Unit 1
Introduction

Operating System

Operating System is a system software that control and coordinated all the parts of computer system. It acts as interface between user and computer. It manages computer hardware and software resources, and provides common services for computer programs. Example: Windows, macOS, Linux, Unix etc.

Function of Operating System

- Resource Management: The OS acts like a master organizer, allocating resources like memory, storage, and processors to different programs running on the computer. It ensures everything runs smoothly without conflicts. Imagine it as a traffic controller for your computer's resources.
- Process Management: The OS juggles multiple programs running at once. It decides which program gets the CPU's attention at what time, ensuring efficient execution and preventing crashes.
- File Management: The OS keeps track of your files and folders on the hard drive. It lets you create, delete, organize, and access them whenever you need them. Think of it like a librarian for your computer's files.
- Device Management: The OS acts as an interpreter between your computer and its devices like printers, scanners, and keyboards. It translates your requests to a language the devices understand and vice versa.
- Security: The OS acts as a guard, protecting your computer from unauthorized access, viruses, and other malicious software. It keeps your system safe and sound.
- User Interface: The OS provides a user interface (UI) like the graphical desktop you see on most computers. This UI lets you interact with your computer and its programs easily.

Evolution of Operating System

Operating Systems have evolved in past years. It went through several changes before getting its original form. These changes in the operating system are known as the **evolution of operating systems**. OS improve itself with the invention of new technology. Basically , OS added the feature of new technology and making itself more powerful. Let us see the evolution of operating system year-wise in detail:

- No OS – (0s to 1940s)
- Batch Processing Systems -(1940s to 1950s)
- Multiprogramming Systems -(1950s to 1960s)
- Time-Sharing Systems -(1960s to 1970s)
- Introduction of GUI -(1970s to 1980s)
- Networked Systems – (1980s to 1990s)
- Mobile Operating Systems – (Late 1990s to Early 2000s)
- AI Integration – (2010s to ongoing)

1. No OS – (0s to 1940s)

- As we know that before 1940s, there was no use of OS . Earlier, people are lacking OS in their computer system so they had to manually type instructions for each tasks in machine language(0-1 based language) . And at that time , it was very hard for users to implement even a simple task. And it was very time consuming and also not user-friendly . Because not everyone had that much level of understanding to understand the machine language and it required a deep understanding.

2. Batch Processing Systems -(1940s to 1950s)

- With the growth of time, batch processing system came into the market .Now Users had facility to write their programs on punch cards and load it to the computer operator. And then operator make different batches of similar types of jobs and then serve the different batch(group of jobs) one by one to the CPU .CPU first executes jobs of one batch and them jump to the jobs of other batch in a sequence manner.

3. Multiprogramming Systems -(1950s to 1960s)

- Multiprogramming was the first operating system where actual revolution began. It provide user facility to load the multiple program into the memory and provide a specific portion of memory to each program. When one program is waiting for any I/O operations (which take much time) at that time the OS give permission to CPU to switch from previous program to other program(which is first in ready queue) for continuous execution of program with interrupt.

4. Time-Sharing Systems -(1960s to 1970s)

- Time-sharing systems is extended version of multiprogramming system. Here one extra feature was added to avoid the use of CPU for long time by any single program and give access of CPU to every program after a certain interval of time. Basically OS switches from one program to another program after a certain interval of time so that every program can get access of CPU and complete their work.

5. Introduction of GUI -(1970s to 1980s)

- With the growth of time, Graphical User Interfaces (GUIs) came. First time OS became more user-friendly and changed the way of people to interact with computer. GUI provides computer system visual elements which made user's interaction with computer more comfortable and user-friendly. User can just click on visual elements rather than typing commands. Here are some feature of GUI in Microsoft's windows icons, menus and windows.

6. Networked Systems – (1980s to 1990s)

- At 1980s,the craze of computer networks at it's peak .A special type of Operating Systems needed to manage the network communication . The OS like Novell NetWare and Windows NT were developed to manage network communication which provide users facility to work in collaborative environment and made file sharing and remote access very easy.

7. Mobile Operating Systems – (Late 1990s to Early 2000s)

- Invention of smartphones create a big revolution in software industry, To handle the operation of smartphones , a special type of operating systems were developed. Some of them are : iOS and Android etc. These operating systems were optimized with the time and became more powerful.

8. AI Integration – (2010s to ongoing)

- With the growth of time, Artificial intelligence came into picture. Operating system integrates features of AI technology like Siri, Google Assistant, and Alexa and became more powerful and efficient in many way. These AI features with operating system create a entire new feature like voice commands, predictive text, and personalized recommendations.

Note: The above-mentioned OS basically tells how the OS evolved with the time by adding new features but it doesn't mean that only new generation OS are in use and previously OS system are not in use, according to the need, all these OS are still used in software industry.

Types of Operating System

There are several types of Operating System which are mentioned below:

i. Batch Operating System
ii. Multi-Programming System
iii. Multi-Processing System
iv. Multi-Tasking Operating System
v. Time-Sharing Operating System
vi. Distributed Operating System
vii. Network Operating System
viii. Real-Time Operating System

1. Batch Operating System

This type of operating system does not interact with the computer directly. There is an operator which takes similar jobs having the same requirement and groups them into batches. It is the responsibility of the operator to sort jobs with similar needs.

Advantages

- ❖ It is very difficult to guess or know the time required for any job to complete. Processors of the batch systems know how long the job would be when it is in the queue.
- ❖ Multiple users can share the batch systems.
- ❖ The idle time for the batch system is very less.
- ❖ It is easy to manage large work repeatedly in batch systems.

Disadvantages

- ❖ The computer operators should be well known with batch systems.
- ❖ Batch systems are hard to debug.
- ❖ It is sometimes costly.
- ❖ The other jobs will have to wait for an unknown time if any job fails.

Examples of Batch Operating Systems: Payroll Systems, Bank Statements, etc.

2. Multiprogramming Operating System

Multiprogramming Operating System can be simply illustrated as more than one program is present in the main memory and any one of them can be kept in execution. This is basically used for better execution of resources.

Advantages

- ❖ Multi Programming increases the Throughput of the System.
- ❖ It helps in reducing the response time.

Disadvantages

- ❖ There is not any facility for user interaction of system resources with the system.

Examples: Windows Operating System, MAC OS, Android OS etc.

3. Multi-Processing Operating System

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

Advantages

- ❖ It increases the throughput of the system.
- ❖ As it has several processors, so, if one processor fails, we can proceed with another processor.

Disadvantages

- ❖ Due to the multiple CPU, it can be more complex and somehow difficult to understand.

Examples: UNIX, LINUX, and Solaris etc.

4. Multi-Tasking Operating System

Multitasking Operating System is simply a multiprogramming Operating System with having facility of a Round-Robin Scheduling Algorithm. It can run multiple programs simultaneously.

There are two types of Multi-Tasking Systems which are listed below.

- Preemptive Multi-Tasking
- Cooperative Multi-Tasking

Advantages

- ❖ Multiple Programs can be executed simultaneously in Multi-Tasking Operating System.
- ❖ It comes with proper memory management.

Disadvantages

- ❖ The system gets heated in case of heavy programs multiple times.

Examples: Windows, macOS, and Linux etc.

5. Time Sharing Operating System

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. These systems are also known as Multitasking Systems. 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.

Advantages

- Each task gets an equal opportunity.

- Fewer chances of duplication of software.
- CPU idle time can be reduced.
- Resource Sharing: Time-sharing systems allow multiple users to share hardware resources such as the CPU, memory, and peripherals, reducing the cost of hardware and increasing efficiency.
- Improved Productivity: Time-sharing allows users to work concurrently, thereby reducing the waiting time for their turn to use the computer. This increased productivity translates to more work getting done in less time.

Disadvantages

- Reliability problem.
- One must have to take care of the security and integrity of user programs and data.
- Data communication problem.
- High Overhead: Time-sharing systems have a higher overhead than other operating systems due to the need for scheduling, context switching, and other overheads that come with supporting multiple users.

Examples IBM VM/CMS, TSO, Windows Terminal Services etc.

6. Distributed Operating System

These types of operating system is a recent advancement in the world of computer technology and are being widely accepted all over the world and, that too, at a great pace. Various autonomous interconnected computers communicate with each other using a shared communication network. Independent systems possess their own memory unit and CPU. These are referred to as loosely coupled systems or distributed systems. These systems' processors differ in size and function. 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 i.e., remote access is enabled within the devices connected in that network.

Advantages

- ❖ Failure of one will not affect the other network communication, as all systems are independent of each other.
- ❖ Electronic mail increases the data exchange speed.
- ❖ Since resources are being shared, computation is highly fast and durable.
- ❖ Load on host computer reduces.
- ❖ These systems are easily scalable as many systems can be easily added to the network.
- ❖ Delay in data processing reduces.

Disadvantages

- ❖ Failure of the main network will stop the entire communication.
- ❖ To establish distributed systems the language is used not well-defined yet.
- ❖ These types of systems are not readily available as they are very expensive. Not only that the underlying software is highly complex and not understood well yet.

Examples LOCUS etc.

7. Network Operating System

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. One more important aspect of Network Operating Systems is that all the users are well aware of the underlying configuration, of all other users within the network, their individual connections, etc. and that's why these computers are popularly known as tightly coupled systems.

Advantages

- Highly stable centralized servers.
- Security concerns are handled through servers.
- New technologies and hardware up-gradation are easily integrated into the system.
- Server access is possible remotely from different locations and types of systems.

Disadvantages

- Servers are costly.
- User has to depend on a central location for most operations.
- Maintenance and updates are required regularly.

Examples: Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, BSD, etc.

8. Real Operating System

These types of OSs serve real-time systems. The time interval required to process and respond to inputs is very small. This time interval is called **response time**.

Real-time systems are used when there are time requirements that are very strict like missile systems, air traffic control systems, robots, etc.

Advantages

- **Maximum Consumption:** Maximum utilization of devices and systems, thus more output from all the resources.
- **Task Shifting:** The time assigned for shifting tasks in these systems is very less. For example, in older systems, it takes about 10 microseconds in shifting from one task to another, and in the latest systems, it takes 3 microseconds.
- **Focus on Application:** Focus on running applications and less importance on applications that are in the queue.
- Real-time **operating system in** the **embedded system:** Since the size of programs is small, RTOS can also be used in 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

- **Limited Tasks:** Very few tasks run at the same time and their concentration is very less on a 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 signal 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.

Components of Operating System

The components of an operating system play a key role to make a variety of computer system parts work together. There are the following components of an operating system, such as:

- Process Management
- Memory Management
- File Management
- Main Memory Management
- Secondary Storage Management
- I/O Device Management
- Security Management
- Command Interpreter System

1. Process Management

This component is responsible for handling the creation, execution, and termination of processes. A process is an instance of a computer program that is being executed. Process management determines which processes get to use the CPU and for how long.

2. Memory Management

This component allocates and manages RAM (Random Access Memory), which is the primary memory of a computer. Memory management keeps track of which parts of memory are being used by which processes and ensures that processes do not interfere with each other's memory space.

3. **File Management**

This component keeps track of files and directories on the computer's storage devices. It allows users to create, delete, rename, and access files. File management also ensures that data integrity is maintained.

4. **Storage Management**

This component manages secondary storage devices such as hard disks and solid-state drives. It is responsible for formatting, partitioning, and reading/writing data to these devices.

5. **Device Management**

This component is responsible for controlling and coordinating all peripheral devices (such as printers, scanners, and disk drives) attached to the computer. It provides a uniform interface to these devices so that application programs can interact with them without knowing the specific details of each device.

6. **Security Management**

This component is responsible for protecting the computer system from unauthorized access, use, disclosure, disruption, modification, or destruction. It includes mechanisms for user authentication, authorization, and encryption.

7. **User Interface**

This component provides the user with a way to interact with the operating system. It can be a graphical user interface (GUI) or a command-line interface (CLI).

Introduction to System call

OS contains set of routines (functions) for performing various low-level operations. A system call is a mechanism used by an application for requesting a service from the OS. I.e. It is interface between the OS & the user program.

In sense, it is like making a special kind of procedural call but only system calls enter the kernel & procedural call do not.

Introduction to Shell

An OS shell is a software component that presents a user interface to various OS functions & services. Shell provides following services

- File Management
- Process Management
- Batch Management
- OS setup

Examples of OS: UNIX

UNIX is a computer operating system that was first developed in the 1960s at Bell Labs. It is a multitasking, multi-user operating system that is known for its robustness, flexibility, and portability. UNIX is the foundation for many popular operating systems today, including Linux, macOS, and Android.

Characteristics

i. Multitasking
ii. Multi-user
iii. Command-Line Interface
iv. Shell
v. Hierarchical file system
vi. Open Source

Linux OS

Linux is a family of open-source Unix-like operating systems based on the Linux kernel, an operating system kernel first released by Linus Torvalds on September 17, 1991. It is one of the most prominent examples of free and open-source software collaboration. The Linux kernel serves as the foundation for various

distributions (distros) of Linux, each tailored for specific purposes, ranging from personal computing to embedded systems, servers, and supercomputers.

Characteristics

i. Open Source
ii. Multitasking
iii. Multiuser
iv. Stability
v. Security
vi. Variety of Distributions

Windows OS

Windows OS, developed by Microsoft, is the most popular operating system for personal computers worldwide. It's known for its user-friendly graphical interface (GUI) and wide range of software compatibility. It is a graphical operating system, meaning you interact with it using a mouse and icons on the screen, rather than just typing commands. It provides a platform for running various software applications for productivity, creativity, gaming and more.

Characteristics

i. User-friendly Interface
ii. Software compatibility
iii. Regular updates
iv. Hardware compatibility
v. Integrations with Microsoft services.

Handheld OS

Windows OS, developed by Microsoft, is the most popular operating system for personal computers worldwide. It's known for its user-friendly graphical interface (GUI) and wide range of software compatibility. It is a graphical operating system, meaning you interact with it using a mouse and icons on the screen, rather than just typing commands. It provides a platform for running various software applications for productivity, creativity, gaming and more.

Characteristics

i. User-friendly Interface
ii. Software compatibility
iii. Regular updates
iv. Hardware compatibility
v. Integrations with Microsoft services.

Assignment-I

1. Justify "Operating System as a resource manager". Explain the types of operating system in brief.
2. Explain batch System, time sharing system and real time system in brief.
3. What is shell? Explain "Operating System as a resource manager".
4. Differentiate between Multiprocessing and multiprogramming operating System.
5. Differentiate between multiprogramming and multitasking operating System.
6. Write Short notes on: System calls and Shell.
7. Define Operating System. Explain the function of Operating System.
8. Explain about the components of operating system.

Unit 2
Process Management

Process

A process is the instance of a computer program that is being executed by one or more threads (light-weight process).

In the operation system, a process is something that is currently under execution. so, an active program can be called a process. For example, when you want to search something on web then you start a browser. So, this can be a process.

Program

A **program** is a set of instruction codes that has been designed to complete a certain task. It is a passive entity stored in the secondary memory of the computer system. A program is considered as a passive and static entity. A program is like a file which contains a set of instruction codes stored on a disk in the form of an executable file. A program contains instructions written in any programming language. Programs have an unlimited span of time.

Differentiate between Process and Program

S.N	Process	Program
1	It is an instance of a program that is being currently executed.	It is a set of instructions that has been designed to complete a certain task.
2	It is an active entity.	It is a passive entity.
3	It is created when a program is in execution and is being loaded into the main memory.	It resides in the secondary memory of the system.
4	It exists for a limited amount of time and it gets terminated once the task has been completed.	It exists in a single place and continues to exist until it has been explicitly deleted.
5	It is considered as a dynamic entity.	It is considered as a static entity.
6	It has a high resource requirement.	It doesn't have a resource requirement.
7	It requires resources such as CPU, memory address, I/O during its working.	It requires memory space to store instructions.

Process States/ Process Life Cycle

- During the execution of a process, it undergoes a no. of states.
- These stages may differ in different OS, and the names of these states are also not standardized.

Each process may be in any one of the following states −

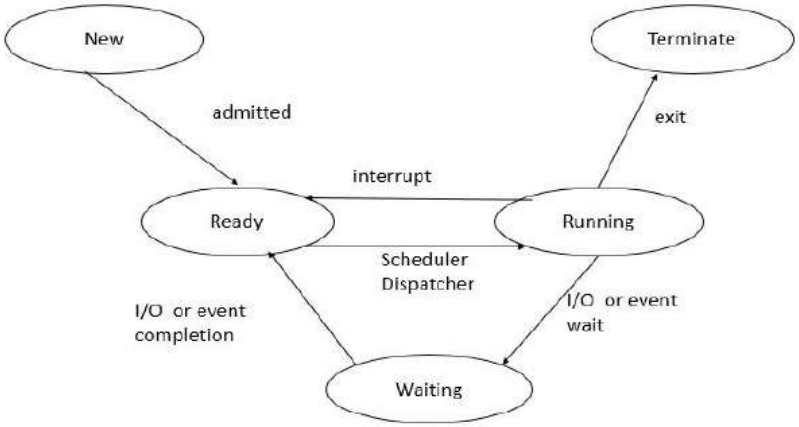
a. **New** − The process is being created.

b. **Running** − In this state the instructions are being executed.

c. **Waiting** − The process is in waiting state until an event occurs like I/O operation completion or receiving a signal.

d. **Ready** − The process is waiting to be assigned to a processor.

e. **Terminated** − the process has finished execution.



Explanation

Step 1 – Whenever a new process is created, it is admitted into ready state.

Step 2 – If no other process is present at running state, it is dispatched to running based on scheduler dispatcher.

Step 3 – If any higher priority process is ready, the uncompleted process will be sent to the waiting state from the running state.

Step 4 – Whenever I/O or event is completed the process will send back to ready state based on the **interrupt signal** given by the running state.

Step 5 – Whenever the execution of a process is completed in running state, it will exit to terminate state, which is the completion of process.

Process Models

A process model is a way of describing a series of activities that needs to be done to achieve a particular outcome. It's basically a blueprint that shows the steps involved in a process, often visually using flowcharts or diagrams.

In an operating system (OS), a process model describes the different states a program goes through during its execution. There are different models used to represent this. They are:

i. Two state Model

ii. Five State Model

iii. Six State Model

Two State Process Model

The two-state process model is a simplified version of how processes behave in an operating system. It focuses on the two fundamental states a process can be in:

i. Running: In this state, the process has control of the CPU and is actively executing instructions.

ii. Not Running: This encompasses all the other states can be in besides running. This includes:

- Waiting for I/O operations to complete (e.g., waiting for data from a disk)
- Waiting for system resources (e.g., memory)
- Waiting in a queue for its turn to run on the CPU

While the two-state model is a basic representation, it captures the core concept of process scheduling: the CPU can only execute one process at a time, and processes constantly switch between running and not running states.

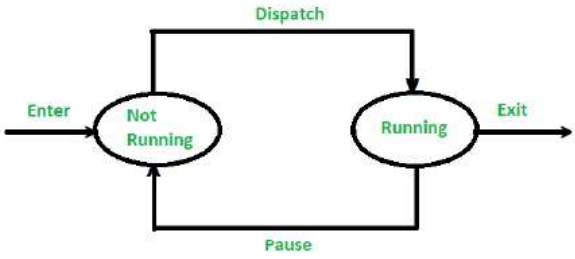
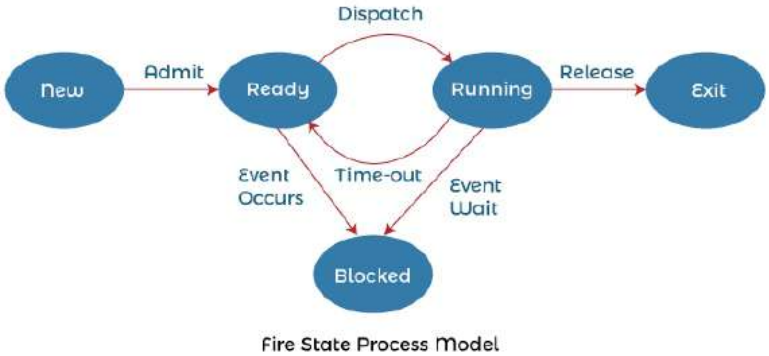


Figure: Two State Model

Five state Process Model

The 5-state process model is a more detailed representation of a process's lifecycle compared to the two-state model. It defines five distinct states a process can be in:

- **New:** This is the initial state when a new process is created. The program exists, but it's not yet loaded into main memory and resources haven't been allocated.
- **Ready:** The process is prepared to run. It's loaded into main memory and has all the necessary resources, but it's waiting for its turn on the CPU. The operating system's scheduler manages a queue of ready processes.
- **Running:** This is the lucky state! The process currently has control of the CPU and is actively executing instructions. There can only be one process in the running state at a time.
- **Blocked:** The process is temporarily halted because it's waiting for an event to occur before it can continue. This event could be:
    - I/O operation completion (e.g., waiting for data to be read from the disk)
    - A resource becoming available (e.g., waiting for another process to release a lock)
- **Exit:** The process has finished execution or has been terminated due to an error. Its resources are released, and the process is removed from the system.



Fire State Process Model

Six state Process Model

The 6-state process model is an extension of the 5-state model, adding an additional state to account for situations where the system workload is high. Here's a breakdown of the six states:

1. **New:** Same as the 5-state model - a new process is created but not yet admitted to the system.
2. **Ready:** Same as the 5-state model - the process is prepared to run and waits for the CPU.
3. **Running:** Same as the 5-state model - the process has control of the CPU and is actively executing.
4. **Blocked:** Same as the 5-state model - the process is waiting for an event (like I/O) to complete before it can continue.
5. **Exit:** Same as the 5-state model - the process has finished execution and is terminated.

6. **Suspended:** This is the additional state introduced in the 6-state model. A suspended process is temporarily halted but can be resumed later. There are two main reasons a process might be suspended:
   - ❖ **Resource limitations:** If the system is overloaded and there's not enough memory to keep all ready processes in main memory, the OS might suspend some processes and swap them out to secondary storage (like a hard disk). This frees up memory for other processes that need it immediately.
   - ❖ **Administrative intervention:** The system administrator might choose to suspend a process for various reasons, such as debugging or prioritizing other processes.

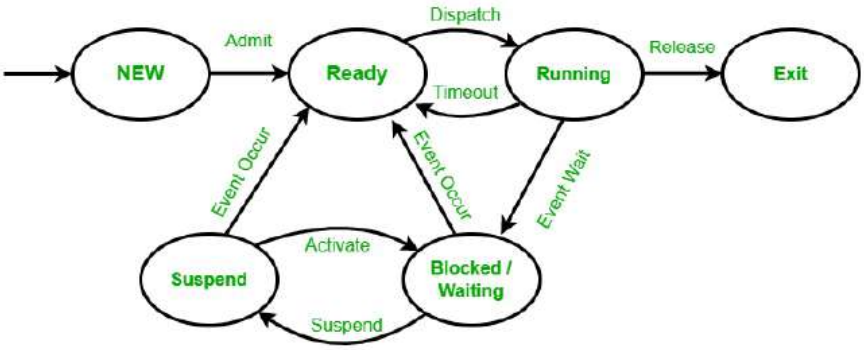


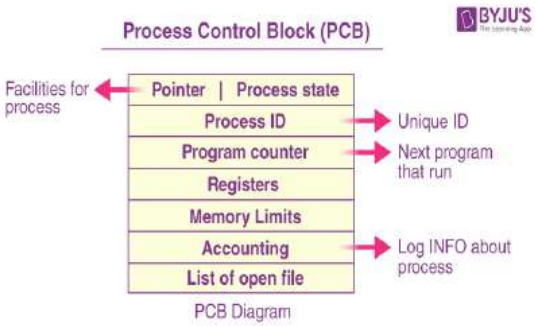
Figure: Six State Process Model

Process Control Box (PCB)

In the world of operating systems, a Process Control Block (PCB), also sometimes called a Task Control Block (TCB), is a fundamental data structure. It's essentially a file cabinet specially for a process, storing all the critical information the operating system (OS) needs to manage and control that process.

The exact contents of a PCB can vary depending on the specific operating system, but here are some general components you'll typically find:

- **Process State:** This indicates the current status of the process, such as new, ready, running, blocked, or terminated.
- **Process ID (PID):** A unique identifier for the process.
- **Program Counter (PC):** Keeps track of the memory address of the next instruction to be executed.
- **CPU Registers:** These store temporary data used by the CPU during process execution.
- **Memory Management Information:** This includes details about the memory allocated to the process, such as its starting and ending addresses.
- **I/O Status Information:** Tracks any ongoing I/O (Input/Output) operations associated with the process.
- **Priority:** Some processes might have higher priority than others, influencing scheduling decisions.
- **Accounting Information:** This might include data on CPU time used, memory used, and other metrics.



What's inside PCB?

The PCB plays a critical role in process management by providing the OS with the necessary information to:

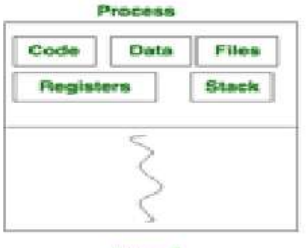
- ❖ Context Switching: When the OS switches between processes, it uses the PCB to save the state of the current process and restore the state of the process about to be run.

- ❖ Process Scheduling: The OS scheduler uses information like process state and priority from the PCB to decide which process gets the CPU next.
- ❖ Memory Management: The PCB helps the OS track memory allocated to each process and ensure efficient memory utilization.
- ❖ Synchronization and Communication: PCBs can be used to facilitate communication and synchronization between processes.

Overall, the Process Control Block acts as the control center for each process, providing the OS with the critical data it needs to juggle multiple programs efficiently.

Thread

- ❖ Thread is the segment of a process which means a process can have multiple threads and these multiple threads are contained within a process. A thread has three states: Running, Ready, and Blocked.
- ❖ The thread takes less time to terminate as compared to the process but unlike the process, threads do not isolate.



S.N	Process	Thread
1	Process means any program is in execution.	Thread means a segm
2	The process takes more time to terminate.	The thread takes less t
3	It takes more time for creation.	It takes less time
4	It also takes more time for context switching.	It takes less time for c
5	The process is less efficient in terms of communication.	Thread is more efficient in te
6	Multiprogramming holds the concepts of multi-process.	We don't need multi program threads because a single proc threads
7	The process is isolated.	Threads share
8	The process does not share data with each other.	Threads share data w

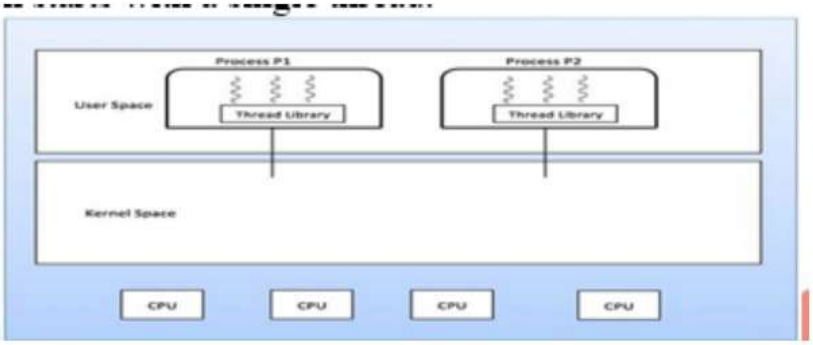
Thread Model

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

- ❖ User Level Thread Model
- ❖ Kernel Level Thread Model

i. User level Threads

In this case, the thread management kernel is not aware of the existence of threads. The thread library contains code for creating and destroying threads, for passing message and data between threads, for scheduling thread execution and for saving and restoring thread contexts. The application starts with a single thread.



Advantages

- ❖ Thread switching does not require Kernel mode privileges.
- ❖ User level thread can run on any operating system.
- ❖ Scheduling can be application specific in the user level thread.
- ❖ User level threads are fast to create and manage.

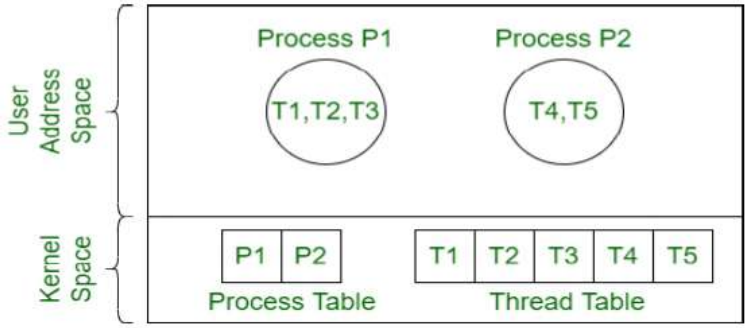
Disadvantages

- ❖ In a typical operating system, most system calls are blocking.
- ❖ Multithreaded application cannot take advantage of multiprocessing.

ii. Kernal Level Threads

In this case, thread management is done by the Kernel. There is no thread management code in the application area. Kernel threads are supported directly by the operating system. Any application can be programmed to be multithreaded. All of the threads within an application are supported within a single process.

The Kernel maintains context information for the process as a whole and for individual threads within the process. Scheduling by the Kernel is done on a thread basis. The Kernel performs thread creation, scheduling and management in Kernel space. Kernel threads are generally slower to create and manage than the user threads.



Advantages

- ❖ Kernel can simultaneously schedule multiple threads from the same process on multiple processes.
- ❖ If one thread in a process is blocked, the Kernel can schedule another thread of the same process.
- ❖ Kernel routines themselves can be multithreaded.

Disadvantages

- ❖ Kernel threads are generally slower to create and manage than the user threads.

- Transfer of control from one thread to another within the same process requires a mode switch to the Kernel.

S.N	User-Level Threads	Kernel-Level Threads
1	User-level threads are faster to create and manage.	Kernel-level threads are slower to create and manage.
2	Implementation is by a thread library at the user level.	Operating system supports creation of Kernel threads.
3	User-level thread is generic and can run on any operating system.	Kernel-level thread is specific to the operating system.
4	Multi-threaded applications cannot take advantage of multiprocessing.	Kernel routines themselves can be multithreaded.

Multithreading

Multithreading is a technique that allows a program or a process to execute many tasks concurrently. at the same time and parallel. It allows a process to run its tasks in parallel mode on a single processor system.

A multithreading is a specialized form of multitasking. Multitasking threads require less overhead than multitasking processes. I need to define another term related to threads process: A process consists of the memory space allocated by the operating system that can contain one or more threads. A thread cannot exist on its own; it must be a part of a process. A process remains running until all of the non-daemon threads are done executing.

Multithreading enables you to write very efficient programs that make maximum use of the CPU, because idle time can be kept to a minimum.

Advantages of multithreading over multitasking:

- Reduces the computation time.
- Improves performance of an application.
- Threads share the same address space so it saves the memory.
- Context switching between threads is usually less expensive than between processes.
- Cost of communication between threads is relatively low.

Disadvantages

- A multithreading system operates without interruptions.
- The code might become more intricate to comprehend.
- The costs associated with handling various threads could be excessive for straightforward tasks.
- Identifying and resolving problems may become more demanding due to the intricate nature of the code.

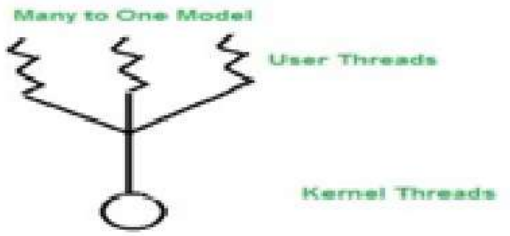
Multithreading Model

Many operating systems support kernel thread and user thread in a combined way. Example of such system is Solaris. Multi-threading model are of three types.

i. Many-to-one
ii. One-to-one
iii. Many-to-many

i. Many-to-one Model

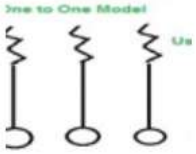
In this model, we have multiple user threads mapped to one kernel thread. In this model when a user thread makes a blocking system call entire process blocks. As we have only one kernel thread and only one user thread can access kernel at a time, so multiple threads are not able access multiprocessor at the same time. The thread management is done on the user level so it is more efficient.



ii. One-to-one Model

In this model, one to one relationship between kernel and user thread. In this model multiple thread can run on multiple processors. Problem with this model is that creating a user thread requires the corresponding kernel thread.

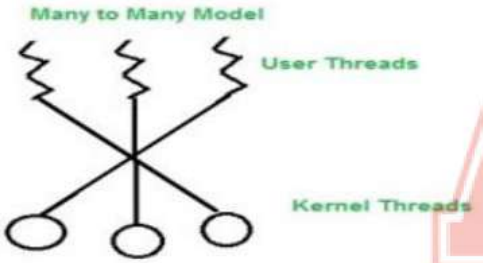
As each user thread is connected to different kernel, if any user thread makes a blocking system call, the other user threads won't be blocked.



iii. Many-to-Many Model

In this model, we have multiple user threads multiplex to same or lesser number of kernel level threads. Number of kernel level threads are specific to the machine; advantage of this model is if a user thread is blocked we can schedule others user thread to other kernel thread. Thus, System doesn't block if a particular thread is blocked.

It is the best multi-threading model.



CPU Scheduling

CPU Scheduling is a process that allows one process to use the CPU while another process is delayed due to unavailability of any resources such as I / O etc, thus making full use of the CPU. In short, CPU scheduling decides the order and priority of the processes to run and allocates the CPU time based on various parameters such as CPU usage, throughput, turnaround, waiting time, and response time. The purpose of CPU Scheduling is to make the system more efficient, faster, and fairer.

CPU scheduling Criteria

CPU scheduling is essential for the system's performance and ensures that processes are executed correctly and on time. Different CPU scheduling algorithms have other properties and the choice of a particular algorithm depends on various factors. Many criteria have been suggested for comparing CPU scheduling algorithms.

CPU Scheduling has several criteria. Some of them are mentioned below.

i. CPU utilization
ii. Throughput
iii. Turnaround Time
iv. Waiting Time
v. Response Time
vi. Completion time
vii. Priority
viii. Predictability
ix. **CPU Utilization**

The main objective of any CPU scheduling algorithm is to keep the CPU as busy as possible. Theoretically, CPU utilization can range from 0 to 100 but in a real-time system, it varies from 40 to 90 percent depending on the load upon the system.

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.

Turn Around Time=Completion Time – Arrival Time

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.

Waiting Time= Turnaround Time – Brust Time

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.

Response Time=CPU Allocation Time(when the CPU was allocated for the first)- Arrival 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

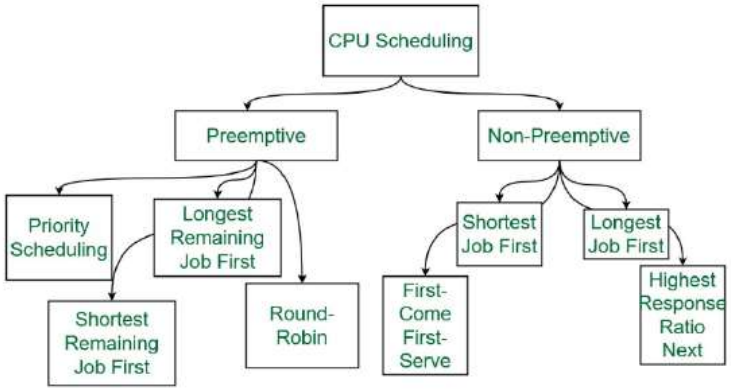
A given process always should run in about the same amount of time under a similar system load.

Process Scheduling Algorithms

Process scheduling algorithms are essential components of operating systems that determine the order in which processes are executed by the CPU. These algorithms aim to maximize system throughput, minimize response time, and ensure fairness among processes.

There are mainly two types of scheduling methods:

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



Objectives and Goals of Process Scheduling Algorithm

- Utilization of CPU at maximum level. **Keep CPU as busy as possible**.
- **Allocation of CPU should be fair**.
- **Throughput should be Maximum**. i.e. Number of processes that complete their execution per time unit should be maximized.
- **Minimum turnaround time**, i.e. time taken by a process to finish execution should be the least.
- There should be a **minimum waiting time** and the process should not starve in the ready queue.
- **Minimum response time.** It means that the time when a process produces the first response should be as less as possible.

i. Batch System: FIFO,SJN, SRTN

This is a general technique for executing programs or tasks automatically in groups. Users submit their jobs (which can be programs, scripts, or commands) to the operating system, and the system queues and processes them efficiently, one after another. This is ideal for repetitive tasks that deal with large volumes of data.

Algorithms that executed on the basis of batch system are listed below:

a. First In First Out (FIFO)
b. Shortest Job First (SJF) or Shortest Job Next (SJN)
c. Shortest Time Remaining Next (SRTN)

a. **First in First Out (FIFO)**

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.

Characteristics of FCFS

- FCFS supports non-preemptive and preemptive CPU scheduling algorithms.
- Tasks are always executed on a First-come, First-serve concept.
- FCFS is easy to implement and use.
- This algorithm is not much efficient in performance and wait time is high.

. Advantages

- Easy to implement.
- First come, first serve method.

Disadvantages

- FCFS suffers from Convoy effect. (if the burst time of the first job entering the ready queue of the CPU is the highest of all)
- The average waiting time is much higher than the other algorithms.
- FCFS is very simple and easy to implement and hence not much efficient.

b. Shortest Job First (SJF) or Shortest Job Next (SJN)

- This is also known as **shortest job first**, or SJF
- This is a non-preemptive, pre-emptive scheduling algorithm.
- Best approach to minimize waiting time.
- Easy to implement in Batch systems where required CPU time is known in advance.
- Impossible to implement in interactive systems where required CPU time is not known.
- The processer should know in advance how much time process will take.

Advantage

i. Reduce Average Waiting Time
ii. Improved Job Throughput
iii. Ideal for Batch Processing

Disadvantage

i. Starvation: If a consent stream of short job arrives, long jobs can be indefinitely pushed back, potentially never getting completed.
ii. Impractical for short term scheduling

iii. Overhead for maintaining information.

c. Shortest Remaining Job First (SRTN)

This Algorithm is the **preemptive version** of **SJF scheduling**. In SRTF, the execution of the process can be stopped after certain amount of time. At the arrival of every process, the short term scheduler schedules the process with the least remaining burst time among the list of available processes and the running process.

Once all the processes are available in the **ready queue**, No preemption will be done and the algorithm will work as **SJF scheduling**. The context of the process is saved in the **Process Control Block** when the process is removed from the execution and the next process is scheduled. This PCB is accessed on the **next execution** of this process.

Advantages

- **Fast processing of short jobs:** SRTN prioritizes processes with the shortest remaining time, ensuring quick completion of short jobs. This improves the overall system responsiveness for users who submit short tasks.
- **Reduced waiting time:** Since shorter jobs are completed faster, the waiting time for other processes in the queue is generally lower compared to non-preemptive algorithms.
- **Fairness for short processes:** SRTN prevents short processes from being indefinitely stuck behind long processes, ensuring fairer CPU allocation.

Disadvantages

- **Complex Implementation**: SRTN requires constant monitoring of remaining execution times for all processes. This can lead to increased overhead compared to simpler scheduling algorithms.
- **Starvation for long processes**: Continuously prioritizing short jobs can lead to indefinite starvation for a long process. If a constant stream of short jobs enters the system, long processes might never get a chance to run.
- **Not Ideal for real-world systems**: The frequent context switching between processes due to preemption can be inefficient for some hardware architecture.
- **Difficult to predict performance**: Due to the dynamic nature of process arrival times and burst times, predicating the average waiting time and turnaround time for processes can be challenging with SRTN.

d. Round Robbin Scheduling Algorithm

Round Robin is a CPU scheduling algorithm where each process is cyclically assigned a fixed time slot. It is the preemptive version of the First come First Serve CPU Scheduling algorithm.

- Round Robin CPU Algorithm generally focuses on Time Sharing technique.
- The period of time for which a process or job is allowed to run in a pre-emptive method is called time **quantum**.
- Each process or job present in the ready queue is assigned the CPU for that time quantum, if the execution of the process is completed during that time then the process will **end** else the process will go back to the **waiting table** and wait for its next turn to complete the execution.

Characteristics/Features

- It is simple, easy to implement, and starvation-free as all processes get a fair share of CPU.
- One of the most commonly used techniques in CPU scheduling is a core.
- It is preemptive as processes are assigned CPU only for a fixed slice of time at most.
- The disadvantage of it is more overhead of context switching.

Advantages

- There is fairness since every process gets an equal share of the CPU.
- The newly created process is added to the end of the ready queue.
- A round-robin scheduler generally employs time-sharing, giving each job a time slot or quantum.
- While performing a round-robin scheduling, a particular time quantum is allotted to different jobs.
- Each process get a chance to reschedule after a particular quantum time in this scheduling.

Disadvantages

- There is Larger waiting time and Response time.

- There is Low throughput.
- There is Context Switches.
- Gantt chart seems to come too big (if quantum time is less for scheduling. For Example:1 ms for big scheduling.)
- Time consuming scheduling for small quantum.

e. Highest Response Ratio Next (HRRN)

HRRN is basically the modification of Shortest Job First(SJF) in order to reduce the problem of starvation.

In HRRN, CPU is assigned to the next process that has the highest response ratio and not the process having less burst time.

Criteria: Response Ratio (RR)

Mode: Non Preemptive

Response Ratio = (w+s)/s

Where, w= waiting for a process so far

s=Service time or burst time

Characteristics

- Highest Response Ratio Next is a non-preemptive CPU Scheduling algorithm and it is considered as one of the most optimal scheduling algorithms.
- The criteria for HRRN is Response Ratio, and the mode is Non-Preemptive.
- HRRN is considered as the modification of the Shortest Job First to reduce the problem of starvation.
- In comparison with SJF, during the HRRN scheduling algorithm, the CPU is allotted to the next process which has the highest response ratio, and not to the process having less burst time.

Advantages

- Dynamic Priority Assignment
- Avoidance of starvation
- Fairness

Disadvantages

- Complexity
- Subject to convoy effect
- Potentially inefficient for batch processing
- Sensitivity to parameters

Critical Section Problem

When more than one processes try to access the same code segment that segment is known as the critical section. The critical section contains shared variables or resources that need to be synchronized to maintain the consistency of data variables. The critical section problem is to design a protocol that the processes can use to cooperate.

Consider a system consists of processes (P0,P1,.....Pn) and each process has a segment of code, called a critical section in which the process may be changing common variables, updating a table, writing a file, and

so on. When a process is executing in its critical section, no other process is allowed to execute in its critical section. That is , no two processes are executing in their critical sections at the same time.

Rules that needs to followed by processes while accessing critical section

i. Each process must request permission to enter its critical section

ii. The section of code implementing this request is the entry section.

iii. The critical section may ne followed by an exit section

iv. The remaining code is the remainder section.

Code

```
do{
Entry section
Critical section
Exit section
Remainder section
}while (TRUE);
```

Figure: General Structure of a typical process

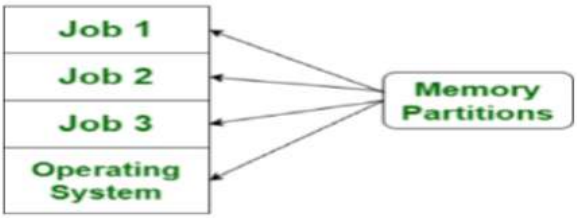
Unit 3

Memory Management

Concept of Multiprogramming

An operating system that allows multiple program to run simultaneously on a single processor machine is known as a multiprogramming operating system. The other program are prepared to use the CPU while one program waits for an input/output transfer. It is an extension of batch processing OS. It is non-preemptive type of OS.

Multiprogramming



Advantages

- It provides less response time.
- It may help to run various jobs in a single application simultaneously.
- It helps to optimize the total job throughput of the computer.
- Various users may use the multiprogramming system at once.
- Short-time jobs are done quickly in comparison to long-time jobs.
- It may help to improve turnaround time for short-time tasks.
- It helps in improving CPU utilization and never gets idle.
- The resources are utilized smartly.

Disadvantages

- It is highly complicated and sophisticated.
- The CPU scheduling is required.
- Memory management is needed in the operating system because all types of tasks are stored in the main memory.
- The harder task is to handle all processes and tasks.
- If it has a large number of jobs, then long-term jobs will require a long wait.

Memory Management

In a multiprogramming computer, the Operating System resides in a part of 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.

Memory management function

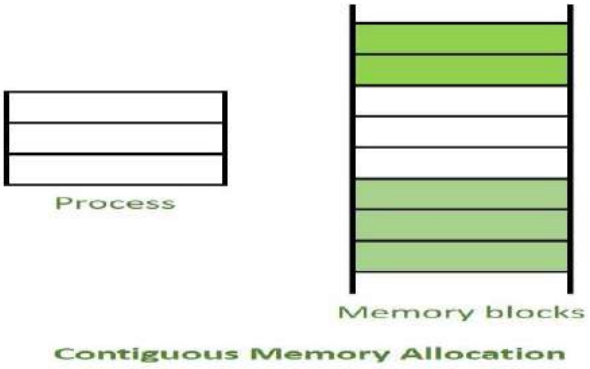
Memory management functions are essential components of an operating system responsible for handling memory allocation and deallocation. These functions ensure that programs have access to the memory they need while preventing conflicts and resource wastage. Here are some common memory management functions:

- **Memory Allocation**: This function assigns portions of memory to processes or programs that request it. It involves finding an appropriate block of memory of the required size and marking it as allocated, ensuring that it is not assigned to another process.
- **Memory Deallocation**: Also known as memory freeing, this function releases memory that is no longer needed by a process back to the system. It involves marking the allocated memory block as available for reuse.
- **Memory Mapping**: Memory mapping allows processes to access files and devices as if they were accessing memory. It involves creating a mapping between a portion of the process's address space and the file or device, allowing the process to read from or write to it using standard memory access operations.
- **Memory Protection**: Memory protection mechanisms ensure that processes cannot access memory locations that they are not authorized to access. This prevents processes from accidentally or maliciously modifying the memory of other processes or the operating system itself.
- **Virtual Memory Management**: Virtual memory allows processes to use more memory than is physically available by using a combination of RAM and disk storage. Memory management functions handle mapping portions of a process's virtual address space to physical memory or disk storage, as well as managing page faults when a required memory page is not currently in physical memory.
- **Memory Fragmentation Handling**: Memory fragmentation occurs when free memory is divided into small, non-contiguous blocks, making it difficult to allocate large contiguous blocks of memory. Memory management functions may include techniques such as compaction or memory pooling to reduce fragmentation and improve memory utilization.
- **Memory Sharing**: Memory management functions may facilitate memory sharing between processes, allowing multiple processes to access the same portion of memory. This can be useful for inter-process communication and sharing of data structures.
- **Memory Optimization**: Memory management functions may include optimization techniques such as memory caching and prefetching to improve performance by reducing memory access latency and increasing the efficiency of memory usage.

Contiguous Memory Allocation

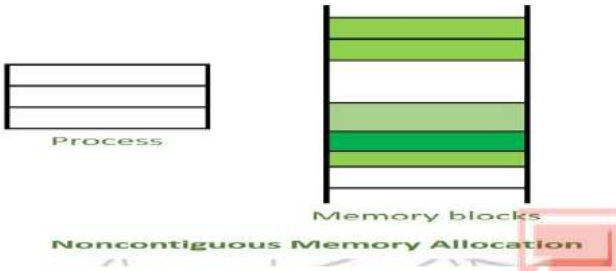
Contiguous Memory Allocation : Contiguous memory allocation is basically a method in which a single contiguous section/part of memory is allocated to a process or file needing it. Because of this all the available memory space resides at the same place together, which means that the freely/unused available memory partitions are not distributed in a random fashion here and there across the whole memory space.

The main memory is a combination of two main portions- one for the operating system and other for the user program. We can implement/achieve contiguous memory allocation by dividing the memory partitions into fixed size partitions.



Non-contiguous Memory allocation

Non-Contiguous Memory Allocation : Non-Contiguous memory allocation is basically a method on the contrary to contiguous allocation method, allocates the memory space present in different locations to the process as per it's requirements. As all the available memory space is in a distributed pattern so the freely available memory space is also scattered here and there. This technique of memory allocation helps to reduce the wastage of memory, which eventually gives rise to Internal and external fragmentation.



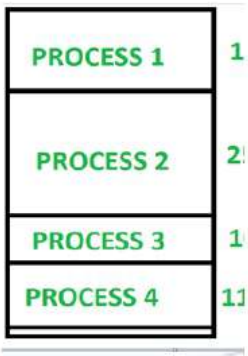
S.N	Contiguous Memory Allocation	Non-Contiguous Memory Allocation
1	Contiguous memory allocation allocates consecutive blocks of memory to a file/process.	Non-Contiguous memory allocation allocates separate blocks of memory to a file/process.
2	Faster in Execution	Slower in Execution.
3	It is easier for the OS to control.	It is difficult for the OS to control.
4	Overhead is minimum as not much address translations are there while executing a process.	More Overheads are there as there are more address translations.
5	Both Internal fragmentation and external fragmentation occurs in Contiguous memory allocation method.	Only External fragmentation occurs in Non-Contiguous memory allocation method.
6	It includes single partition allocation and multi-partition allocation.	It includes paging and segmentation.
7	Wastage of memory is there.	No memory wastage is there.

Multiprogramming with fixed Partition

Multi-programming with fixed partitioning is a contiguous memory management technique in which the main memory is divided into fixed sized partitions which can be of equal or unequal size. Whenever we have to allocate a process memory then a free partition that is big enough to hold the process is found. Then the memory is allocated to the process. If there is no free space available then the process waits in the queue to be allocated memory. It is one of the most oldest memory management technique which is easy to implement.

Multiprogramming with variable partition

Multi-programming with variable partitioning is a contiguous memory management technique in which the main memory is not divided into partitions and the process is allocated a chunk of free memory that is big enough for it to fit. The space which is left is considered as the free space which can be further used by other processes. It also provides the concept of compaction. In compaction the spaces that are free and the spaces which not allocated to the process are combined and single large memory space is made.



S.N	Multiprogramming with fixed partition	Multiprogramming with variable partition
1	In multi-programming with fixed partitioning the main memory is divided into fixed sized partitions.	In multi-programming with variable partitioning the main memory is not divided into fixed sized partitions.
2	Only one process can be placed in a partition.	In variable partitioning, the process is allocated a chunk of free memory.
3	It does not utilize the main memory effectively.	It utilizes the main memory effectively.
4	There is presence of internal fragmentation and external fragmentation	There is external fragmentation.
5	Degree of multi-programming is less.	Degree of multi-programming is higher.
6	It is more easier to implement.	It is less easier to implement.
7	There is limitation on size of process.	There is no limitation on size of process.

S.N	Internal Fragmentation	External Fragmentation
1	In internal fragmentation fixed-sized memory, blocks square measure appointed to process.	In external fragmentation, variable-sized memory blocks square measure appointed to the method.
2	Internal fragmentation happens when the method or process is smaller than the memory.	External fragmentation happens when the method or process is removed.
3	The solution of internal fragmentation is the best-fit block.	The solution to external fragmentation is compaction and paging.
4	Internal fragmentation occurs when memory is divided into fixed-sized partitions.	External fragmentation occurs when memory is divided into variable size partitions based on the size of processes.
5	The difference between memory allocated and required space or memory is called Internal fragmentation.	The unused spaces formed between non-contiguous memory fragments are too small to serve a new process, which is called External fragmentation.
6	Internal fragmentation occurs with paging and fixed partitioning.	External fragmentation occurs with segmentation and dynamic partitioning.
7	It occurs in worst fit memory allocation method.	It occurs in best fit and first fit memory allocation method.

Memory Allocation Algorithm

Memory allocation refers to the process of assigning portions of a computer's memory (RAM) to various programs, processes, or data structures during their execution. Memory allocation algorithm can be categorized into four types. They are:

i. First Fit Algorithm
ii. Next Fit Algorithm
iii. Best Fit Algorithm and
iv. Worst Fit Algorithm

a. First Fit Algorithm

- In the First Fit algorithm, the system allocates the first available memory block that is large enough to accommodate the incoming process or data.
- The search for the available block starts from the beginning of the memory and continues until a suitable block is found.
- It's a simple and fast algorithm since it stops searching as soon as a suitable block is located.
- However, it can lead to fragmentation issues, as it may leave small unused gaps between allocated blocks.

b. Next Fit Algorithm

- Next Fit is similar to First Fit, but the search for available memory starts from the location where the last allocation was made.
- After finding a suitable block, the system continues searching from that point onwards, wrapping around to the beginning of the memory if necessary.
- Next Fit reduces search time compared to First Fit, especially if the next allocation request is close to the previous one.

- However, it still suffers from fragmentation issues and may not fully utilize the available memory space.

c. Best Fit Algorithm

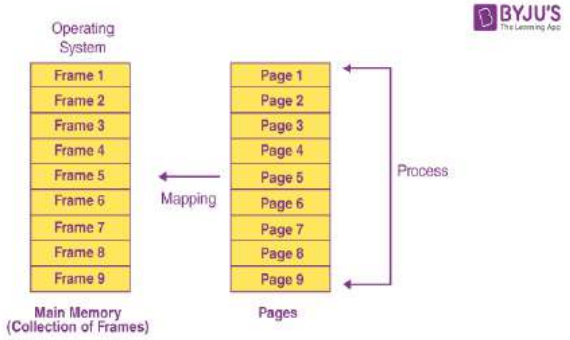
- The Best Fit algorithm allocates the smallest available block of memory that is large enough to accommodate the incoming process or data.
- It involves searching the entire memory space to find the block that minimizes wastage, i.e., the block with the smallest leftover space after allocation.
- Best Fit aims to minimize fragmentation by utilizing memory space more efficiently.
- However, it can be slower and more computationally intensive than First Fit or Next Fit due to the need to search for the best-fitting block.

d. Worst Fit Algorithm

- In contrast to Best Fit, the Worst Fit algorithm allocates the largest available block of memory to the incoming process or data.
- This results in the largest leftover fragment after allocation, potentially leading to more fragmentation over time.
- Worst Fit is generally less efficient than Best Fit in terms of memory utilization since it tends to leave larger unused spaces between allocated blocks.
- However, it can be simpler to implement compared to Best Fit.

Concept of Paging

- Paging is a storage mechanism used in OS to retrieve processes from secondary storage to the main memory as pages. The primary concept behind paging is to break each process into individual pages. Thus the primary memory would also be separated into frames.
- One page of the process must be saved in one of the given memory frames. These pages can be stored in various memory locations, but finding contiguous frames/holes is always the main goal. Process pages are usually only brought into the main memory when they are needed; else, they are stored in the secondary storage.
- The frame sizes may vary depending on the OS. Each frame must be of the same size. Since the pages present in paging are mapped on to the frames, the page size should be similar to the frame size.



Concept of page fault

A page fault in an operating system occurs when a program accesses a memory page that is not currently present in physical memory (RAM) and needs to be retrieved from secondary storage (like a hard drive or SSD). This can happen for various reasons:

- **Demand Paging**: Most modern operating systems use demand paging, where only the pages of memory needed by a program are loaded into RAM. When a program accesses a page that is not currently in RAM, a page fault occurs, and the operating system must bring the required page from secondary storage into RAM.
- **Copy-on-Write**: When multiple processes are sharing the same memory space (as in forked processes), the operating system may use a copy-on-write mechanism. This means that initially, the pages are shared among the processes, but if one process tries to write to a shared page, the OS creates a copy of that page for the process, avoiding interference with other processes. This also can lead to page faults.
- **Memory Swapping**: In systems with limited physical memory, the operating system may need to swap out entire pages of memory to disk to make room for others. When a process that has been swapped out is resumed, page faults occur as the OS fetches the necessary pages back into RAM.
- **Page Replacement**: If physical memory is full, the operating system needs to select pages to evict to make space for incoming pages. This involves a page replacement algorithm (like LRU - Least Recently Used or FIFO - First In First Out). When a page selected for replacement is needed later, it results in a page fault.

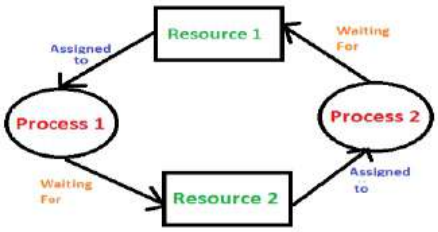
When a page fault occurs, the operating system typically handles it by interrupting the executing process, finding the required page on disk, loading it into RAM, updating page tables, and then allowing the process to continue execution.

Unit 4
Deadlock Management

Deadlock Concept

*A **deadlock*** is a situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.

A deadlock in an operating system occurs when two or more processes are unable to proceed because each is waiting for the other to release a resource they need. This creates a cyclic dependency, where each process is holding a resource that the other process requires to complete. As a result, neither process can move forward, and the system becomes "deadlocked."



Deadlock Condition

Deadlocks typically involve four conditions, known as the Coffman conditions:

1. Mutual Exclusion: At least one resource must be held in a non-sharable mode, meaning only one process can use it at a time.
2. Hold and Wait: Processes already holding resources may request additional resources while still holding the current resources.
3. No Preemption: Resources cannot be forcibly taken away from a process; they must be released voluntarily.
4. Circular Wait: A set of waiting processes must exist such that each process in the set is waiting for a resource held by another process in the set.

Deadlock handling Strategies

Deadlock handling strategies in operating systems aim to prevent, detect, and recover from deadlocks efficiently. Here are some common deadlock handling strategies:

i Deadlock Prevention

ii. Deadlock detection

iii. Deadlock Avoidance

iv. Recovery from Deadlock

a. Deadlock Prevention

The idea is to not let the system into a deadlock state. This system will make sure that above mentioned four conditions will not arise. These techniques are very costly so we use this in cases where our priority is making a system deadlock-free.
One can zoom into each category individually, Prevention is done by negating one of the above-mentioned necessary conditions for deadlock. Prevention can be done in four different ways:

i. Mutual Exclusion Avoidance

ii. Hold and Wait Avoidance

iii. No Preemption

iv. Circular Wait Avoidance

- **Mutual Exclusion Avoidance:** Design resource types that can be shared rather than exclusively owned. For example, read-only resources can be shared among processes without causing deadlocks.
- **Hold and Wait Avoidance:** Require processes to request all necessary resources at once or release held resources before requesting new ones. This ensures that processes don't hold resources while waiting for others, reducing the chance of deadlock.
- **No Preemption:** Allow preemptive release of resources from processes if necessary. However, this can lead to starvation and reduced system performance.
- **Circular Wait Avoidance:** Impose a total ordering on resource types and require processes to request resources in increasing order. This prevents circular wait conditions from occurring.

b. Deadlock detection

If Deadlock prevention or avoidance is not applied to the software then we can handle this by deadlock detection and recovery. which consist of two phases:

- In the first phase, we examine the state of the process and check whether there is a deadlock or not in the system.
- If found deadlock in the first phase then we apply the algorithm for recovery of the deadlock.

In Deadlock detection and recovery, we get the correctness of data but performance decreases.

- **Deadlock Detection:** Periodically check for the existence of deadlocks in the system using algorithms like the Banker's algorithm or resource allocation graphs.
- **Resource Allocation Graphs (RAG):** Maintain a graph where processes are nodes and resource allocations are edges. Deadlocks can be detected by finding cycles in this graph.
- **Timeouts:** Set timeouts for processes waiting on resources. If a process waits too long, it can be aborted, releasing its held resources and potentially breaking deadlocks.
- **Killing Processes:** If a deadlock is detected, one or more processes involved in the deadlock can be terminated to release their resources and resolve the deadlock. The choice of which processes to terminate should be carefully considered to minimize disruption.
- **Rollback:** Roll back the state of the system to a previously consistent state and restart processes from that point. This can be complex and may not always be feasible depending on the application.

c. Deadlock Avoidance

It is another method for dealing with a deadlock. The OS verifies/examines the system state in this manner, which means it checks whether the system is in a safe or hazardous condition at each step. This procedure is repeated until the system is in safe mode. In the event that the system becomes dangerous, the operating system will take a step backwards.

In simpler words, the OS verifies each allocation to ensure that the system does not experience a deadlock. It is preferable to avoid a given deadlock than to take steps once one has occurred. Here, the wait for graph could be utilized to prevent deadlock. Deadlock avoidance is only effective for smaller DBs since larger databases can get rather complex.

It necessitates an understanding of future process requests. There are two methods for avoiding deadlock:

- Denial of process initiation
- Denial of resource allocation

d. Recovery from deadlock

Deadlock recovery is a process in computer science and operating systems that aims to resolve or mitigate the effects of a deadlock after it has been detected. Deadlocks are situations where multiple processes are stuck and unable to proceed because each process is waiting for a resource held by another process. Recovery strategies are designed to break this deadlock and allow the system to continue functioning.

Recovery Strategies:

- **Process-Termination:**
  One way to recover from a deadlock is to terminate one or more of the processes involved in the deadlock. By releasing the resources held by these terminated processes, the remaining processes may be able to continue executing. However, this approach should be used cautiously, as terminating processes could lead to loss of data or incomplete transactions.
- **Resource-Preemption:**
  Resources can be forcibly taken away from one or more processes and allocated to the waiting processes. This approach can break the circular wait condition and allow the system to proceed. However, resource preemption can be complex and needs careful consideration to avoid disrupting the execution of processes.
- **Process-Rollback:**
  In situations where processes have checkpoints or states saved at various intervals, a process can be rolled back to a previously saved state. This means that the process will release all the resources acquired after the saved state, which can then be allocated to other waiting processes. Rollback, though, can be resource-intensive and may not be feasible for all types of applications.
- **Wait-Die-and-Wound-Wait-Schemes:**
  As mentioned in the Deadlock Detection in OS section, these schemes can also be used for recovery. Older processes can preempt resources from **younger processes** (Wound-Wait), or younger processes can be terminated if they try to access resources held by **older processes** (Wait-Die).
- **Kill-the-Deadlock:**
  In some cases, it might be possible to identify a specific process that is causing the deadlock and terminate it. This is typically a last resort option, as it directly terminates a process without any complex recovery mechanisms.

Banker's Algorithm

The Banker's algorithm is a resource allocation and deadlock avoidance algorithm used in operating systems, particularly in multitasking environments. It was developed by Edsger Dijkstra in 1965.

The algorithm ensures that the system will not enter a deadlock state by checking for safety before granting resource allocation requests from processes. It works by keeping track of the available resources and the maximum resources each process can request. Before allocating resources to a process, the Banker's algorithm simulates the allocation and checks if it could lead to a safe state, meaning that there's a sequence in which all processes can complete their execution without getting stuck in a deadlock.

Here are the basic steps of the Banker's algorithm:

1. **Initialization**: When the system starts up or when a new process enters the system, the algorithm initializes several data structures to keep track of the resources.
2. **Requesting Resources**: When a process requests resources, the system checks if granting those resources would leave the system in a safe state. If the resources can be allocated without causing a deadlock, the request is granted. Otherwise, the process is forced to wait until it can be safely allocated the requested resources.
3. **Releasing Resources**: When a process releases resources, they are added back to the available pool of resources. This may allow other processes to acquire them.
4. **Safety Check**: Periodically, or whenever a resource allocation request is made, the system checks if the current allocation state is safe. This involves simulating the allocation of resources to all processes and ensuring that there is at least one safe sequence in which all processes can complete. If such a sequence exists, the allocation is considered safe, and the request can be granted.

Unit 5

File and Input/Output Management

File Naming

A filename or file name is a name used to uniquely identify a computer file in a file system. Different file system impose different restrictions on filename lengths.

A filename may(depending on the file system) include:

i. **name**- base name of the file

ii. **Extension**- may indicate the format of the file (e.g. .txt for plain text, .pdf for Portable Document Format, .dat for unspecified binary data etc.)

File Structure

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, Specially, it's to minimize the number of trips to the disk to get the data you're looking for.

File Types

A file has a certain defined structure according to its types:

i. Text File: Sequence of structure organized into lines.

ii. Source File: Sequence of subroutine & functions written in programming language.

iii. Object File: Sequence of bytes organized into blocks understandable by the system linker.

iv. Executable file: It contains code that is used by the loader to execute the program.

File Type	Usual Extension	Functions
Executable	Exe, com, bin	Read to run machine language program
Object	Obj, o, class	Compiled by machine language not linked
Source code	C, Java, html, CPP	Source codes in various language
Txt	Txt, doc	It contains textual data and document
Word Processor	Wp, rrf, .doc	Various word processor formats
Library	Lib, a, so, dll, mpeg, mov, lm	It contains libraires of routine for programmers.

File Access

When a file is used, information is read and accessed into computer memory and there are several ways to access this information of the file. Some systems provide only one access method for files. Other systems, such as those of IBM, support many access methods, and choosing the right one for a particular application is a major design problem.

There are three ways to access a file into a computer system: Sequential-Access, Direct Access, Index sequential Method.

1. Sequential Access

It is the simplest access method. Information in the file is processed in order, one record after the other. This mode of access is by far the most common; for example, editor and compiler usually access the file in this fashion.

- Data is accessed one record right after another record in an order.
- When we use read command, it move ahead pointer by one
- When we use write command, it will allocate memory and move the pointer to the end of the file
- Such a method is reasonable for tape.

Advantages

- it is simple to implement this file access mechanism.
- It uses lexicographic order to quickly access the next entry.
- It is suitable for applications that require access to all records in a file, in a specific order.
- It is less prone to data corruption as the data is written sequentially and not randomly.

Disadvantages

- If the file record that needs to be accessed next is not present next to the current record, this type of file access method is slow.
- Moving a sizable chunk of the file may be necessary to insert a new record.
- It does not allow for quick access to specific records in the file. The entire file must be searched sequentially to find a specific record, which can be time-consuming.
- It is not well-suited for applications that require frequent updates or modifications to the file. Updating or inserting a record in the middle of a large file can be a slow and cumbersome process.

2. Direct Access

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.

Advantages

- The files can be immediately accessed decreasing the average access time.
- In the direct access method, in order to access a block, there is no need of traversing all the blocks present before it.

Disadvantages

- Direct access file does not provide backup facility.
- It is expensive.
- It has less storage space as compared to sequential file.

3. 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.

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

Advantages

- It provides quick access for sequential and direct processing.
- It reduces the degree of the sequential search

Disadvantages

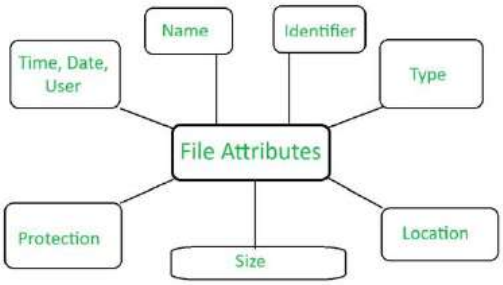
- Indexed sequential access file requires unique keys and periodic reorganization.
- Indexed sequential access file takes longer time to search the index for the data access or retrieval.

File attributes

File attributes are type of meta-data that describe and may modify how files and/or directories in a filesystem behave. In other words, it provides an extra information about file itself. Typical file attributes may, for example, indicate or specify whether a file is visible, modifiable, compressed or encrypted.

Following are some common file attributes:

- **Name:** File name is the name given to the file. A name is usually a string of characters.
- **Identifier:** Identifier is a unique number for a file. It identifies files within the file system. It is not readable to us, unlike file names.
- **Type:** Type is another attribute of a file which specifies the type of file such as archive file (.zip), source code file (.c, .java), .docx file, .txt file, etc.
- **Location:** Specifies the location of the file on the device (The directory path). This attribute is a pointer to a device.
- **Size:** Specifies the current size of the file (in Kb, Mb, Gb, etc.) and possibly the maximum allowed size of the file.
- **Protection:** Specifies information about Access control (Permissions about Who can read, edit, write, and execute the file.) It provides security to sensitive and private information.
- **Time, date, and user identification:** This information tells us about the date and time on which the file was created, last modified, created and modified by which user, etc.



File Operation

A file is a collection of logically related data that is recorded on the secondary storage in the form of sequence of operations. The content of the files are defined by its creator who is creating the file. The various operations which can be implemented on a file such as read, write, open and close etc. are called file operations. These operations are performed by the user by using the commands provided by the operating system. Some common operations are as follows:

i. Create operations
ii. Open Operation
iii. Write Operation
iv. Read Operation
v. Re-position or seek Operation
vi. Delete operation
vii. Truncate operation
viii. Close Operation
ix. Append Operation
x. Rename Operation

Create operation

This operation is used to create a file in the file system. It is the most widely used operation performed on the file system. To create a new file of a particular type the associated application program calls the file system. This file system allocates space to the file. As the file system knows the format of directory structure, so entry of this new file is made into the appropriate directory.

Open Operation

This operation is the common operation performed on the file. Once the file is created, it must be opened before performing the file processing operations. When the user wants to open a file, it provides a file name to open the particular file in the file system. It tells the operating system to invoke the open system call and passes the file name to the file system.

Write operation:

This operation is used to write the information into a file. A system call write is issued that specifies the name of the file and the length of the data has to be written to the file. Whenever the file length is increased by specified value and the file pointer is repositioned after the last byte written.

Read operation:

This operation reads the contents from a file. A Read pointer is maintained by the OS, pointing to the position up to which the data has been read.

Re-position or Seek Operation

The seek system call re-positions the file pointers from the current position to a specific place in the file i.e. forward or backward depending upon the user's requirement. This operation is generally performed with those file management systems that support direct access files.

Delete operation

Deleting the file will not only delete all the data stored inside the file it is also used so that disk space occupied by it is freed. In order to delete the specified file the directory is searched. When the directory entry is located, all the associated file space and the directory entry is released.

Truncate operation

Truncating is simply deleting the file except deleting attributes. The file is not completely deleted although the information stored inside the file gets replaced.

Close operation

When the processing of the file is complete, it should be closed so that all the changes made permanent and all the resources occupied should be released. On closing it deallocates all the internal descriptors that were created when the file was opened.

Append operation

This operation adds data to the end of the file.

Rename-operation
This operation is used to rename the existing file.

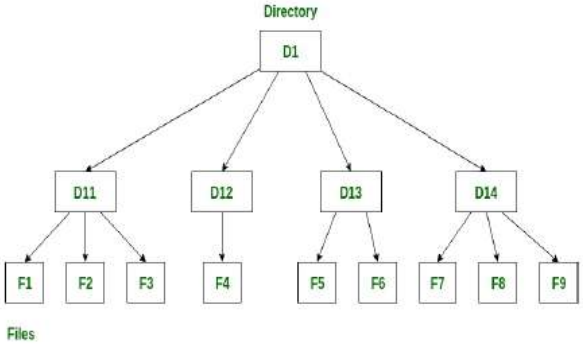
File Directory System

A **directory** is a container that is used to contain folders and files. It organizes files and folders in a hierarchical manner.There are mainly three types of file directories in OS. They are:

i. Single Level Directory

ii. Two Level Directory

iii. Tree Structured Directory



i. Single level Directory

All the files belong to a single directory

Advantages

i. Easy to implement

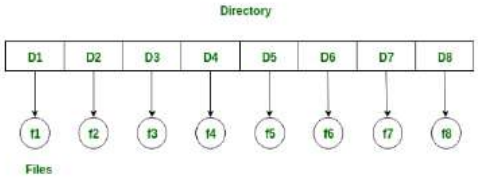
ii. Reduced Redundancy

Disadvantages

i. All files belong to one same directory

ii. User can't have 2 files with the same names

iii. User doesn't have option to group file according to his/her needs.



ii. Two level Directory

In this type of directory each user has its own directory.

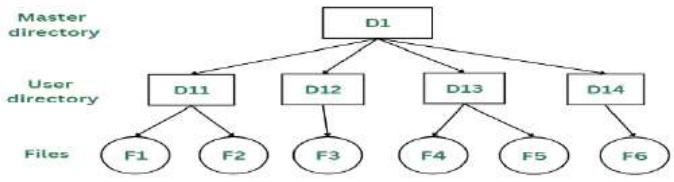
In the two-level directory structure, each user has his own user file directory (UFD). The UFDs have similar structures, but each lists only the files of a single user. When a user job starts or a user logs in, the system's master file directory (MFD) is searched. The MFD is indexed by user name or account number, and each entry points to the UFD for that user.

Advantages

- Sharing Directory by user problem is solved
- Efficiently
- More secure then 1 level directory
- Two users can have file with same name in their own directories.

Disadvantages

- Grouping problem is still not solved, for example if user wants to create a directory group for all files that are movies or music., he can't do that
- No two files for a single user can have same names.
- Due to two level there is a path name for every file to locate that file.



iii. Tree Structure Directory

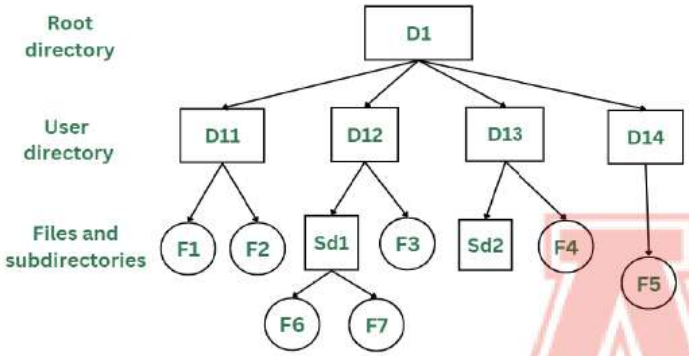
Directory is maintained in the form of a tree. Searching is efficient and also there is grouping capability. We have absolute or relative path name for a file.

Advantages

- Two files can have same names now, if they are in different sub directories.
- Grouping is also possible now.

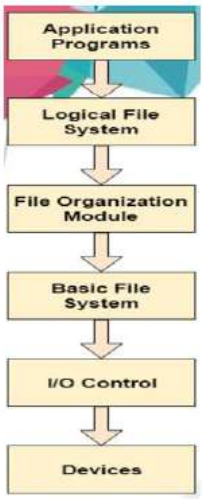
Disadvantages

- As the user isn't allowed to access other user's directory, this prevents the file sharing among users.
- As the user has the capability to make subdirectories, if the number of subdirectories increase the searching may become complicated.
- Users cannot modify the root directory data.
- If files do not fit in one, they might have to be fit into other directories.



File System Layout

- File System provide efficient access to the disk by allowing data to be stored, located and retrieved in a convenient way. A file System must be able to store the file, locate the file and retrieve the file.
- Most of the Operating Systems use layering approach for every task including file systems. Every layer of the file system is responsible for some activities.
- The image shown below, elaborates how the file system is divided in different layers, and also the functionality of each layer.



- When an application program asks for a file, the first request is directed to the logical file system. The logical file system contains the Meta data of the file and directory structure. If the application program doesn't have the required permissions of the file then this layer will throw an error. Logical file systems also verify the path to the file.
- Generally, files are divided into various logical blocks. Files are to be stored in the hard disk and to be retrieved from the hard disk. Hard disk is divided into various tracks and sectors. Therefore, in order to store and retrieve the files, the logical blocks need to be mapped to physical blocks. This mapping is done by File organization module. It is also responsible for free space management.
- Once File organization module decided which physical block the application program needs, it passes this information to basic file system. The basic file system is responsible for issuing the commands to I/O control in order to fetch those blocks.
- I/O controls contain the codes by using which it can access hard disk. These codes are known as device drivers. I/O controls are also responsible for handling interrupts.

File allocation methods

The allocation method define how the files are stored in the blocks.

There are three main disk space or file allocation methods

i. Contiguous Allocation
ii. Linked Allocation
iii. Indexed Allocation

The main idea behind these methods is to provide

i. Efficient disk space utilization
ii. Fast access to the file blocks

a. Contiguous allocation

Occupies a contiguous set of blocks on the disk.

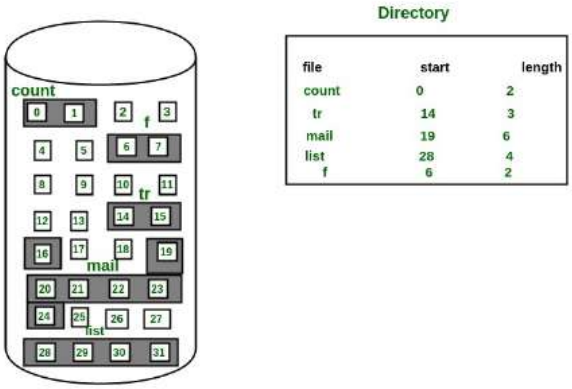
Example

The directory entry for a file with contiguous allocation contains.

a. Address of starting blocks
b. Length of allocated portion

B
B+1
B+2

Example: *file 'mail'* in the following figure starts from the block 19 with length = 6 blocks. Therefore, it occupies *19, 20, 21, 22, 23,24* blocks.



Advantages

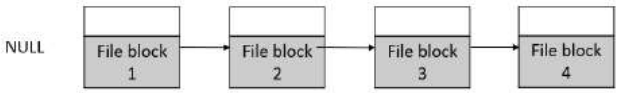
- ❖ It is simple to implement
- ❖ We will get Excellent read performance.
- ❖ Supports Random Access into files

Disadvantages

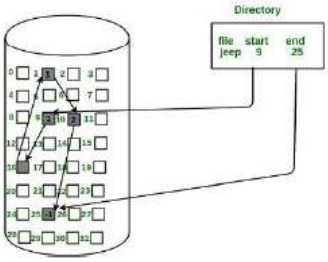
- ❖ The disk will be fragmented
- ❖ Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

b. Linked list allocation

- ❖ Need not to be contiguous.
- ❖ The directory contains a pointer to the starting and 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

i. Very flexible in terms of field size.

ii. File size can be increased easily.

iii. This method does not suffer form external fragmentation.

Disadvantages

i. Internal fragmentation

ii. Data searching is complex.

iii. Some space is used to store address of pointer.

c. Indexed Allocation

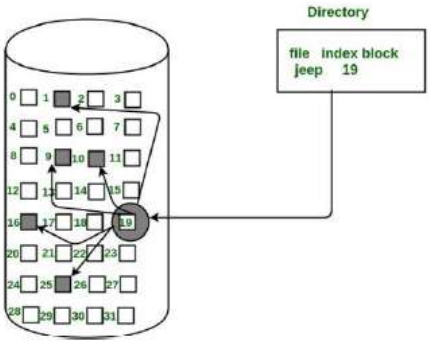
In this scheme, a special block known as the Index block contains the pointer to all the blocks occupied by that file.

Advantages

- It supports direct access.
- It overcomes the problem of external fragmentation.

Disadvantages

- Suffer from wastage of space.



Linked list allocation using table in memory

To address the random access bottleneck, a table in memory (RAM) can be used to store the pointers for each data block of a file. This table is often called a File Allocation Table (FAT).

How It Works

- **File Allocation:** When a file is created or needs to be expanded, the operating system allocates non-contiguous data blocks from the free space on the storage device.
- **Pointer Storage in FAT:** The pointers to these allocated data blocks are then stored in the FAT entry corresponding to that file. The FAT serves as a quick in-memory reference for finding the blocks belonging to the file.
- **Accessing File Data:** When an application needs to access a specific location within the file, it calculates the block index based on the file offset. The operating system then uses the FAT to efficiently locate the pointer to that specific data block.
- **Disk Access:** With the block address obtained from the FAT, the operating system can directly access the desired data block on the storage device.

Example: Consider a file with data blocks allocated at addresses 100, 125, and 130. The FAT entries for this file might look like this:

Block Index	Address in FAT	Pointer Value
0	100	125
1	125	130
2	130	-1 (end of file marker)

Advantages

- **Faster Random Access:** By keeping block pointers readily available in memory, random access to file data becomes significantly faster compared to traditional linked list traversal.
- **Efficient Disk Utilization:** Linked list allocation with a FAT avoids external fragmentation by utilizing scattered free blocks on the storage device.

Disadvantages

- **Internal Fragmentation:** Some storage space within data blocks might be wasted due to unused space at the end of the block, which is known as internal fragmentation.
- **FAT Management Overhead:** Maintaining the FAT in memory adds some overhead to the file system.

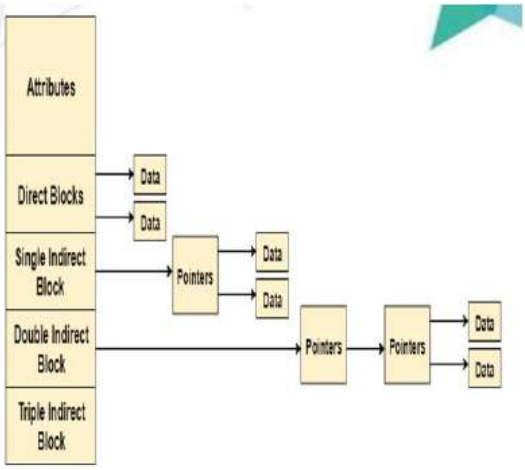
Inodes

In **Unix based** operating system each file is indexed by an **Inode**. Inode are special disk blocks they are created when the file system is created. The number of Inode limits the total number of files/directories that can be stored in the file system.

An Inode is a data structure in UNIX operating system that contains important information pertaining to files within a file system. When a file system is created in UNIX, a set amount of indoes is created as well. Usually, about 1 percent of the file system disk space is allocated to the inode table.

Inodes contains the following informations.

1. Attributes (permissions, time stamp, ownership details, etc) of the file
2. A number of direct blocks which contains the pointers to first 12 blocks of the file.
3. A single indirect pointer which points to an index block. If the file cannot be indexed entirely by the direct blocks then the single indirect pointer is used.
4. A double indirect pointer which points to a disk block that is a collection of the pointers to the disk blocks which are index blocks. Double index pointer is used if the file is too big to be indexed entirely by the direct blocks as well as the single indirect pointer.
5. A triple index pointer that points to a disk block that is a collection of pointers. Each of the pointers is separately pointing to a disk block which also contains a collection of pointers which are separately pointing to an index block that contains the pointers to the file block



Principle of I/O hardware and software

The interplay between I/O (Input/Output) hardware and software is fundamental to how an operating system (OS) interacts with the physical world. Here's a breakdown of the key principles:

- **I/O Hardware:**
- **Components:** These are the physical devices like keyboards, printers, disks, network cards, etc., that allow users to interact with the computer and exchange data.
- **Device Controllers:** Each I/O device has a built-in controller that acts as an intermediary. It manages communication between the device and the computer, often converting raw data streams into a format the computer understands.
- **I/O Ports:** These are special addresses within the computer's memory that devices use to communicate. The controller typically has a unique port for the OS to interact with.
- **Data Transfer Methods:** There are two main methods for transferring data between devices and memory:
    - **Programmed I/O:** The CPU directly controls the transfer, one byte at a time. This is slower but simpler.
    - **Direct Memory Access (DMA):** The device controller transfers data directly to memory without CPU intervention, improving efficiency for high-speed devices.

I/O Software:

- **Device Drivers:** These are software programs that act as translators between the OS and specific I/O devices. They handle device-specific communication protocols and translate generic requests from the OS into commands the device understands. This enables a uniform interface for applications to access various devices.
- **Device Independence:** A core principle is to make programs agnostic to the underlying hardware. A program that reads a file shouldn't care whether it's on a hard drive, USB stick, or network drive. Device drivers provide this abstraction.
- **Buffering:** Due to the speed mismatch between devices and the CPU, the OS often uses buffers. These are temporary storage areas in memory that hold data before it's sent or after it's received. This smooths out data flow and optimizes performance.
- **Caching:** Similar to buffering, the OS might cache frequently accessed data from I/O devices. This allows faster retrieval of the data from memory instead of relying on slower disk access.

- **I/O Subsystems:** The OS often has a dedicated I/O subsystem that manages device drivers, interrupt handling (when devices need attention), scheduling I/O requests, and ensuring data integrity during transfers.

Overall, I/O hardware and software work together to bridge the gap between the physical world and the software realm. By providing a layer of abstraction and managing data flow efficiently, the OS enables users to interact with various devices seamlessly.

Disk formatting

The process of preparing a data storage device such as a hard dism, SSD, memory card or USB flash drive for initial use.

Disk formatting has also the capability to erase the bad applications and various viruses.

There are three level of disk formatting/ part. They are:

i. Low-level Formatting

ii. High-level Formatting

iii. Partitioning

1. **Low-level Formatting**: It is the process of creating actual tracks and sector for storing data on the blank hard-disk.
2. **High-level formatting**: It is the process writing on a file system. It is done to erase the hard-disk and again installing the operating system on the disk-drive.
3. **Partitioning**: Partitioning is the process of dividing the hard-disk into one or more regions.

Disk Arm Scheduling

In an operating system (OS), disk scheduling algorithms play a critical role in optimizing disk access performance. These algorithms dictate the order in which the disk's read/write head services requests scattered across the disk surface. The primary goal is to minimize the total head movement time, also known as seek time, which significantly impacts disk access speed.

- **FCFS (First Come First Served):** This is a straightforward approach where requests are served in the order they arrive. While simple to implement, FCFS can be inefficient, especially if requests are scattered across the disk. Long seeks occur when the head needs to travel large distances to service requests.
- **SSTF (Shortest Seek Time First):** This algorithm prioritizes the request with the shortest seek time from the current head position. It minimizes seek time on average but can lead to starvation for requests further away from the head. Continuously servicing closer requests might indefinitely delay requests on the outer tracks.
- **SCAN (Elevator Algorithm):** This approach mimics an elevator's movement. The head starts at a specific position and moves in one direction, servicing all requests it encounters along the way. Upon reaching the end of the disk, it reverses direction and services requests in the opposite direction. SCAN ensures fairness to all requests but can be slow for requests in the middle if the head is moving away.
- **C-SCAN (Circular SCAN):** Similar to SCAN, the head moves in one direction servicing requests. However, unlike SCAN, it doesn't revisit the traversed area on its return trip. Instead, it jumps to the other end of the disk and starts servicing requests from there, in a circular fashion. C-SCAN avoids

unnecessary head movements but might not be suitable for heavily loaded systems where new requests are constantly arriving.

- **LOOK:** This is a variant of SCAN that only services requests in the current direction until the end is reached. It then reverses direction and services requests encountered on the way back, similar to SCAN. However, LOOK avoids unnecessary movement to the other end of the disk on the return trip, making it potentially faster than SCAN for some request patterns.

The choice of algorithm depends on various factors like workload characteristics, fairness requirements, and access patterns. FCFS is simple but might not be optimal. SSTF reduces seek time but can lead to starvation. SCAN and LOOK offer a balance between seek time and fairness, while C-SCAN is efficient for specific scenario.

Stable Storage

In an operating system, stable storage refers to a mechanism that guarantees data persistence even in the face of hardware or power failures. Unlike main memory, which loses data when the power goes out, stable storage ensures information is written reliably to disk. This is crucial for maintaining system integrity and preventing data loss.

The OS achieves stable storage through various techniques. One approach involves replicating data across multiple storage devices with independent failure modes, like RAID systems. This way, even if one disk fails, the data remains intact on the others. Additionally, the OS might employ write ordering techniques to ensure data updates are completed successfully before the system acknowledges completion. By employing these strategies, the OS provides a foundation for applications to store and retrieve data reliably, safeguarding information even during unexpected situations.

Error handling

- An operating system (OS) relies on robust error handling to maintain stability and a smooth user experience. This involves systematically detecting, responding to, and recovering from errors that occur during various system operations. The OS employs mechanisms like hardware error detection, software exception handling, and system call monitoring to identify issues.
- Once an error is detected, the OS takes action through techniques like exception handling with predefined recovery routines, using error codes to signal the problem's nature, or retrying operations in case of temporary glitches. Error logging also plays a crucial role, providing valuable information for troubleshooting and debugging systemic issues or software bugs.
- Effective error handling safeguards system stability by preventing crashes and data corruption. Users are notified of errors with clear messages, aiding in problem-solving. Additionally, it ensures data integrity and attempts system recovery whenever possible.
- Ultimately, a well-implemented error handling system contributes to a more reliable and user-friendly OS. It minimizes downtime by preventing crashes and facilitating recovery. Clear error messages improve the user experience, while logged errors aid in debugging. Robust error handling even strengthens system security by mitigating potential vulnerabilities. In essence, error handling is the backbone of a healthy operating system.

Unit 6

Security

Security Goals

Thera are several goal of system security. Some of them are listed below:

Integrity

- Unauthorized users must not be allowed to access the system's objects, and users with insufficient rights should not modify the system's critical files and resources.

Secrecy

- The system's objects must only be available to a small number of authorized users. The system files should not be accessible to everyone.

Availability

- All system resources must be accessible to all authorized users, i.e., no single user/process should be able to consume all system resources. If such a situation arises, service denial may occur. In this case, malware may restrict system resources and preventing legitimate processes from accessing them.

Security Attacks

Security attacks are any attempts to steal, expose, alter, disable, or destroy information, infrastructure, or systems. They can be motivated by financial gain, political activism, or even personal revenge.

There are two main types of security attacks:

i. passive attacks

ii. Active attacks

Active Attacks

Active attacks are a type of cybersecurity attack in which an attacker attempts to alter, destroy, or disrupt the normal operation of a system or network. Active attacks involve the attacker taking direct action against the target system or network, and can be more dangerous than passive attacks, which involve simply monitoring or eavesdropping on a system or network.

Types of active attacks are as follows:

- Masquerade
- Modification of messages
- Repudiation
- Replay
- Denial of Service

Passive Attacks

A Passive attack attempts to learn or make use of information from the system but does not affect system resources. Passive Attacks are in the nature of eavesdropping on or monitoring transmission. The goal of the opponent is to obtain information that is being transmitted. Passive attacks involve an attacker passively monitoring or collecting data without altering or destroying it. Examples of passive attacks include

eavesdropping, where an attacker listens in on network traffic to collect sensitive information, and sniffing, where an attacker captures and analyzes data packets to steal sensitive information.

Types of Passive attacks are as follows:

- The release of message content
- Traffic analysis

Cryptography Basic

Cryptography is the practice and study of techniques for securing communication and data from adversaries. It involves transforming plaintext (human-readable data) into ciphertext (unreadable data) using various algorithms and keys. The recipient can then decrypt the ciphertext back into plaintext, provided they have the appropriate key.

Some basic concept in cryptography

1. **Encryption**: The process of converting plaintext into ciphertext using an encryption algorithm and a key.
2. **Decryption**: The process of converting ciphertext back into plaintext using a decryption algorithm and a key.
3. **3. Key**: A parameter used in conjunction with the encryption or decryption algorithm. Keys can be public (known to everyone) or private (known only to the parties involved).
4. **4. Symmetric Encryption**: A type of encryption where the same key is used for both encryption and decryption. Examples include AES (Advanced Encryption Standard) and DES (Data Encryption Standard).
5. **5. Asymmetric Encryption (Public-Key Cryptography)**: A type of encryption where two keys are used - a public key for encryption and a private key for decryption. The public key can be shared openly, while the private key is kept secret. RSA (Rivest-Shamir-Adleman) and ECC (Elliptic Curve Cryptography) are common asymmetric encryption algorithms.

Access Control list (ACL)

Access-list (ACL) is a set of rules defined for controlling network traffic and reducing network attacks. ACLs are used to filter traffic based on the set of rules defined for the incoming or outgoing of the network.

ACL Features

1. The set of rules defined are matched serial wise i.e matching starts with the first line, then 2nd, then 3rd, and so on.
2. The packets are matched only until it matches the rule. Once a rule is matched then no further comparison takes place and that rule will be performed.
3. There is an implicit denial at the end of every ACL, i.e., if no condition or rule matches then the packet will be discarded.

Once the access-list is built, then it should be applied to inbound or outbound of the interface:

- **Inbound access lists –**
  When an access list is applied on inbound packets of the interface then first the packets will be processed according to the access list and then routed to the outbound interface.

- **Outbound access lists –**
  When an access list is applied on outbound packets of the interface then first the packet will be routed and then processed at the outbound interface.

Protection Mechanism

Protection mechanisms are tools built into a system's architecture to enforce security policies. These policies dictate who can access what information and resources within the system.

Needs of protection in Operating System

1. There may be security risks like unauthorized reading, writing, modification, or preventing the system from working effectively for authorized users.
2. It helps to ensure data security, process security, and program security against unauthorized user access or program access.
3. It is important to ensure no access rights' breaches, no viruses, no unauthorized access to the existing data.
4. Its purpose is to ensure that only the systems' policies access programs, resources, and data.