interaction with I/O devices like mouse, keyboards, USB devices, printers, etc. Several commands are made via external available devices which makes the OS function upon each of them one by one.

I/O software is organized in the following ways:

User Level Libraries– Provides a simple interface to program for input-output functions.

Kernel Level Modules– Provides device driver to interact with the device-independent I/O modules and device controller.

Hardware-A layer including hardware controller and actual hardware which interact with device drivers.

Goals Of I/O Software

1. Uniform naming: Naming of file systems in Operating Systems is done in a way that the user does not have to be aware of the underlying hardware name.

2.. Device Independence: The most important part of I/O software is device independence. It is always preferable to write a program that can open all other I/O devices. For example, it is not necessary to write the input-taking program again and again for taking input from various files and devices. As this creates much work to do and also much space to store the different programs.

5. Error handling: Errors and mostly generated by the controller and also they are mostly handled by the controller itself. When the lower level solves the problem it does not reach the upper level.

6. Shareable and Non-Shareable Devices: Devices like Hard Disk can be shared among multiple processes while devices like Printers cannot be shared. The goal of I/O software is to handle both types of devices.

4. Buffering: Data that we enter into a system cannot be stored directly in memory. For example, the data is converted into smaller groups and then transferred to the outer buffer for examination. Buffer has a major impact on I/O software as it is the one that ultimately helps store the data and copy data. Many devices have constraints and just to avoid it some data is always put into the buffer in advance so the buffer rate of getting filled with data and getting empty remains balanced.

7. Caching: Caching is the process in which all the most accessible and most used data is kept in a separate memory (known as Cache memory) for access by creating a copy of the originally available data. The reason for implementing this Caching process is just to increase the speed of accessing the data since accessing the Cached copy of data is more efficient as compared to accessing the original data.

Polled I/O and **Interrupt-Driven I/O**

Polled I/O and **Interrupt-Driven I/O** are two methods used to manage input/output (I/O) operations in a computer system. Both methods allow communication between the CPU and peripheral devices, but they differ in how the CPU interacts with those devices.

1. Polled I/O (Polling)

In polled I/O, the CPU actively checks the status of an I/O device at regular intervals to determine if it is ready for data transfer. This means the CPU repeatedly queries the device to see if it needs attention, such as whether data is available to read or whether it is ready to accept data.

Key Features:

- **Continuous checking**: The CPU repeatedly checks the device for data availability.
- **Simplicity**: It's easy to implement because the CPU has full control of the timing and the polling loop is straightforward.
- **Inefficiency**: If the device doesn't require attention, the CPU wastes processing time checking repeatedly.

Advantages:

- Simpler to implement in basic systems.
- Predictable behavior since the CPU controls when and how often to check the device.

Disadvantages:

- Waste of CPU time when the device is idle.
- Less efficient in systems with many devices or where devices are infrequently ready to communicate.

2. Interrupt-Driven I/O

In interrupt-driven I/O, the peripheral device can signal the CPU when it is ready for communication, rather than the CPU having to continuously check the device's status. When the device is ready, it sends an interrupt to the CPU, which temporarily stops its current operations, handles the interrupt, and then resumes its previous task.

Key Features:

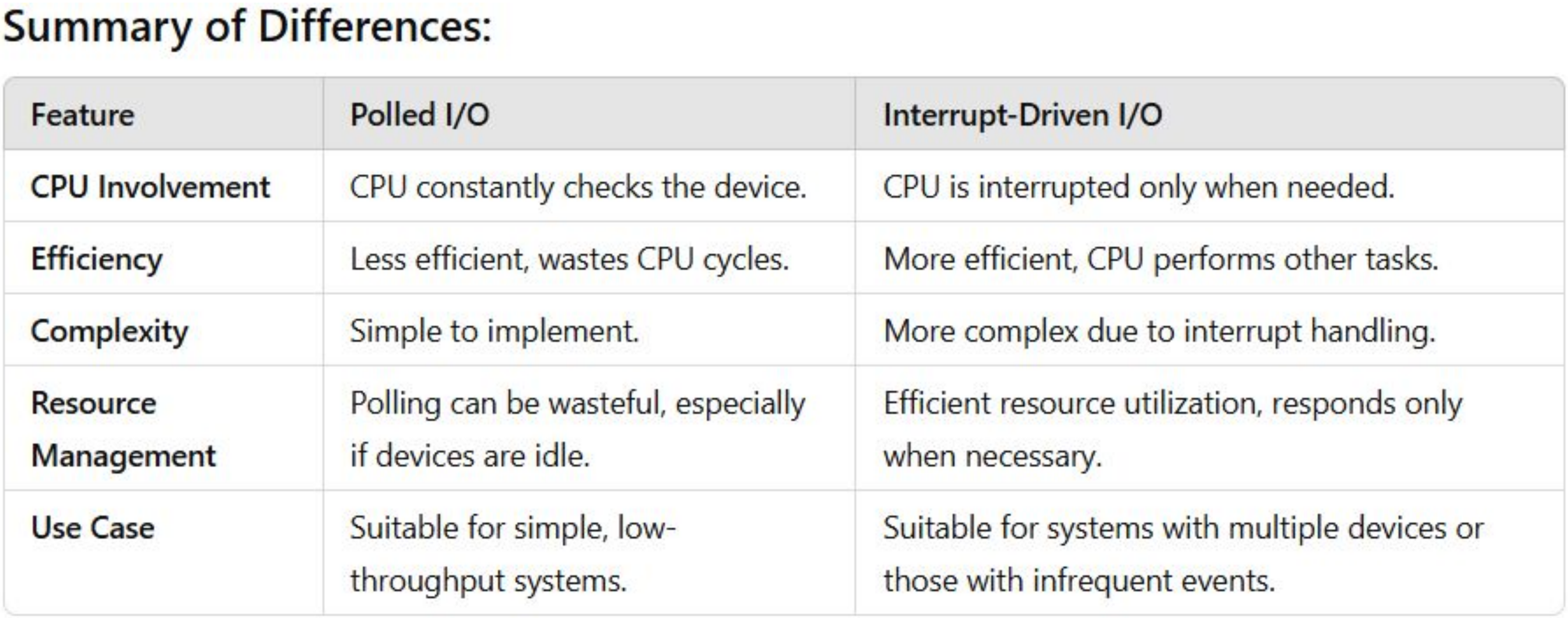
- **Interrupt signals**: The I/O device sends an interrupt to the CPU when it requires attention.
- **Efficiency**: The CPU can continue performing other tasks and only stops to handle an interrupt when necessary.
- **Complexity**: Requires an interrupt controller and interrupt service routines to manage interrupts, making it more complex to implement.

Advantages:

- Efficient CPU usage because the CPU doesn't waste time polling devices.
- Better for systems with multiple devices or devices that don't need constant attention.

Disadvantages:

- More complex to set up, as handling interrupts requires managing priorities, context switching, and possible interrupt collisions.
- Some overhead due to context switching when handling the interrupt.



Summary of Differences:

Feature	Polled I/O	Interrupt-Driven I/O
CPU Involvement	CPU constantly checks the device.	CPU is interrupted only when needed.
Efficiency	Less efficient, wastes CPU cycles.	More efficient, CPU performs other tasks.
Complexity	Simple to implement.	More complex due to interrupt handling.
Resource Management	Polling can be wasteful, especially if devices are idle.	Efficient resource utilization, responds only when necessary.
Use Case	Suitable for simple, low-throughput systems.	Suitable for systems with multiple devices or those with infrequent events.

CUI (Character User Interface) and **GUI (Graphical User Interface)** are two types of interfaces used to interact with computers or devices, but they differ in how users communicate with the system.

CUI (Character User Interface):

- **Definition:** CUI is a text-based interface that allows users to interact with the system through text commands.
- **Interaction:** Users type commands in a terminal or command prompt, often using a keyboard.
- **Appearance:** It uses text characters to represent options and outputs, with no graphical elements.
- **Example:** MS-DOS, Unix shell, Command Prompt in Windows.
- **Advantages:** Efficient for advanced users, uses minimal system resources.
- **Disadvantages:** Steeper learning curve, less intuitive for beginners, relies on memorization of commands.

GUI (Graphical User Interface):

- **Definition:** GUI is a visual interface that allows users to interact with the system using graphical elements such as icons, buttons, and windows.
- **Interaction:** Users interact by clicking, dragging, and selecting items using a pointing device (like a mouse or touchpad).
- **Appearance:** Displays graphical icons, buttons, images, and other visual elements.
- **Example:** Windows, macOS, Android, and iOS interfaces.
- **Advantages:** User-friendly, intuitive, and visually appealing, good for beginners and general users.
- **Disadvantages:** Can be resource-intensive, less efficient for power users compared to CUI.

In summary:

- **CUI** relies on text and command-line input.
- **GUI** uses visuals and interactions like pointing and clicking.

Device-independent I/O software

Device-independent I/O software refers to software that abstracts and manages input/output (I/O) operations in such a way that applications do not need to know the specifics of the underlying hardware. This abstraction allows applications to work with different types of devices (e.g., printers, keyboards, storage devices, display screens) without needing to be rewritten or adjusted for each particular device.

Here are key points about device-independent I/O software:

1. Abstraction Layer

- The software provides an abstraction layer that separates the application logic from the hardware. The application interacts with this layer instead of directly with hardware.
- The operating system (OS) typically manages this abstraction by using device drivers or a device-independent I/O system that communicates with various devices.

2. Device Drivers

- Device drivers are essential components in the I/O system that translate generic I/O requests from applications into device-specific commands. While drivers themselves are device-specific, they allow the application to remain unaware of the details of the hardware.

3. Unified Interface

- Device-independent I/O software typically provides a unified API (Application Programming Interface) that can work with different devices. For example, in a graphical user interface (GUI) system, the same code can handle different kinds of input devices, such as touchscreens, mice, or keyboards, through a single interface.

4. Portability

- By abstracting the hardware details, applications can be more portable. The same application can run on different machines with different hardware configurations without needing to be reprogrammed for each configuration.

Benefits:

- **Flexibility**: Allows software to run on a wide range of hardware.
- **Maintainability**: Device drivers can be updated or replaced without impacting the application code.
- **Ease of Use**: Developers can focus on application logic instead of worrying about device-specific details.

User-space I/O (Input/Output)

User-space I/O (Input/Output) software refers to the handling of I/O operations within the user space of an operating system, as opposed to the kernel space. In traditional operating systems, I/O operations are managed by the kernel, which controls the hardware devices and ensures secure and efficient access to system resources. However, user-space I/O software allows applications to manage their own I/O operations without directly involving the kernel in every operation. This can lead to better performance, reduced overhead, and more flexibility in certain use cases.

1. Preemptable Resources:

- **Definition**: These are system resources that can be interrupted or taken away from a running process by the operating system or the scheduler. When a resource is preemptable, the system can allocate it to a different process or thread, even if the current process is still using it.
- **Example**:
  - **CPU**: The CPU is often preemptable. A process using the CPU can be interrupted by the operating system's scheduler, allowing other processes to run in a multitasking environment.
  - **Memory (in certain systems)**: In some virtual memory systems, memory can be preempted by swapping out pages that are no longer needed.

2. Non-preemptable Resources:

- **Definition**: These are system resources that cannot be forcibly taken away from a running process. Once a process acquires such a resource, it retains it until the process voluntarily releases it.
- **Example**:
  - **I/O Devices**: When a process has exclusive access to an I/O device (e.g., disk, printer), the resource is typically non-preemptable. The process retains access to the device until its operation completes.
  - **Locks**: In concurrent programming, locks used to protect shared resources are often non-preemptable. Once a process has acquired a lock, other processes must wait until the lock is released.

Key Differences:

- **Preemptable** resources are more flexible and can be shared or allocated dynamically by the operating system, often to ensure fairness and optimize system performance.
- **Non-preemptable** resources require careful management, as once acquired, they block other processes from accessing them until released. These resources are often managed in ways that avoid starvation or deadlock.

Methods of handling deadlock

Deadlock occurs in computer systems, particularly in multi-threaded or multi-process environments, when two or more processes are blocked forever, waiting for each other to release resources.

1. Deadlock Prevention

Deadlock prevention aims to ensure that at least one of the necessary conditions for deadlock (mutual exclusion, hold and wait, no preemption, and circular wait) is denied.

- **Mutual Exclusion**: Ensure that resources are shared as much as possible, so that exclusive access is not required.
- **Hold and Wait**: Disallow processes from holding one resource while waiting for another. This can be prevented by requiring processes to request all resources at once.
- **No Preemption**: If a process is holding resources and waiting for others, preempt those resources and allocate them to processes that can use them immediately.

Banker's Algorithm

The **Banker's Algorithm** is a resource allocation and deadlock avoidance algorithm used in operating systems to allocate resources to processes while ensuring that a system never enters a deadlock state. It was designed by **Edsger Dijkstra**.

What is Memory Management?

Memory management mostly involves management of main memory. In a multiprogramming computer, the Operating System resides in a part of the main memory, and the rest is used by multiple processes. The task of subdividing the memory among different processes is called Memory Management. Memory management is a method in the operating system to manage operations between main memory and disk during process execution. The main aim of memory management is to achieve efficient utilization of memory.

Why Memory Management is Required?

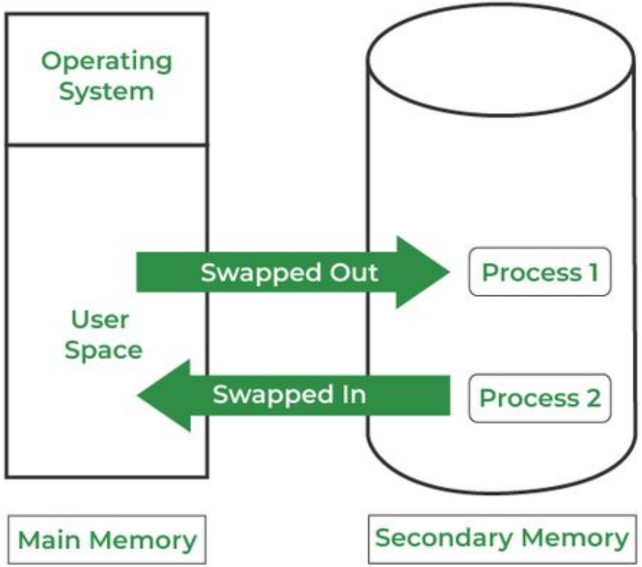
- Allocate and de-allocate memory before and after process execution.
- To keep track of used memory space by processes.
- To minimize fragmentation issues.
- To proper utilization of main memory.
- To maintain data integrity while executing of process.

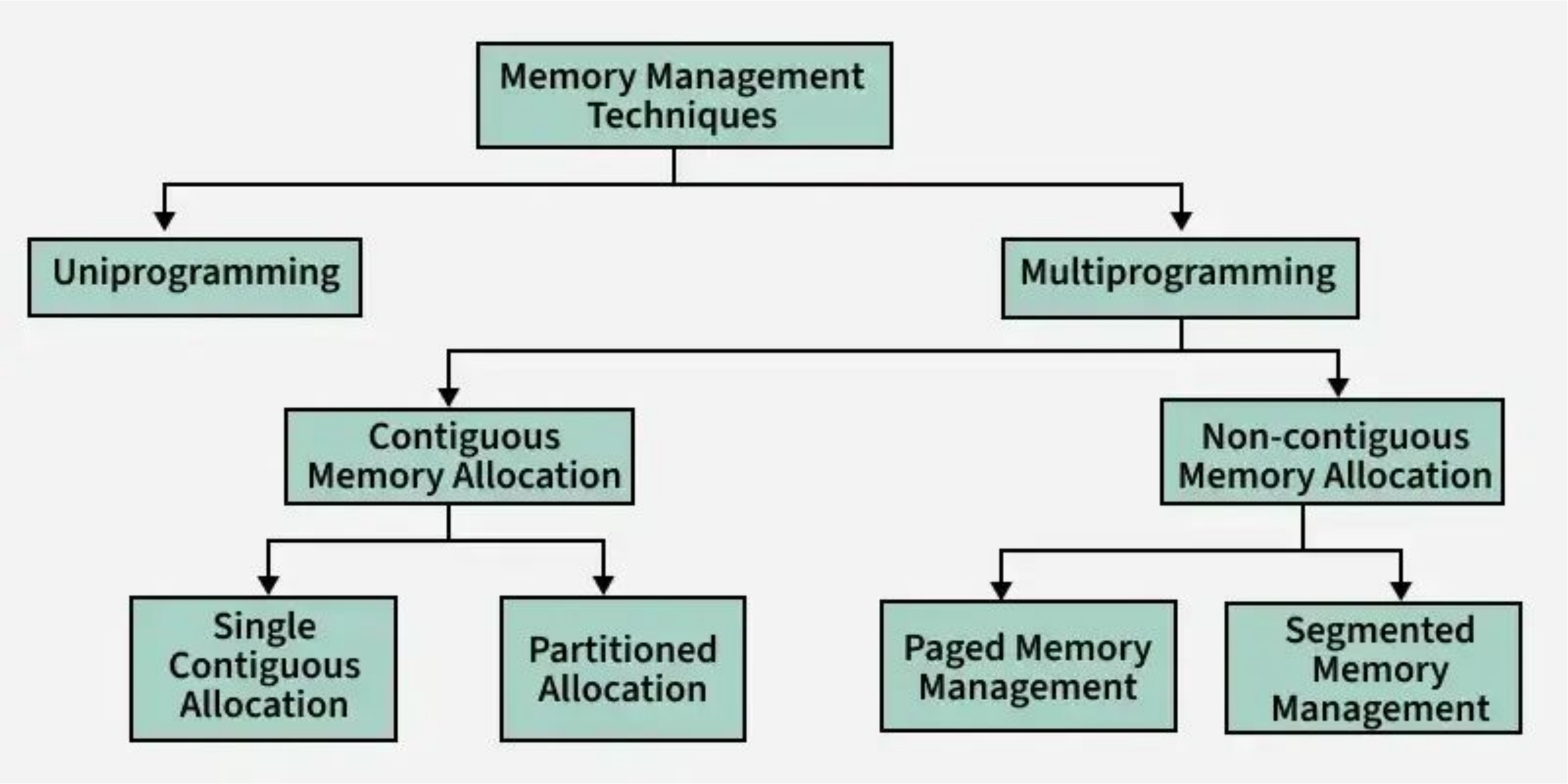
Logical and Physical Address Space

- **Logical Address Space:** An address generated by the CPU is known as a “Logical Address”. It is also known as a Virtual address. Logical address space can be defined as the size of the process. A logical address can be changed.
- **Physical Address Space:** An address seen by the memory unit (i.e. the one loaded into the memory address register of the memory) is commonly known as a “Physical Address”. A Physical address is also known as a Real address. The set of all physical addresses corresponding to these logical addresses is known as Physical address space. A physical address is computed by MMU. The run-time mapping from virtual to physical addresses is done by a hardware device Memory Management Unit(MMU). The physical address always remains constant.

Swapping

When a process is executed it must have resided in memory. Swapping is a process of swapping a process temporarily into a secondary memory from the main memory, which is fast compared to secondary memory. A swapping allows more processes to be run and can be fit into memory at one time. The main part of swapping is transferred time and the total time is directly proportional to the amount of memory swapped.




Memory Management Techniques
Uniprogramming
Multiprogramming
Contiguous Memory Allocation
Non-contiguous Memory Allocation
Single Contiguous Allocation
Partitioned Allocation
Paged Memory Management
Segmented Memory Management

memory management with bitmaps

Memory management using bitmaps is a technique where a bitmap (or bit vector) is used to represent the allocation status of memory blocks in a system. Each bit in the bitmap corresponds to a specific block or unit of memory, and the bit's value indicates whether that memory block is free or allocated.

A bitmap is a sequence of bits (0s and 1s), where each bit represents the status of a memory block. For example:

- `0` = block is free.
- `1` = block is allocated.

The memory is divided into fixed-size blocks, and each block is represented by a corresponding bit in the bitmap.

Allocation and Deallocation:

- When a block is allocated, the corresponding bit in the bitmap is set to `1`.
- When a block is freed, the corresponding bit is set to `0`.

Efficiency:

- Bitmap management is efficient in terms of space, but finding the next available block can require scanning the bitmap.
- Depending on the system design, this scanning can be optimized (e.g., maintaining a pointer to the next free bit).

Advantages:

- Simple and effective for managing memory.
- Easy to implement and understand.
- Constant-time allocation if the free block is known (but typically linear time for finding free space).

Disadvantages:

- Inefficient for very large memory systems, as the bitmap itself can become large.
- It can lead to fragmentation, especially when allocating and freeing variable-sized blocks.

Let’s consider a system with 8 blocks of memory, represented by the following bitmap:

```
Memory Blocks:  1  2  3  4  5  6  7  8

Bitmap:         0  1  0  1  1  0  1  0
```

- Block 2, 4, 5, and 7 are allocated (bit = 1).
- Block 1, 3, 6, and 8 are free (bit = 0).

If a process requests a free block of memory, the system will scan the bitmap to find an available block. In this case, it would find block 1 (since bit 0 indicates it's free), and the bitmap would then be updated:

```
Bitmap after allocation: 1  1  0  1  1  0  1  0
```

Linked list data structure and its implementation in memory allocation

A linked list is a type of data structure that consists of a set of bumps, each of which has a reference to the following knot in the chain and some data. The list's last knot has a pointer that's set to null, signifying the list's end, and the first knot is known as the head. Memory allocation constantly makes use of the linked list data structure to keep track of the accessible memory blocks.

Memory is divided into a number of equal-sized blocks in Linked List Allocation, each of which is represented by a knot in the linked list. Every knot has a pointer to the following knot in the list and a data field with the position of the memory block.

Allocation strategies in Linked List Allocation

In Linked List Allocation, the allocator searches the linked list for a memory block of the desired size according to the allocation strategies. In Linked List distribution, there are three typical distribution methods −

First-fit − In this tactic, the memory allocator begins its look at the head of the linked list and allocates the first memory block that is big enough to accommodate the request. The memory may become fragmented, but this is the simplest and fastest allocation method.

Best-fit − In this approach, the allocator looks through the complete linked list in search of the smallest memory block that is sizable enough to accommodate the request. While fragmentation is reduced, the additional searching that must be done could slow down allocation periods.

Worst-fit − Using this tactic, the allocator looks through the complete linked list for the biggest memory block that satisfies the request. While this may lead to bigger unused memory blocks and more fragmentation, it can be advantageous when large memory blocks are required.

Advantages of Linked List allocation

In terms of memory management, Linked List Allocation has several benefits, including −

Flexibility − Linked List Allocation is a flexible option for dynamic memory allocation because it provides efficient memory allocation and deallocation.

Efficiency − Linked List Allocation allows for the efficient allocation and deallocation of memory by reducing fragmentation and maximizing memory utilization.

Scalability − Linked List Allocation is scalable for programs with changing memory usage patterns because it can manage varying memory requirements.

Disadvantages of Linked List allocation

Additionally, Linked List Allocation has some drawbacks, such as −

Memory overhead − Linked List Allocation can result in higher memory overhead because it needs extra memory to store the linked list.

Time complexity − When compared to other memory allocation strategies, Linked List Allocation may take longer to allocate and deallocate memory.

Fragmentation − Poor allocation and deallocation practices can result in fragmentation, which lowers the effectiveness of memory utilization.

Memory protection

Memory protection in an operating system (OS) is a mechanism that prevents one process from accessing the memory allocated to another process, ensuring that each process operates within its own allocated memory space. This is crucial for system stability and security, as it helps to prevent errors like a process accidentally corrupting the memory of another or malicious processes from tampering with the OS or other processes.

Here are some key components and methods involved in memory protection:

1. Virtual Memory

- Virtual memory allows each process to have the illusion that it is running on its own dedicated memory, even though multiple processes share the physical memory.
- The OS manages a mapping between virtual addresses and physical addresses using structures like page tables.
- This enables processes to access memory independently without interfering with each other.

2. Memory Segmentation

- Memory is divided into segments, such as code, data, stack, and heap. Each segment has its own protections (e.g., read-only, read-write).
- This ensures, for example, that code cannot be accidentally modified or that stack overflows don’t corrupt other segments.

3. Page-Based Memory Protection

- The memory is divided into small blocks called "pages" (typically 4 KB in size).
- The OS uses a **page table** to map virtual addresses to physical addresses.
- Each page can have specific permissions, such as:
  - **Read**: the process can read from the memory.
  - **Write**: the process can write to the memory.
  - **Execute**: the process can execute code from that memory region.
  - **No access**: the process cannot access that memory region at all.

These permissions help prevent unauthorized or dangerous access to memory.

4. Hardware Support: Memory Management Unit (MMU)

- The MMU is a hardware component that helps implement memory protection.
- It translates virtual addresses to physical addresses and enforces access controls (read, write, execute) for each page.
- The MMU uses the page table provided by the OS to check the permissions of each memory access attempt.

5. Privilege Levels

- Most operating systems use different privilege levels (user mode vs kernel mode).
- In **user mode**, processes can only access their allocated memory, while in **kernel mode**, the OS has unrestricted access to all memory.
- This separation prevents user applications from directly manipulating critical OS structures or interfering with other applications.

Paging is a memory management scheme used by operating systems to allow efficient and safe access to memory. It breaks down the physical memory into small fixed-sized blocks called **pages** and divides the logical memory (used by programs) into blocks of the same size, also called **page frames**. This allows the system to map logical addresses to physical addresses, which helps in utilizing memory more efficiently.

Pages and Page Frames:

- A page is a fixed-length contiguous block of virtual memory.
- A page frame is a fixed-length block of physical memory.

Page Table:

- It maps virtual addresses (used by programs) to physical addresses (used by hardware). The page table stores the base address of each page in physical memory.

Address Translation:

- When a program accesses a memory location, the system uses the page table to translate the virtual address to a physical address.
- The virtual address is split into two parts: the **page number** (which identifies the page) and the **offset** (which identifies the location within the page).

Page Fault:

- If a program tries to access a page that is not currently loaded into physical memory, the system experiences a "page fault". The operating system then loads the page from secondary storage (e.g., hard drive) into physical memory.

TLB (Translation Lookaside Buffer):

- A cache used by the processor to store recent translations of virtual addresses to physical addresses. This improves the performance of address translation.

Segmentation and Paging:

- Paging avoids fragmentation issues that can occur with segmentation. However, it can cause **internal fragmentation** because each page must be of a fixed size, and not all parts of a page may be used by a program.

Page Replacement Algorithms

Page replacement algorithms are techniques used in operating systems to manage memory efficiently when the virtual memory is full. When a new page needs to be loaded into physical memory, and there is no free space, these algorithms determine which existing page to replace.

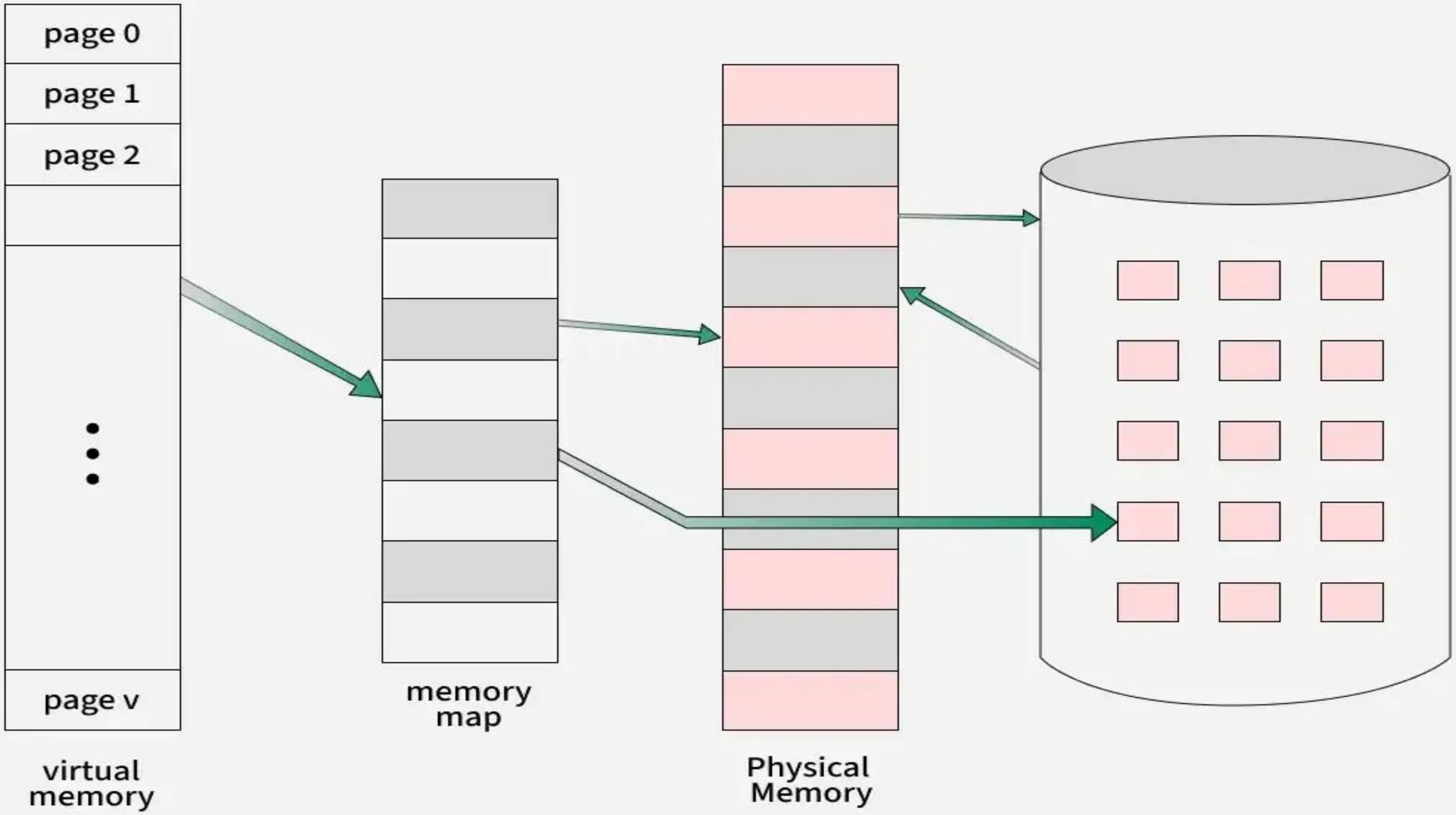
Common Page Replacement Techniques

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

Virtual Memory

Virtual memory is a memory management technique used by operating systems to give the appearance of a large, continuous block of memory to applications, even if the physical memory (RAM) is limited. It allows larger applications to run on systems with less RAM.

- The main objective of virtual memory is to support multiprogramming, The main advantage that virtual memory provides is, a running process does not need to be entirely in memory.
- Programs can be larger than the available physical memory. Virtual Memory provides an abstraction of main memory, eliminating concerns about storage limitations.
- A memory hierarchy, consisting of a computer system’s memory and a disk, enables a process to operate with only some portions of its address space in RAM to allow more processes to be in memory.


page 0
page 1
page 2
page v
virtual memory
memory map
Physical Memory

Types of Virtual Memory

There are two main types of virtual memory:

- Paging
- Segmentation

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. 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.

Segmentation

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

Dining Philosopher Problem

Problem Statement

Five philosophers are seated around a circular table. Each one has a plates spaghetti. The spaghetti is so slipper that a philosopher needs two forks to eat it. Between each pair of plates is one fork. The life of philosopher consists of alternate periods of eating and thinking. When philosopher gets hungry, she tries to acquire her left and right fork, one at a time, in either order. If successful to get two forks, she eats for a while, then puts down the forks and continue to think.

Solution using semaphores

```
#define N 5 /*no. of philosophers*/
#define LEFT ( i-1)%N /* no of i’s left neighbor*/
#define RIGHT (i=1) %N
#define THIKING 0
#define HUNGRY 1
#define EATING 2
Typedef int semaphore;
int state [N]; /* array to keep track of everyone’s state*/
semaphore mutex = 1;
semaphore s[N]; /* one semaphore per philosopher*/
```

```
void philosopher (int i)
{
While (TRUE){
Think ();
take_forks (i);
eat ();
put_forks (i);
}
}
```

```
void take_forks (int i)
{
down (& mutex);
state [i] = HUNGRY;
test (i) ;
up (& mutex);
down (& s [i]);
}
```

```
void put_fork (int i)
{ down (& mutex);
state [i] = THINKING;
test (LEFT)
test (RIGHT);
up (& mutex);
}

Void test (int i)
{
If (state [i] = = HUNGRY && state [LEFT] ! = EATING && state [RIGHT] ! = EATING);
{ state [i] =EATING;
Up (& s [i]);
}
```

The TSL(Test and Set Lock) instructions

Here, the shared variable is lock which is initialized to false. TestAndSet(lock) algorithm works in this way – it always returns whatever value is sent to it and sets lock to true. The first process will enter the critical section at once as TestAndSet(lock) will return false and it'll break out of the while loop. The other processes cannot enter now as lock is set to true and so the while loop continues to be true. Mutual exclusion is ensured. Once the first process gets out of the critical section, lock is changed to false. So, now the other processes can enter one by one. Progress is also ensured. However, after the first process, any process can go in. There is no queue maintained, so any new process that finds the lock to be false again can enter. So bounded waiting is not ensured.

Test and Set Pseudocode –

```
//Shared variable lock initialized to false
boolean lock;
boolean TestAndSet (boolean &target){
   boolean rv = target;
   target = true;
   return rv;
}
while(1){
   while (TestAndSet(lock));
critical section
   lock = false;
remainder section
}
```

Producer Consumer Problem using Semaphores

We have a buffer of fixed size. A producer can produce an item and can place it in the buffer. A consumer can pick items and consume them. We need to ensure that when a producer is placing an item in the buffer, then at the same time consumer should not consume any item. In this problem, the buffer is the critical section.

To solve this problem, we need two counting semaphores – Full and Empty. “Full” keeps track of some items in the buffer at any given time and “Empty” keeps track of many unoccupied slots.

Initialization of semaphores
mutex = 1
Full = 0 // Initially, all slots are empty. Thus full slots are 0
Empty = n // All slots are empty initially

Solution for Producer

```
do{

//produce an item

wait(empty);

wait(mutex);

//place in buffer

signal(mutex);

signal(full);

}while(true)
```

When producer produces an item then the value of “empty” is reduced by 1 because one slot will be filled now. The value of mutex is also reduced to prevent consumer to access the buffer. Now, the producer has placed the item and thus the value of “full” is increased by 1. The value of mutex is also increased by 1 because the task of producer has been completed and consumer can access the buffer.

Solution for Consumer

```
do{

wait(full);

wait(mutex);

// consume item from buffer

signal(mutex);

signal(empty);

}while(true)
```

As the consumer is removing an item from buffer, therefore the value of “full” is reduced by 1 and the value is mutex is also reduced so that the producer cannot access the buffer at this moment. Now, the consumer has consumed the item, thus increasing the value of “empty” by 1. The value of mutex is also increased so that producer can access the buffer now.

Readers-Writers Problem

The Readers-Writers Problem is a classic synchronization issue in operating systems that involves managing access to shared data by multiple threads or processes. The problem addresses the scenario where:

- **Readers**: Multiple readers can access the shared data simultaneously without causing any issues because they are only reading and not modifying the data.
- **Writers**: Only one writer can access the shared data at a time to ensure data integrity, as writers modify the data, and concurrent modifications could lead to data corruption or inconsistencies.

Challenges of the Reader-Writer Problem

The challenge now becomes how to create a synchronization scheme such that the following is supported:

- **Multiple Readers**: A number of readers may access simultaneously if no writer is presently writing.
- **Exclusion for Writers**: If one writer is writing, no other reader or writer may access the common resource.

Solution of the Reader-Writer Problem

There are two fundamental solutions to the Readers-Writers problem:

- **Readers Preference:** In this solution, readers are given preference over writers. That means that till readers are reading, writers will have to wait. The Writers can access the resource only when no reader is accessing it.
- **Writer's Preference:** Preference is given to the writers. It simply means that, after arrival, the writers can go ahead with their operations; though perhaps there are readers currently accessing the resource.

Focus on the solution Readers Preference in this paper. The purpose of the Readers Preference solution is to give a higher priority to the readers to decrease the waiting time of the readers and to make the access of resource more effective for readers.

Sleeping Barber problem

There is a barber shop with one barber and a number of chairs for waiting customers. Customers arrive at random times and if there is an available chair, they take a seat and wait for the barber to become available. If there are no chairs available, the customer leaves. When the barber finishes with a customer, he checks if there are any waiting customers. If there are, he begins cutting the hair of the next customer in the queue. If there are no customers waiting, he goes to sleep.

The problem is to write a program that coordinates the actions of the customers and the barber in a way that avoids synchronization problems, such as deadlock or starvation.

One solution to the Sleeping Barber problem is to use semaphores to coordinate access to the waiting chairs and the barber chair. The solution involves the following steps:

Initialize two semaphores: one for the number of waiting chairs and one for the barber chair. The waiting chairs semaphore is initialized to the number of chairs, and the barber chair semaphore is initialized to zero.

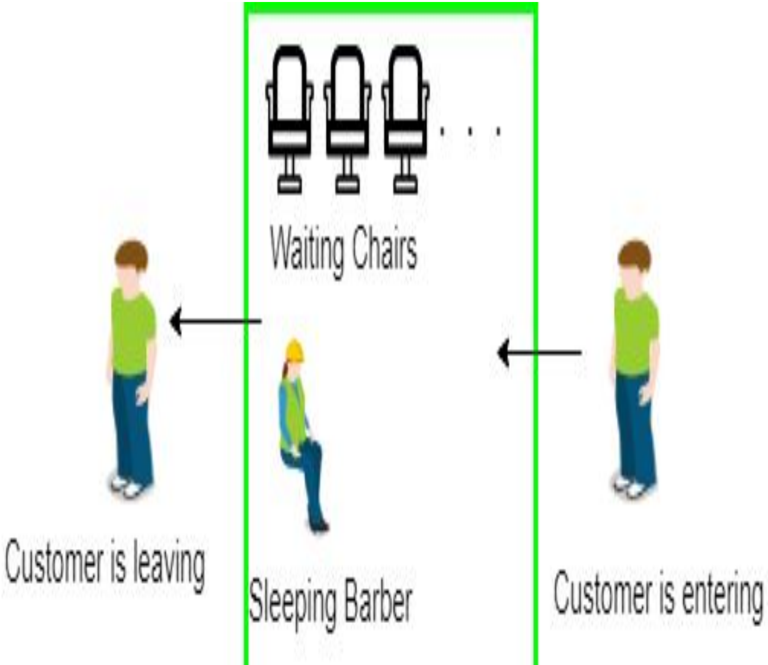
Customers should acquire the waiting chairs semaphore before taking a seat in the waiting room. If there are no available chairs, they should leave.

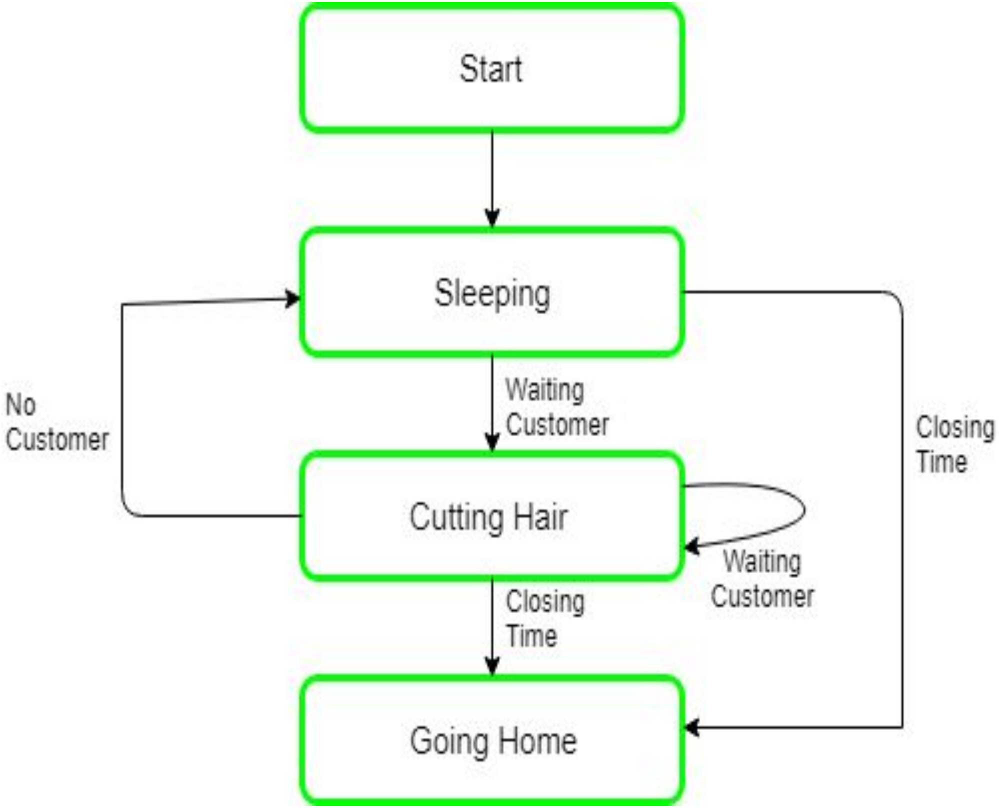
When the barber finishes cutting a customer’s hair, he releases the barber chair semaphore and checks if there are any waiting customers. If there are, he acquires the barber chair semaphore and begins cutting the hair of the next customer in the queue.

The barber should wait on the barber chair semaphore if there are no customers waiting.

The solution ensures that the barber never cuts the hair of more than one customer at a time, and that customers wait if the barber is busy. It also ensures that the barber goes to sleep if there are no customers waiting.

However, there are variations of the problem that can require more complex synchronization mechanisms to avoid synchronization issues. For example, if multiple barbers are employed, a more complex mechanism may be needed to ensure that they do not interfere with each other.

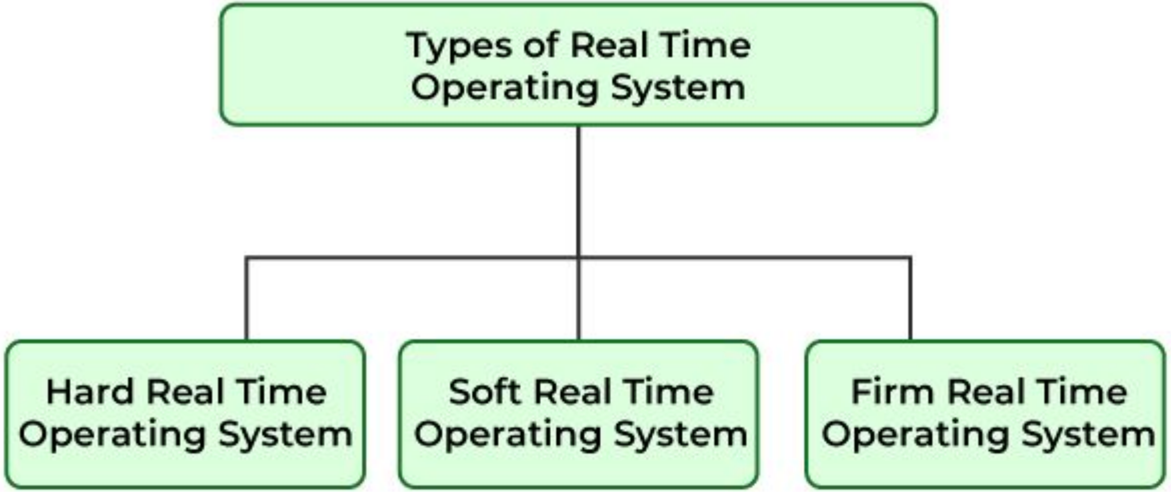

Waiting Chairs
Customer is leaving
Sleeping Barber
Customer is entering


Start
Sleeping
Waiting Customer
No Customer
Closing Time
Cutting Hair
Waiting Customer
Closing Time
Going Home

Unit 7: New Trends in OS

Real-time **operating systems (RTOS)** are used in environments where a large number of events, mostly external to the computer system, must be accepted and processed in a short time or within certain deadlines. such applications are industrial control, telephone switching equipment, flight control, and real-time simulations.

Examples of real-time operating systems are airline traffic control systems, Command Control Systems, airline reservation systems, Heart pacemakers, Network Multimedia Systems, robots, etc.


Types of Real Time Operating System
Hard Real Time Operating System
Soft Real Time Operating System
Firm Real Time Operating System

Hard Real-Time Operating System

These operating systems guarantee that critical tasks are completed within a range of time. For example, a robot is hired to weld a car body. If the robot welds too early or too late, the car cannot be sold, so it is a hard real-time system that requires complete car welding by the robot hardly on time., scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.

Soft Real-Time Operating System

This operating system provides some relaxation in the time limit. For example – Multimedia systems, digital audio systems, etc. Explicit, programmer-defined, and controlled processes are encountered in real-time systems. A separate process is changed by handling a single external event. The process is activated upon the occurrence of the related event signaled by an interrupt.

Firm Real-time Operating System

RTOS of this type have to follow deadlines as well. In spite of its small impact, missing a deadline can have unintended consequences, including a reduction in the quality of the product. Example: Multimedia applications.

The main goal of an RTOS is to perform critical tasks on time. It ensures that certain processes are finished within strict deadlines, making it perfect for situations where timing is very important. It is also good at handling multiple tasks at once.

An RTOS provides real-time control over hardware resources, like random access memory (RAM), by ensuring predictable and reliable behavior. It uses system resources efficiently while maintaining high reliability and responsiveness. By managing multiple tasks effectively, an RTOS ensures smooth operation even when the system is under heavy use or changing conditions.

Uses of RTOS

- Defense systems likeRADAR .
- Air traffic control system.
- Networked multimedia systems.
- Medical devices like pacemakers.
- Stock trading applications.

Advantages
The advantages of real-time operating systems are as follows:

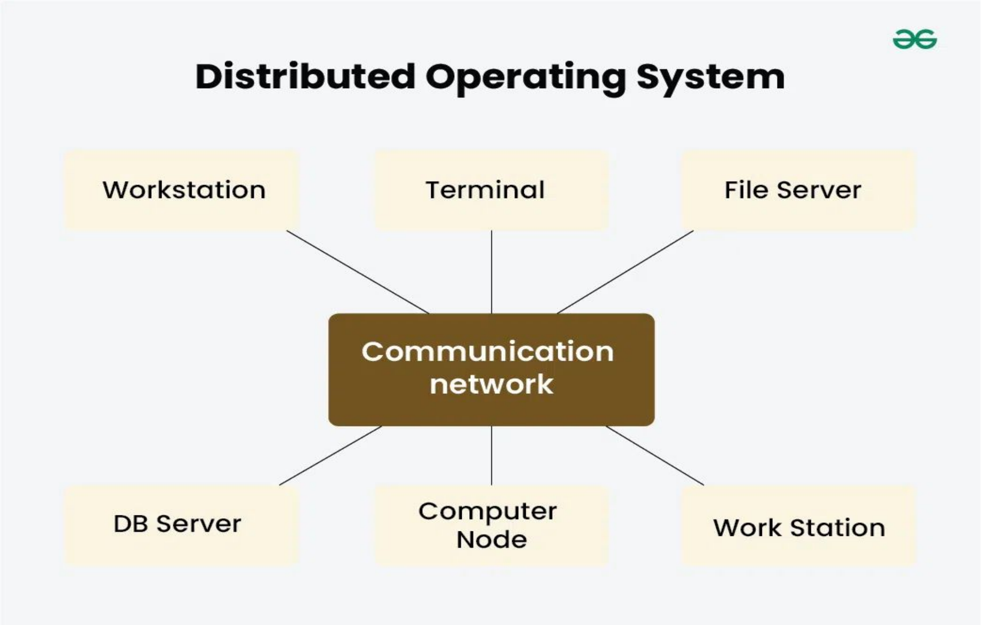
- **Maximum Consumption:** Maximum utilization of devices and systems. Thus more output from all the resources.
- **Task Shifting:** Time assigned for shifting tasks in these systems is very less. For example, in older systems, it takes about 10 microseconds. Shifting one task to another and in the latest systems, it takes 3 microseconds.
- **Focus On Application:** Focus on running applications and less importance to applications that are in the queue.
- **Real-Time Operating System In Embedded System:** Since the size of programs is small, RTOS can also be embedded systems like in transport and others.
- **Error Free:** These types of systems are error-free.
- **Memory Allocation:** Memory allocation is best managed in these types of systems.

Disadvantages
The disadvantages of real-time operating systems are as follows:

- **Limited Tasks:** Very few tasks run simultaneously, and their concentration is very less on few applications to avoid errors.
- **Use Heavy System Resources:** Sometimes the system resources are not so good and they are expensive as well.
- **Complex Algorithms** : The algorithms are very complex and difficult for the designer to write on.
- **Device Driver And Interrupt Signals:** It needs specific device drivers and interrupts signals to respond earliest to interrupts.
- **Thread Priority:** It is not good to set thread priority as these systems are very less prone to switching tasks.
- **Minimum Switching:** RTOS performs minimal task switching.

Distributed Operating System

In a Distributed Operating System, multiple CPUs are utilized, but for end-users, it appears as a typical centralized operating system. It enables the sharing of various resources such as CPUs, disks, network interfaces, nodes, and computers across different sites, thereby expanding the available data within the entire system.



A Distributed Operating System refers to a model in which applications run on multiple interconnected computers, offering enhanced communication and integration capabilities compared to a network operating system.

Applications of Distributed Operating System

Cloud Computing Platforms:

- Distributed operating systems form the backbone of cloud computing platforms like Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP).

Internet of Things (IoT):

- Distributed operating systems play a crucial role in IoT networks, where numerous interconnected devices collect and exchange data.

Distributed Databases:

- Distributed operating systems are used in distributed database management systems (DDBMS) to manage and coordinate data storage and processing across multiple nodes or servers.

Content Delivery Networks (CDNs):

- CDNs rely on distributed operating systems to deliver web content, media, and applications to users worldwide.

High-Performance Computing (HPC):

- Distributed operating systems are employed in HPC clusters and supercomputers to coordinate parallel processing tasks across multiple nodes or compute units.

Distributed File Systems:

- Distributed operating systems power distributed file systems like Hadoop Distributed File System (HDFS), Google File System (GFS), and CephFS.

Examples of Distributed Operating System

- **Solaris:** The SUN multiprocessor workstations are the intended use for it.
- **OSF/1:** The Open Foundation Software Company designed it, and it works with Unix.
- **Micros:** All nodes in the system are assigned work by the MICROS operating system, which also guarantees a balanced data load.
- **DYNIX:** It is created for computers with many processors, known as Symmetry.

Cloud OS

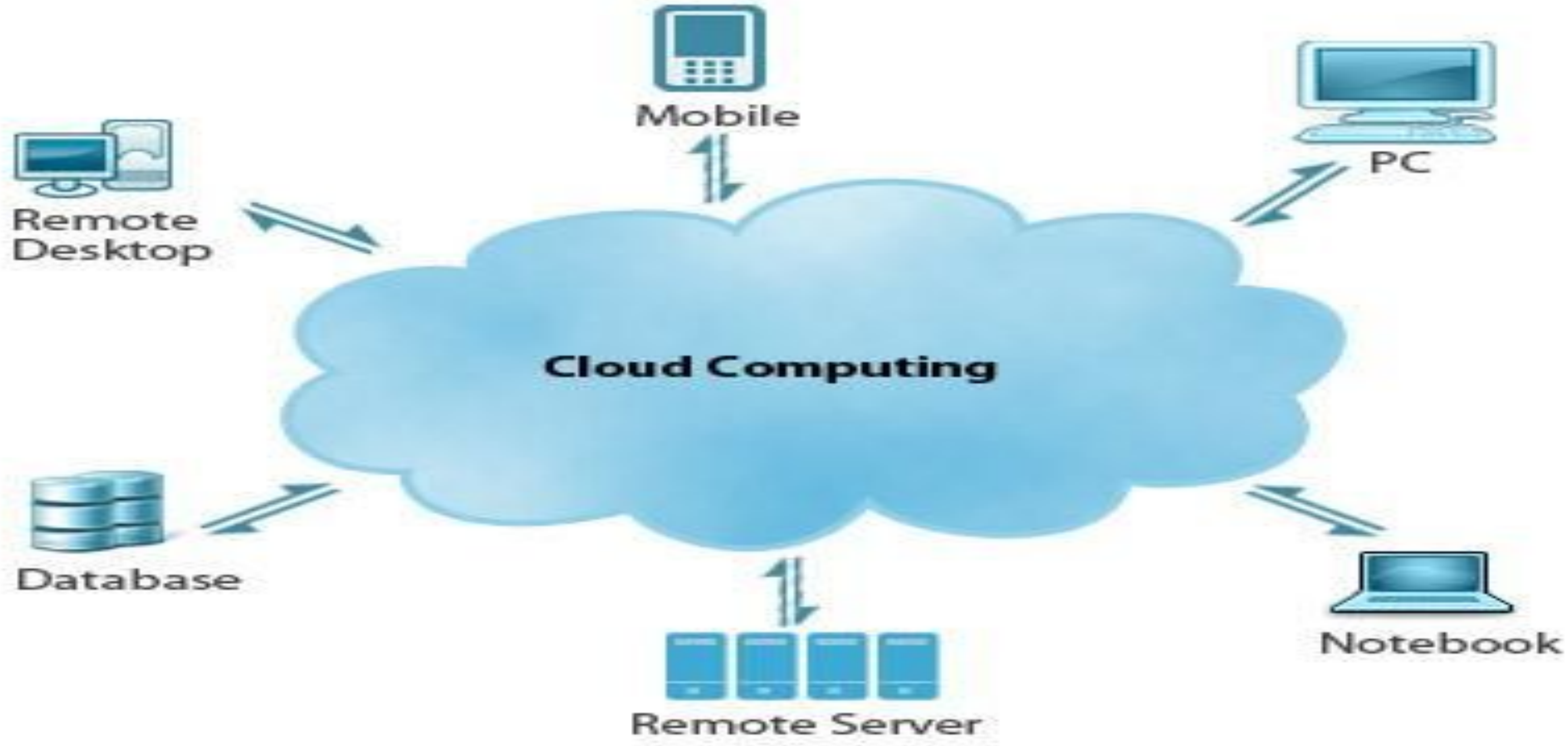
A cloud operating system (Cloud OS), also known as a virtual operating system, is a type of operating system designed to manage and deliver cloud-based services and resources across distributed servers, enabling seamless cloud computing operations.

Cloud OS manages the essential operating processes required to manage virtual infrastructures and, in some cases, the back-end hardware or software resources.

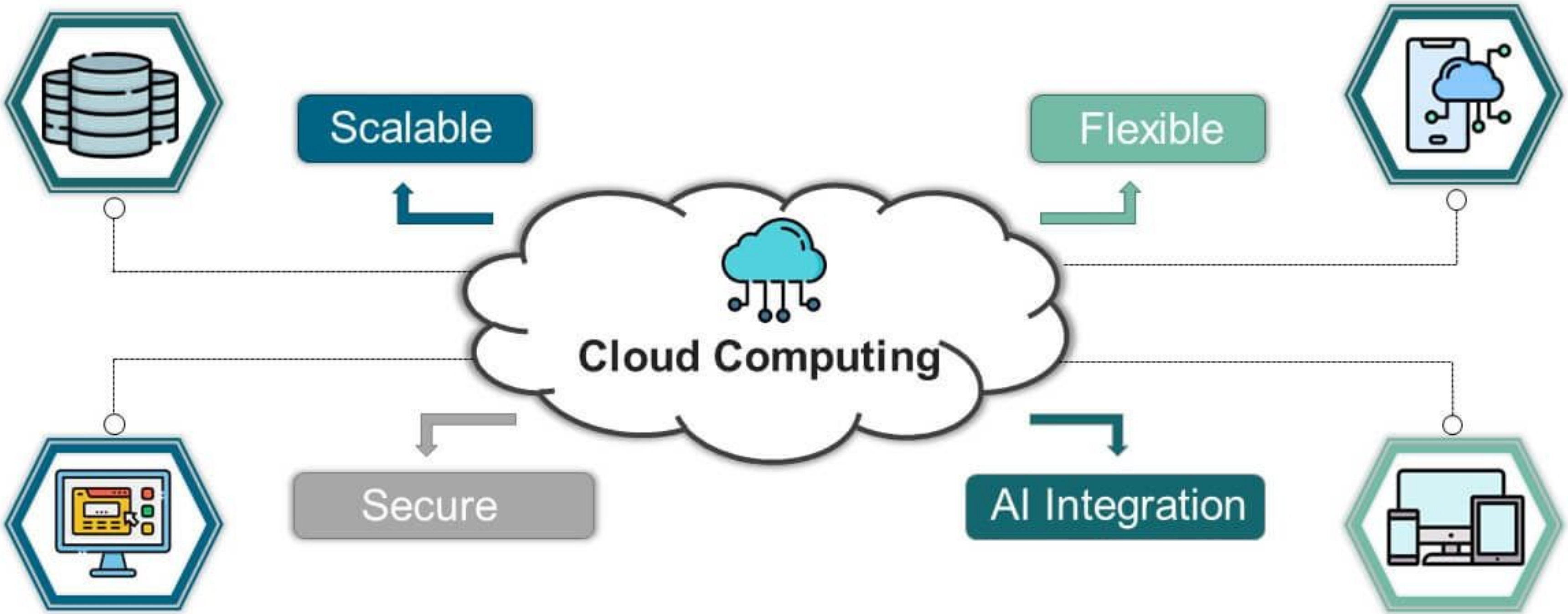
It integrates hardware, software, and networking to enable seamless cloud computing, handling complexities like scalability, multi-tenancy, and resource allocation.

Businesses can leverage Cloud OS for a unified platform that supports cloud-native applications and services, ensuring smooth operations and flexibility.

Unlike traditional operating systems installed on a single device, a Cloud OS operates across distributed servers, enabling access to data and applications from anywhere with an internet connection.


Mobile
Remote Desktop
PC
Cloud Computing
Database
Notebook
Remote Server

Cloud Operating Systems



Scalable

Flexible

Cloud Computing

Secure

AI Integration

Some examples of cloud operating systems include Infor OS, Microsoft Windows Azure, Google Chrome OS, Amazon Web Services (AWS), Alibaba Cloud, and Huawei Cloud.

Cloud OS relies on virtualization technology, which allows multiple users to share the same physical resources without interfering with each other's work.

Bottleneck Condition

A bottleneck condition is a limitation in a system that makes it difficult for data, resources, or activities to flow through it, which lowers overall performance. It acts as a limiting factor, constraining the system’s ability to perform tasks at its optimal speed and efficiency. Understanding the various bottleneck conditions is essential for diagnosing and addressing them effectively.

Types of Bottleneck Conditions

1. CPU Bottlenecks

CPU bottlenecks occur when the central processing unit is unable to handle the volume of processing tasks, leading to significant delays in task execution and overall system responsiveness. Such bottlenecks often arise due to intensive computational tasks, poorly optimized code, or inefficient multithreading, where the CPU becomes the limiting factor in processing data.

2. Memory Bottlenecks

Memory bottlenecks occur when the system’s memory resources are insufficient to meet the demands of data processing and storage, leading to increased access times and decreased overall system performance. This bottleneck type often arises when the volume of data exceeds the available memory capacity, causing frequent data swapping between the RAM and the disk, resulting in significant latency and decreased throughput.

3. Network Bottlenecks

Network bottlenecks occur when the network bandwidth is insufficient to handle the data transmission requirements, resulting in communication delays, packet losses, and degraded network performance. Such bottlenecks often emerge in scenarios where large volumes of data are being transmitted over the network, leading to congestion and reduced data transfer speeds.

4. Storage Bottlenecks

Storage bottlenecks occur when the storage infrastructure is unable to handle the data storage and retrieval demands efficiently, leading to increased latency, slow data access times, and potential data loss. Such bottlenecks often arise due to storage device limitations, inadequate storage configurations, or improper data access patterns that result in excessive disk I/O operations.

Causes of Bottlenecks

1. **Insufficient Resources Allocation:** Improper distribution of resources, such as CPU, memory, network bandwidth, or storage, can lead to bottleneck conditions within the system.
2. **Inefficient Code or Algorithms:** Poorly optimized code or inefficient algorithms can significantly impact system performance, leading to bottlenecks during data processing and execution.
3. **Hardware Limitations:** Outdated hardware or insufficient hardware capabilities may result in bottleneck conditions, especially when handling complex and resource-intensive tasks.

Consequences of Bottleneck Conditions

1. **Reduced Throughput:** Bottleneck conditions can lead to a decrease in the overall throughput of the system, hampering the ability to handle concurrent operations efficiently.
2. **Increased Latency:** Users may experience increased response times and delays in data retrieval due to bottleneck conditions, leading to a degraded user experience.
3. **System Instability:** Prolonged bottleneck conditions can cause system instability, leading to crashes, errors, and potential data loss, thereby impacting the system’s reliability and integrity.

Mitigation Strategies

- **Optimized Resource Allocation**: Ensure proper allocation of resources based on the system's requirements, optimizing CPU, memory, network bandwidth, and storage capacities to prevent bottleneck conditions.
- **Code Optimization and Algorithm Refinement**: Conduct regular code reviews and optimizations to enhance the efficiency of the system's codebase and algorithms, minimizing the risk of bottleneck conditions during data processing and execution.
- **Infrastructure Scaling and Upgradation**: Consider scaling the infrastructure or upgrading hardware components to accommodate the growing demands and prevent potential bottleneck conditions caused by resource limitations.
- **Load Balancing Techniques**: Implement effective load balancing techniques to distribute the workload evenly across multiple servers or resources, preventing any single point of failure and mitigating the risk of bottleneck conditions.