

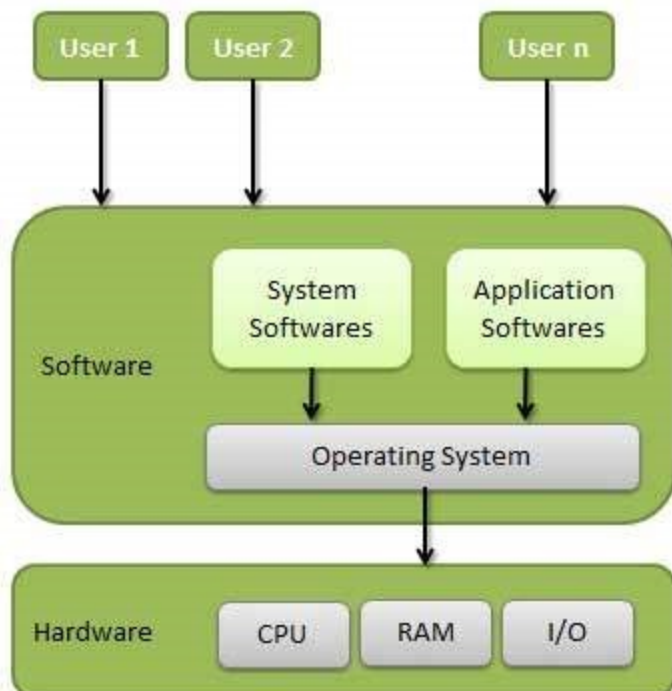
B.C.A study

Unit- 1: Introduction of operating system

An Operating System (OS) is an interface between a computer user and computer hardware. An operating system is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.

Definition

An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs.



Some popular Operating Systems include Linux Operating System, Windows Operating System, VMS, OS/400, AIX, z/OS, etc.

Following are some of important functions of an operating System.

- Memory Management
- Processor Management
- Device Management
- File Management
- Security
- Control over system performance
- Job accounting
- Error detecting aids
- Coordination between other software and users

Applications of Operating System

Following are some of the important activities that an Operating System performs –

- **Security** – By means of password and similar other techniques, it prevents unauthorized access to programs and data.
- **Control over system performance** – Recording delays between request for a service and response from the system.
- **Job accounting** – Keeping track of time and resources used by various jobs and users.
- **Error detecting aids** – Production of dumps, traces, error messages, and other debugging and error detecting aids.
- **Coordination between other softwares and users** – Coordination and assignment of compilers, interpreters, assemblers and other software to the various users of the computer systems.

Types of operating system

Operating systems are there from the very first computer generation and they keep evolving with time. In this chapter, we will discuss some of the important types of operating systems which are most commonly used.

Batch operating system

The users of a batch operating system do not interact with the computer directly. Each user prepares his job on an off-line device like punch cards and submits it to the computer operator. To speed up processing, jobs with similar needs are batched together and run as a group. The programmers leave

their programs with the operator and the operator then sorts the programs with similar requirements into batches.

The problems with Batch Systems are as follows –

- Lack of interaction between the user and the job.
- CPU is often idle, because the speed of the mechanical I/O devices is slower than the CPU.
- Difficult to provide the desired priority.

Time-sharing operating systems

Time-sharing is a technique which enables many people, located at various terminals, to use a particular computer system at the same time. Time-sharing or multitasking is a logical extension of multiprogramming. Processor's time which is shared among multiple users simultaneously is termed as time-sharing.

The main difference between Multiprogrammed Batch Systems and Time-Sharing Systems is that in case of Multiprogrammed batch systems, the objective is to maximize processor use, whereas in Time-Sharing Systems, the objective is to minimize response time.

Multiple jobs are executed by the CPU by switching between them, but the switches occur so frequently. Thus, the user can receive an immediate response. For example, in a transaction processing, the processor executes each user program in a short burst or quantum of computation. That is, if n users are present, then each user can get a time quantum. When the user submits the command, the response time is in few seconds at most.

The operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time. Computer systems that were designed primarily as batch systems have been modified to time-sharing systems.

Advantages of Timesharing operating systems are as follows –

- Provides the advantage of quick response.
- Avoids duplication of software.
- Reduces CPU idle time.

Disadvantages of Time-sharing operating systems are as follows –

- Problem of reliability.
- Question of security and integrity of user programs and data.
- Problem of data communication.

Distributed operating System

Distributed systems use multiple central processors to serve multiple real-time applications and multiple users. Data processing jobs are distributed among the processors accordingly.

The processors communicate with one another through various communication lines (such as high-speed buses or telephone lines). These are referred to as **loosely coupled systems** or distributed systems. Processors in a distributed system may vary in size and function. These processors are referred to as sites, nodes, computers, and so on.

The advantages of distributed systems are as follows –

- With resource sharing facility, a user at one site may be able to use the resources available at another.
- Speedup the exchange of data with one another via electronic mail.
- If one site fails in a distributed system, the remaining sites can potentially continue operating.
- Better service to the customers.
- Reduction of the load on the host computer.
- Reduction of delays in data processing.

Real Time operating System

A real-time system is defined as a data processing system in which the time interval required to process and respond to inputs is so small that it controls the environment. The time taken by the system to respond to an input and display of required updated information is termed as the **response time**. So in this method, the response time is very less as compared to online processing.

Real-time systems are used when there are rigid time requirements on the operation of a processor or the flow of data and real-time systems can be used as a control device in a dedicated application. A real-time operating system must have well-defined, fixed time constraints, otherwise the system will fail. For example, Scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, air traffic control systems, etc.

There are two types of real-time operating systems.

Hard real-time systems

Hard real-time systems guarantee that critical tasks complete on time. In hard real-time systems, secondary storage is limited or missing and the data is stored in ROM. In these systems, virtual memory is almost never found.

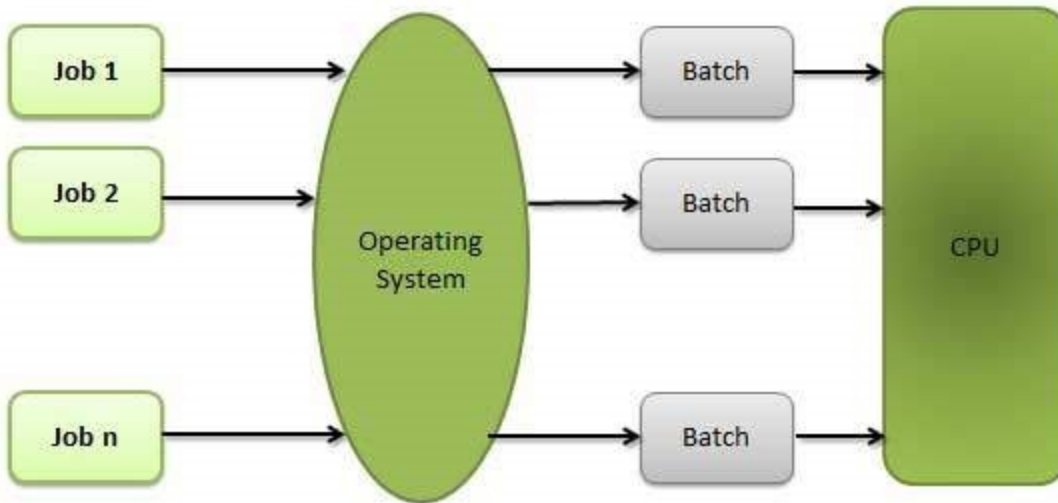
Soft real-time systems

Soft real-time systems are less restrictive. A critical real-time task gets priority over other tasks and retains the priority until it completes. Soft real-time systems have limited utility than hard real-time systems. For example, multimedia, virtual reality, Advanced Scientific Projects like undersea exploration and planetary rovers

Batch processing

Batch processing is a technique in which an Operating System collects the programs and data together in a batch before processing starts. An operating system does the following activities related to batch processing –

- The OS defines a job which has predefined sequence of commands, programs and data as a single unit.
- The OS keeps a number a jobs in memory and executes them without any manual information.
- Jobs are processed in the order of submission, i.e., first come first served fashion.
- When a job completes its execution, its memory is released and the output for the job gets copied into an output spool for later printing or processing.



Advantages

- Batch processing takes much of the work of the operator to the computer.
- Increased performance as a new job get started as soon as the previous job is finished, without any manual intervention.

Disadvantages

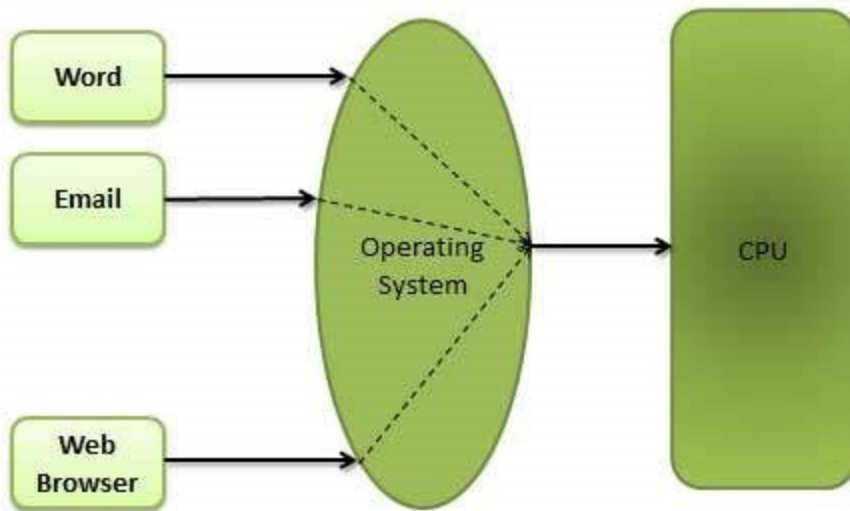
- Difficult to debug program.
- A job could enter an infinite loop.
- Due to lack of protection scheme, one batch job can affect pending jobs.

Multitasking

Multitasking is when multiple jobs are executed by the CPU simultaneously by switching between them. Switches occur so frequently that the users may interact with each program while it is running. An OS does the following activities related to multitasking –

- The user gives instructions to the operating system or to a program directly, and receives an immediate response.
- The OS handles multitasking in the way that it can handle multiple operations/executes multiple programs at a time.

- Multitasking Operating Systems are also known as Time-sharing systems.
- These Operating Systems were developed to provide interactive use of a computer system at a reasonable cost.
- A time-shared operating system uses the concept of CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared CPU.
- Each user has at least one separate program in memory.



- A program that is loaded into memory and is executing is commonly referred to as a **process**.
- When a process executes, it typically executes for only a very short time before it either finishes or needs to perform I/O.
- Since interactive I/O typically runs at slower speeds, it may take a long time to complete. During this time, a CPU can be utilized by another process.
- The operating system allows the users to share the computer simultaneously. Since each action or command in a time-shared system tends to be short, only a little CPU time is needed for each user.
- As the system switches CPU rapidly from one user/program to the next, each user is given the impression that he/she has his/her own CPU, whereas actually one CPU is being shared among many users.

Multiprogramming

Sharing the processor, when two or more programs reside in memory at the same time, is referred to as **multiprogramming**. Multiprogramming assumes a single shared processor. Multiprogramming increases CPU utilization by organizing jobs so that the CPU always has one to execute.

The following figure shows the memory layout for a multiprogramming system.



An OS does the following activities related to multiprogramming.

- The operating system keeps several jobs in memory at a time.
- This set of jobs is a subset of the jobs kept in the job pool.
- The operating system picks and begins to execute one of the jobs in the memory.
- Multiprogramming operating systems monitor the state of all active programs and system resources using memory management programs to ensure that the CPU is never idle, unless there are no jobs to process.

Advantages

- High and efficient CPU utilization.
- User feels that many programs are allotted CPU almost simultaneously.

Disadvantages

- CPU scheduling is required.
- To accommodate many jobs in memory, memory management is required.

Interactivity

Interactivity refers to the ability of users to interact with a computer system. An Operating system does the following activities related to interactivity –

- Provides the user an interface to interact with the system.
- Manages input devices to take inputs from the user. For example, keyboard.
- Manages output devices to show outputs to the user. For example, Monitor.

The response time of the OS needs to be short, since the user submits and waits for the result.

Real Time System

Real-time systems are usually dedicated, embedded systems. An operating system does the following activities related to real-time system activity.

- In such systems, Operating Systems typically read from and react to sensor data.
- The Operating system must guarantee response to events within fixed periods of time to ensure correct performance.

Introduction to Memory Management

Main Memory refers to a physical memory that is the internal memory to the computer. The word main is used to distinguish it from external mass storage devices such as disk drives. Main memory is also known as RAM. The computer is able to change only data that is in main memory. Therefore, every program we execute and every file we access must be copied from a storage device into main memory.

All the programs are loaded in the main memory for execution. Sometimes complete program is loaded into the memory, but some times a certain part or routine of the program is loaded into the main memory only when it is called by the program, this mechanism is called **Dynamic Loading**, this enhance the performance.

Also, at times one program is dependent on some other program. In such a case, rather than loading all the dependent programs, CPU links the dependent programs to the main executing program when its required. This mechanism is known as **Dynamic Linking**.

Swapping

A process needs to be in memory for execution. But sometimes there is not enough main memory to hold all the currently active processes in a timesharing system. So, excess process are kept on disk and brought in to run dynamically. Swapping is the process of bringing in each process in main memory, running it for a while and then putting it back to the disk.

Contiguous Memory Allocation

In contiguous memory allocation each process is contained in a single contiguous block of memory. Memory is divided into several fixed size partitions. Each partition contains exactly one process. When a partition is free, a process is selected from the input queue and loaded into it. The free blocks of memory are known as *holes*. The set of holes is searched to determine which hole is best to allocate.

Memory Protection

Memory protection is a phenomenon by which we control memory access rights on a computer. The main aim of it is to prevent a process from accessing memory that has not been allocated to it. Hence prevents a bug within a process from affecting other processes, or the operating system itself, and instead results in a segmentation fault or storage violation exception being sent to the disturbing process, generally killing of process.

Memory Allocation

Memory allocation is a process by which computer programs are assigned memory or space. It is of three types :

1. **First Fit:**The first hole that is big enough is allocated to program.
 2. **Best Fit:**The smallest hole that is big enough is allocated to program.
 3. **Worst Fit:**The largest hole that is big enough is allocated to program.
-

Fragmentation

Fragmentation occurs in a dynamic memory allocation system when most of the free blocks are too small to satisfy any request. It is generally termed as inability to use the available memory.

In such situation processes are loaded and removed from the memory. As a result of this, free holes exists to satisfy a request but is non contiguous i.e. the memory is fragmented into large no. Of small holes. This phenomenon is known as **External Fragmentation**.

Also, at times the physical memory is broken into fixed size blocks and memory is allocated in unit of block sizes. The memory allocated to a space may be slightly larger than the requested memory. The difference between allocated and required memory is known as **Internal fragmentation** i.e. the memory that is internal to a partition but is of no use.

Paging

A solution to fragmentation problem is Paging. Paging is a memory management mechanism that allows the physical address space of a process to be non-contagious. Here physical memory is divided into blocks of equal size called **Pages**. The pages belonging to a certain process are loaded into available memory frames.

Page Table

A Page Table is the data structure used by a virtual memory system in a computer operating system to store the mapping between *virtual address* and *physical addresses*.

Virtual address is also known as Logical address and is generated by the CPU. While Physical address is the address that actually exists on memory.

Segmentation

Segmentation is another memory management scheme that supports the user-view of memory. Segmentation allows breaking of the virtual address space of a single process into segments that may be placed in non-contiguous areas of physical memory.

Segmentation with Paging

Both paging and segmentation have their advantages and disadvantages, it is better to combine these two schemes to improve on each. The combined scheme is known as 'Page the Elements'. Each segment in this scheme is divided into pages and each segment is maintained in a page table. So the logical address is divided into following 3 parts :

- Segment numbers(S)
- Page number (P)
- The displacement or offset number (D)



[A WordPress.com Website.](#)

B.C.A study

Unit -1: (b)Virtual memory

Virtual Memory is a storage mechanism which offers user an illusion of having a very big main memory. It is done by treating a part of secondary memory as the main memory. In Virtual memory, the user can store processes with a bigger size than the available main memory.

Therefore, instead of loading one long process in the main memory, the OS loads the various parts of more than one process in the main memory. Virtual memory is mostly implemented with demand paging and demand segmentation.

Why Need Virtual Memory?

Here, are reasons for using virtual memory:

- Whenever your computer doesn't have space in the physical memory it writes what it needs to remember to the hard disk in a swap file as virtual memory.
- If a computer running Windows needs more memory/RAM, then installed in the system, it uses a small portion of the hard drive for this purpose.

How Virtual Memory Works?

In the modern world, virtual memory has become quite common these days. It is used whenever some pages require to be loaded in the main memory for the execution, and the memory is not available for those many pages.

So, in that case, instead of preventing pages from entering in the main memory, the OS searches for the RAM space that are minimum used in the recent times or that are not referenced into the secondary memory to make the space for the new pages in the main memory.

Let's understand virtual memory management with the help of one example.

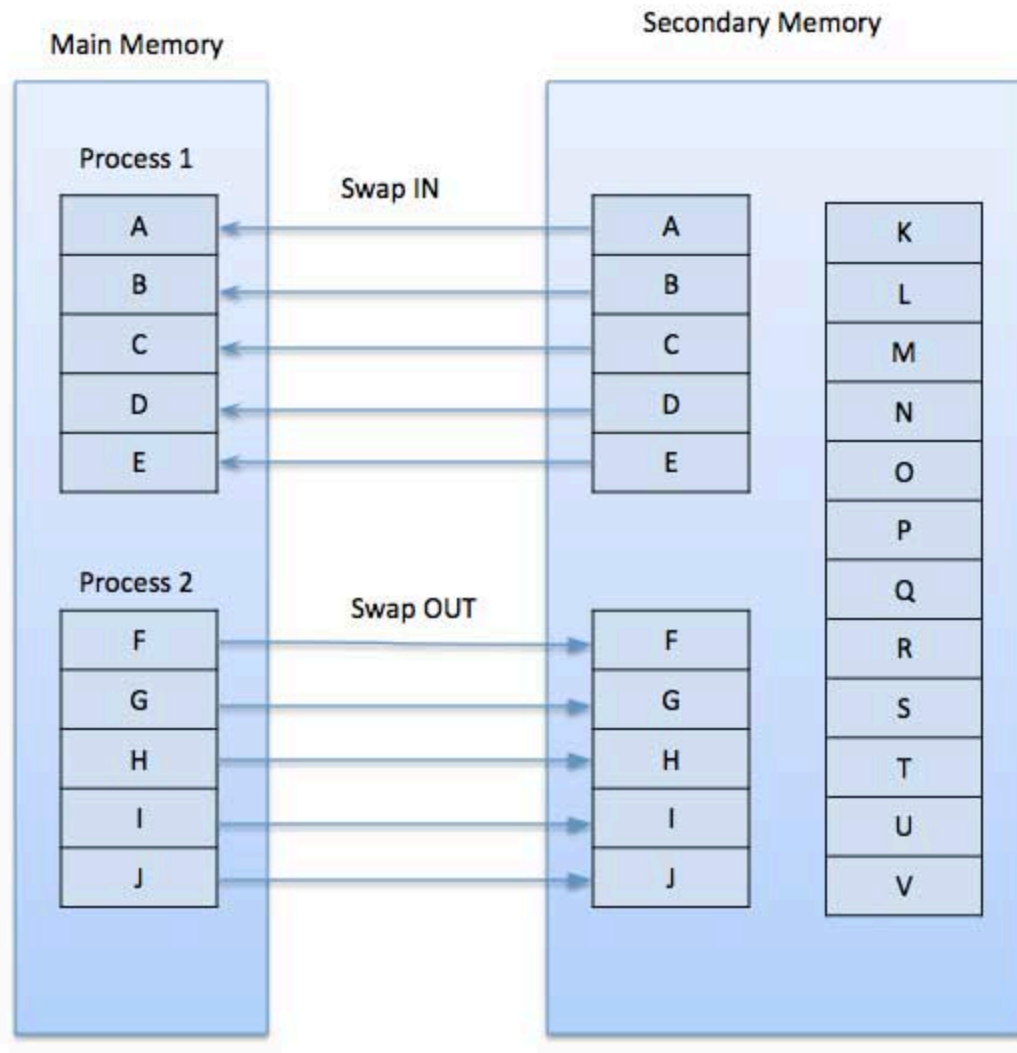
For example:

Let's assume that an OS requires 300 MB of memory to store all the running programs. However, there's currently only 50 MB of available physical memory stored on the RAM.

- The OS will then set up 250 MB of virtual memory and use a program called the Virtual Memory Manager(VMM) to manage that 250 MB.
- So, in this case, the VMM will create a file on the hard disk that is 250 MB in size to store extra memory that is required.
- The OS will now proceed to address memory as it considers 300 MB of real memory stored in the RAM, even if only 50 MB space is available.
- It is the job of the VMM to manage 300 MB memory even if just 50 MB of real memory space is available.

Demand Paging

A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance. When a context switch occurs, the operating system does not copy any of the old program's pages out to the disk or any of the new program's pages into the main memory. Instead, it just begins executing the new program after loading the first page and fetches that program's pages as they are referenced.



While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a **page fault** and transfers control from the program to the operating system to demand the page back into the memory.

Advantages

Following are the advantages of Demand Paging –

- Large virtual memory.
- More efficient use of memory.
- There is no limit on degree of multiprogramming.

Disadvantages

- Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.

Page Replacement Algorithm

Page replacement algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated. Paging happens whenever a page fault occurs and a free page cannot be used for allocation purpose accounting to reason that pages are not available or the number of free pages is lower than required pages.

When the page that was selected for replacement and was paged out, is referenced again, it has to read in from disk, and this requires for I/O completion. This process determines the quality of the page replacement algorithm: the lesser the time waiting for page-ins, the better is the algorithm.

A page replacement algorithm looks at the limited information about accessing the pages provided by hardware, and tries to select which pages should be replaced to minimize the total number of page misses, while balancing it with the costs of primary storage and processor time of the algorithm itself. There are many different page replacement algorithms. We evaluate an algorithm by running it on a particular string of memory reference and computing the number of page faults,

Reference String

The string of memory references is called reference string. Reference strings are generated artificially or by tracing a given system and recording the address of each memory reference. The latter choice produces a large number of data, where we note two things.

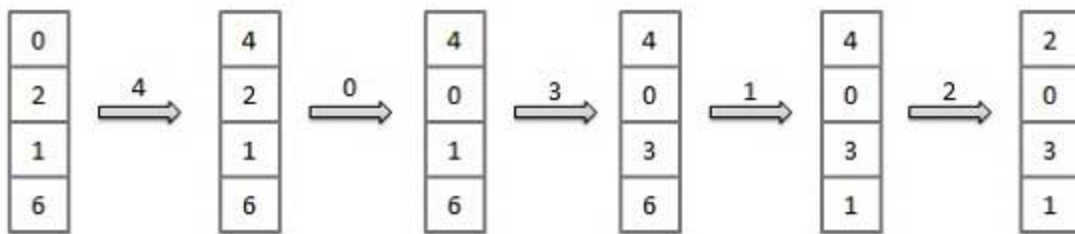
- For a given page size, we need to consider only the page number, not the entire address.
- If we have a reference to a page **p**, then any immediately following references to page **p** will never cause a page fault. Page **p** will be in memory after the first reference; the immediately following references will not fault.
- For example, consider the following sequence of addresses – 123,215,600,1234,76,96
- If page size is 100, then the reference string is 1,2,6,12,0,0

First In First Out (FIFO) algorithm

- Oldest page in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages from the tail and add new pages at the head.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x x



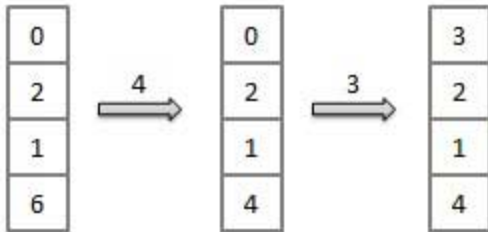
Fault Rate = $9 / 12 = 0.75$

Optimal Page algorithm

- An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.
- Replace the page that will not be used for the longest period of time. Use the time when a page is to be used.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x



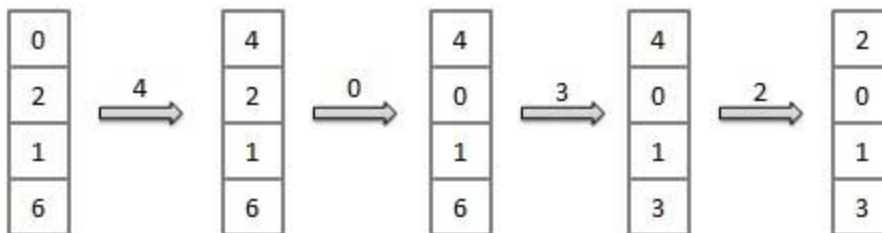
$$\text{Fault Rate} = 6 / 12 = 0.50$$

Least Recently Used (LRU) algorithm

- Page which has not been used for the longest time in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages by looking back into time.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x



$$\text{Fault Rate} = 8 / 12 = 0.67$$

Virtual and physical addresses are the same in compile-time and load-time address-binding schemes. Virtual and physical addresses differ in execution-time address-binding scheme.

The set of all logical addresses generated by a program is referred to as a **logical address space**. The set of all physical addresses corresponding to these logical addresses is referred to as a **physical address space**.

The runtime mapping from virtual to physical address is done by the memory management unit (MMU) which is a hardware device. MMU uses following mechanism to convert virtual address to physical address.

- The value in the base register is added to every address generated by a user process, which is treated as offset at the time it is sent to memory. For example, if the base register value is 10000, then an attempt by the user to use address location 100 will be dynamically reallocated to location 10100.
- The user program deals with virtual addresses; it never sees the real physical addresses.

Static vs Dynamic Loading

The choice between Static or Dynamic Loading is to be made at the time of computer program being developed. If you have to load your program statically, then at the time of compilation, the complete programs will be compiled and linked without leaving any external program or module dependency. The linker combines the object program with other necessary object modules into an absolute program, which also includes logical addresses.

If you are writing a Dynamically loaded program, then your compiler will compile the program and for all the modules which you want to include dynamically, only references will be provided and rest of the work will be done at the time of execution.

At the time of loading, with **static loading**, the absolute program (and data) is loaded into memory in order for execution to start.

If you are using **dynamic loading**, dynamic routines of the library are stored on a disk in relocatable form and are loaded into memory only when they are needed by the program.

Static vs Dynamic Linking

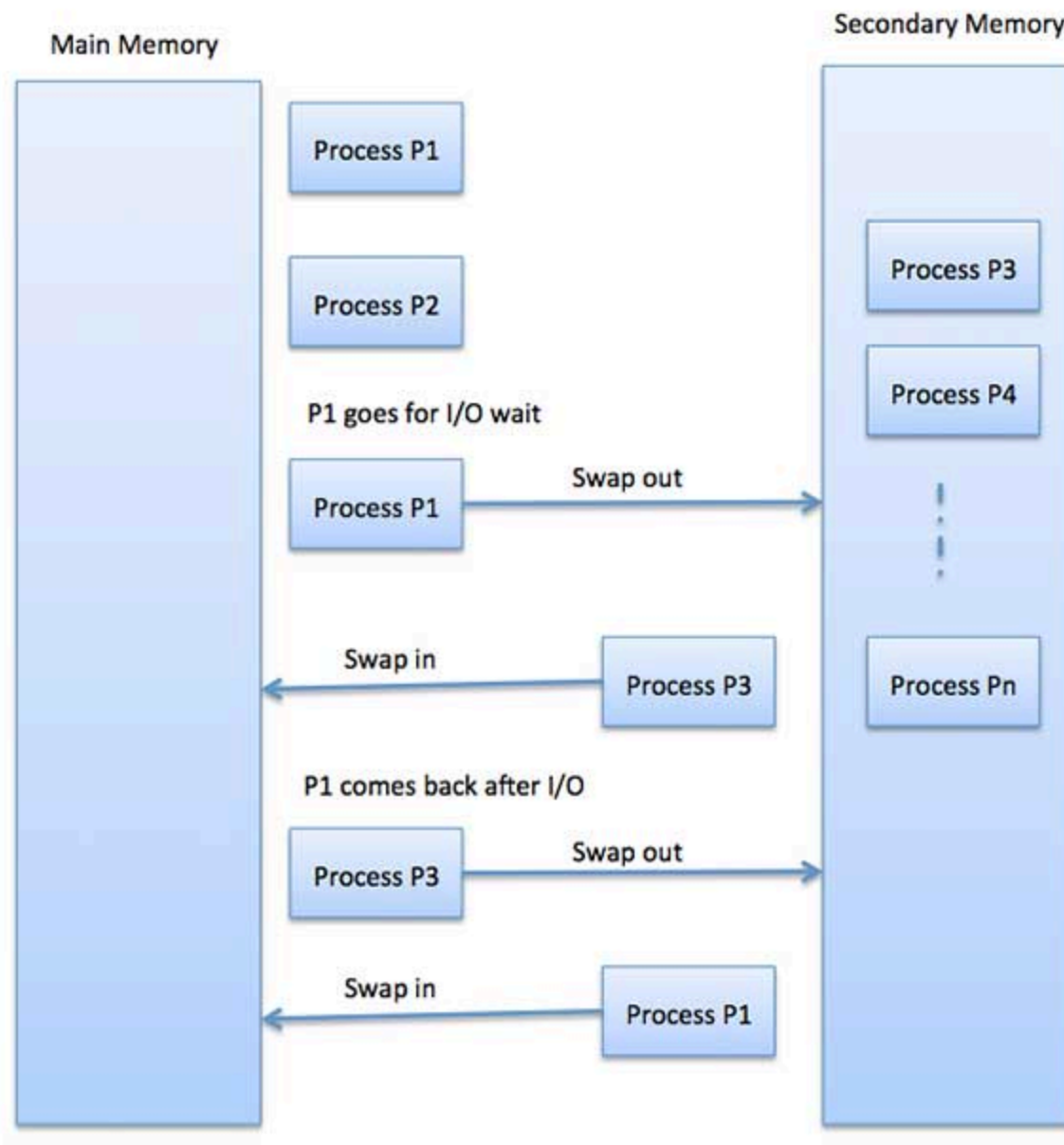
As explained above, when static linking is used, the linker combines all other modules needed by a program into a single executable program to avoid any runtime dependency.

When dynamic linking is used, it is not required to link the actual module or library with the program, rather a reference to the dynamic module is provided at the time of compilation and linking. Dynamic Link Libraries (DLL) in Windows and Shared Objects in Unix are good examples of dynamic libraries.

Swapping

Swapping is a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.

Though performance is usually affected by swapping process but it helps in running multiple and big processes in parallel and that's the reason **Swapping is also known as a technique for memory compaction.**



The total time taken by swapping process includes the time it takes to move the entire process to a secondary disk and then to copy the process back to memory, as well as the time the process takes to regain main memory.

Let us assume that the user process is of size 2048KB and on a standard hard disk where swapping will take place has a data transfer rate around 1 MB per second. The actual transfer of the 1000K process to or from memory will take

$$\begin{aligned} & 2048\text{KB} / 1024\text{KB per second} \\ & = 2 \text{ seconds} \\ & = 2000 \text{ milliseconds} \end{aligned}$$

Now considering in and out time, it will take complete 4000 milliseconds plus other overhead where the process competes to regain main memory.

Memory Allocation

Main memory usually has two partitions –

- **Low Memory** – Operating system resides in this memory.
- **High Memory** – User processes are held in high memory.

Operating system uses the following memory allocation mechanism.

S.N.	Memory Allocation & Description
1	

Single-partition allocation

In this type of allocation, relocation-register scheme is used to protect user processes from each other, and from changing operating-system code and data. Relocation register contains value of smallest physical address whereas limit register contains range of logical addresses. Each logical address must be less than the limit register.

Multiple-partition allocation

In this type of allocation, main memory is divided into a number of fixed-sized partitions where each partition should contain only one process. When a partition is free, a process is selected from the input queue and is loaded into the free partition. When the process terminates, the partition becomes available for another process.

Fragmentation

As processes are loaded and removed from memory, the free memory space is broken into little pieces. It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused. This problem is known as Fragmentation.

Fragmentation is of two types –S.N.Fragmentation & Description1

External fragmentation

Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.²

Internal fragmentation

Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.

The following diagram shows how fragmentation can cause waste of memory and a compaction technique can be used to create more free memory out of fragmented memory –

Fragmented memory before compaction



Memory after compaction



External fragmentation can be reduced by compaction or shuffle memory contents to place all free memory together in one large block. To make compaction feasible, relocation should be dynamic.

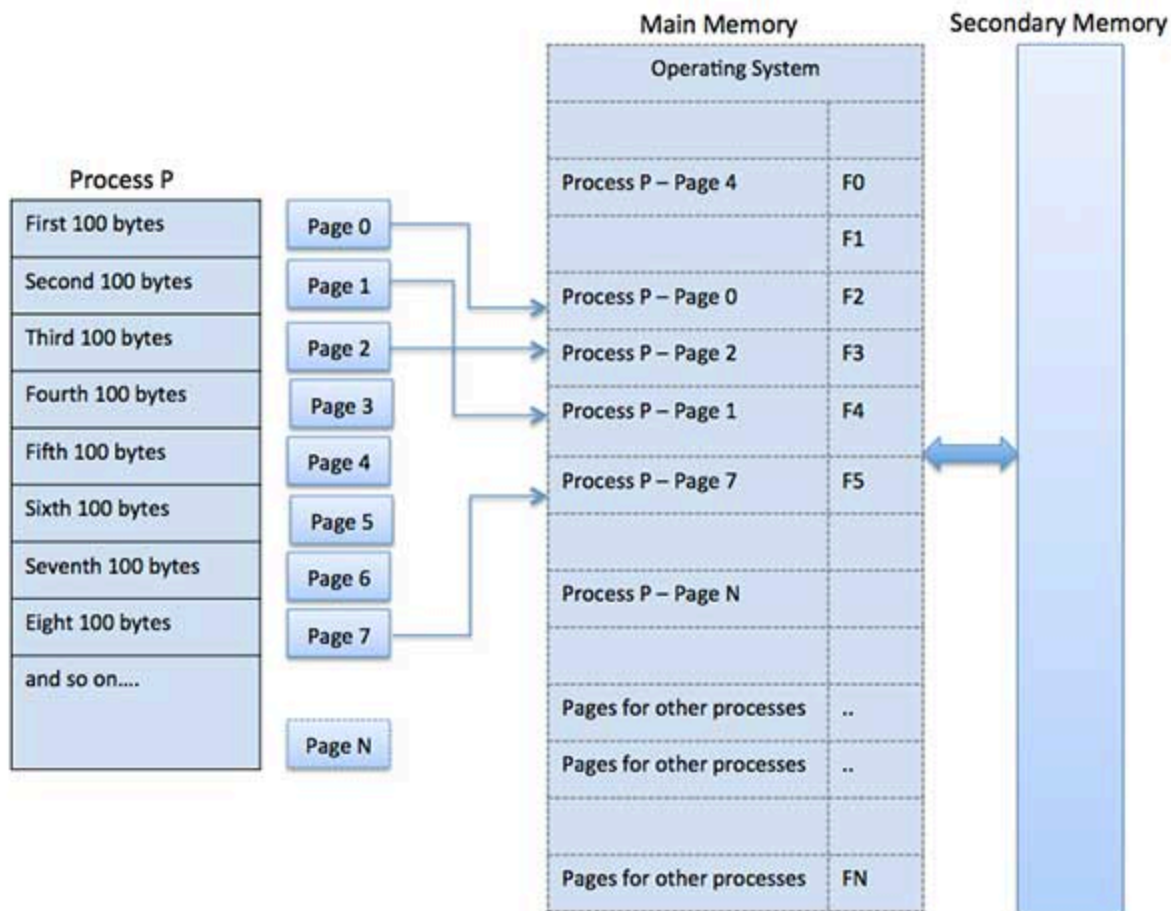
The internal fragmentation can be reduced by effectively assigning the smallest partition but large enough for the process.

Paging

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard that's set up to emulate the computer's RAM. Paging technique plays an important role in implementing virtual memory.

Paging is a memory management technique in which process address space is broken into blocks of the same size called **pages** (size is power of 2, between 512 bytes and 8192 bytes). The size of the process is measured in the number of pages.

Similarly, main memory is divided into small fixed-sized blocks of (physical) memory called **frames** and the size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to avoid external fragmentation.



Address Translation

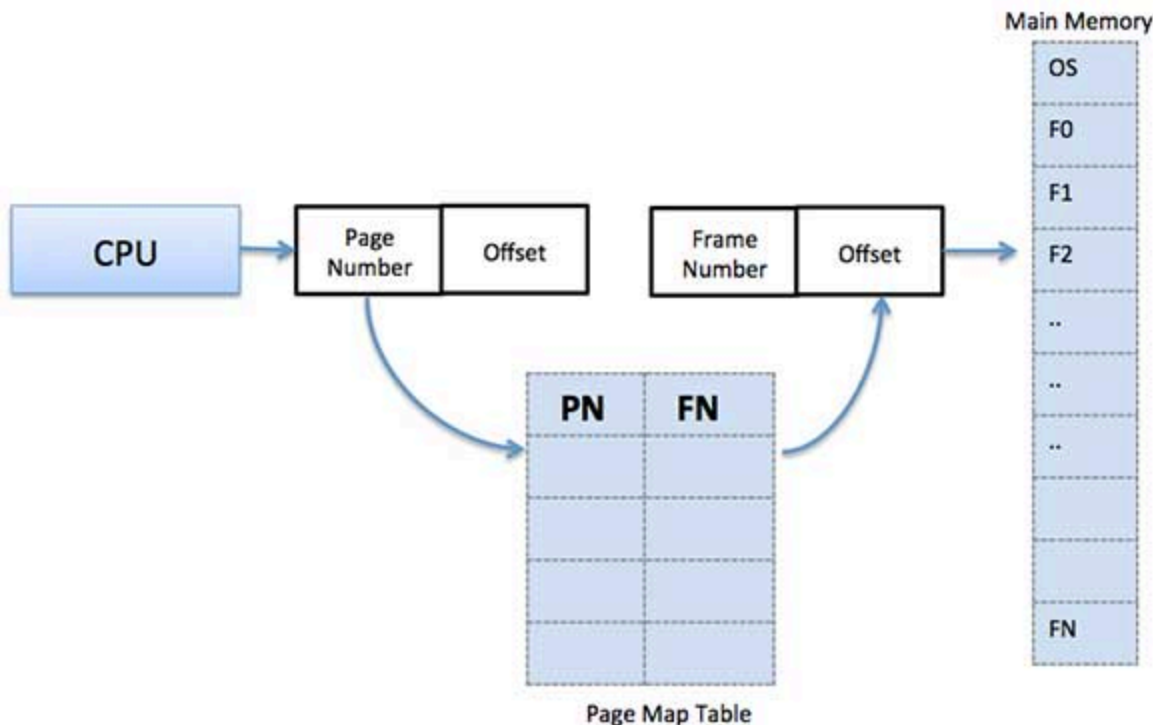
Page address is called **logical address** and represented by **page number** and the **offset**.

$$\text{Logical Address} = \text{Page number} + \text{page offset}$$

Frame address is called **physical address** and represented by a **frame number** and the **offset**.

$$\text{Physical Address} = \text{Frame number} + \text{page offset}$$

A data structure called **page map table** is used to keep track of the relation between a page of a process to a frame in physical memory.



When the system allocates a frame to any page, it translates this logical address into a physical address and create entry into the page table to be used throughout execution of the program.

When a process is to be executed, its corresponding pages are loaded into any available memory frames. Suppose you have a program of 8Kb but your memory can accommodate only 5Kb at a given point in time, then the paging concept will come into picture. When a computer runs out of RAM, the operating

system (OS) will move idle or unwanted pages of memory to secondary memory to free up RAM for other processes and brings them back when needed by the program.

This process continues during the whole execution of the program where the OS keeps removing idle pages from the main memory and write them onto the secondary memory and bring them back when required by the program.

Advantages and Disadvantages of Paging

Here is a list of advantages and disadvantages of paging –

- Paging reduces external fragmentation, but still suffer from internal fragmentation.
- Paging is simple to implement and assumed as an efficient memory management technique.
- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not be good for a system having small RAM.

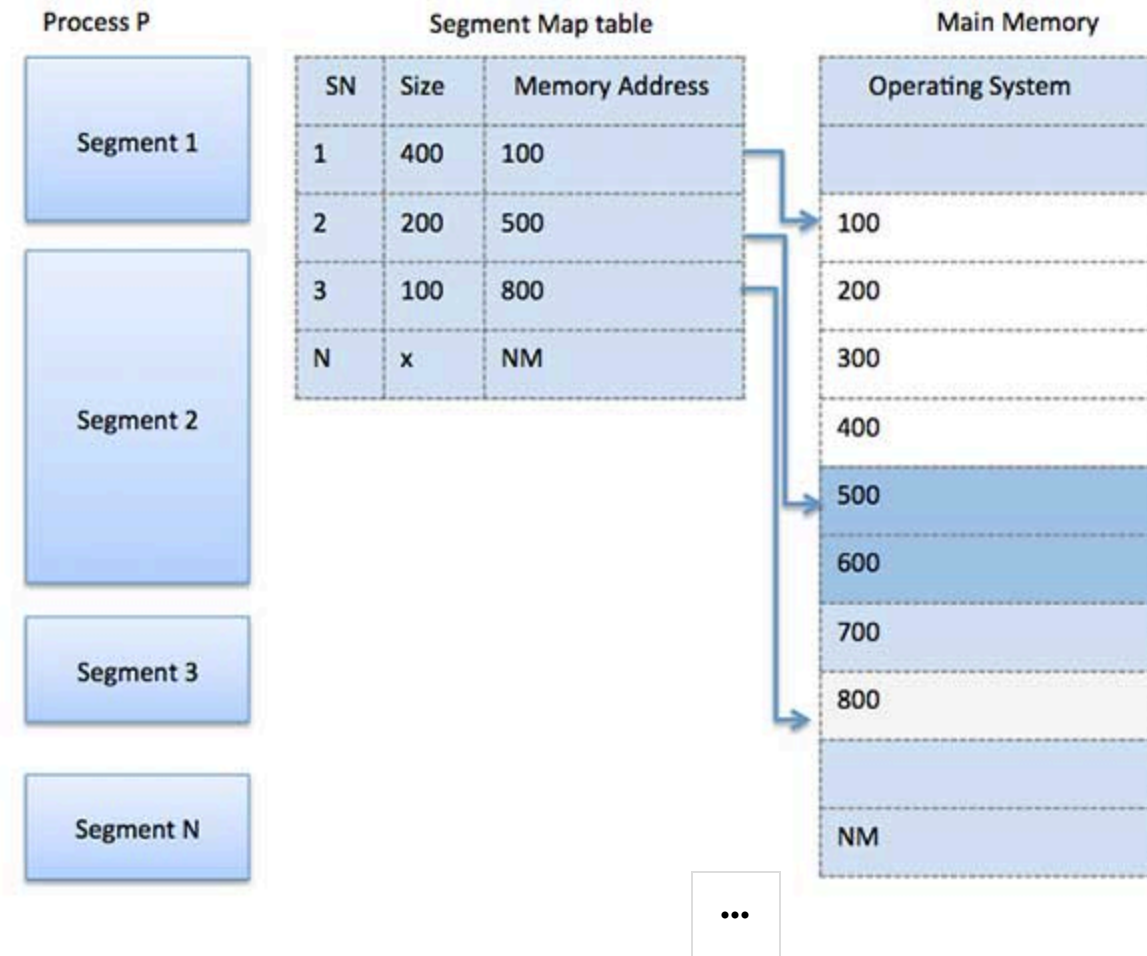
Segmentation

Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions. Each segment is actually a different logical address space of the program.

When a process is to be executed, its corresponding segmentation are loaded into non-contiguous memory though every segment is loaded into a contiguous block of available memory.

Segmentation memory management works very similar to paging but here segments are of variable-length where as in paging pages are of fixed size.

A program segment contains the program's main function, utility functions, data structures, and so on. The operating system maintains a **segment map table** for every process and a list of free memory blocks along with segment numbers, their size and corresponding memory locations in main memory. For each segment, the table stores the starting address of the segment and the length of the segment. A reference to a memory location includes a value that identifies a segment and an offset.



[A WordPress.com Website.](https://bcastudyguide.com/unit-1-bvirtual-memory/)

B.C.A study

Unit -2 : Processes

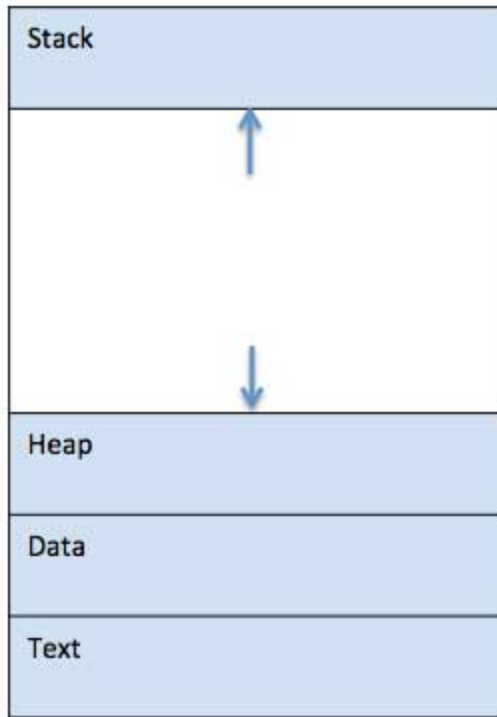
Process

A process is basically a program in execution. The execution of a process must progress in a sequential fashion.

” *A process is defined as an entity which represents the basic unit of work to be implemented in the system.*

To put it in simple terms, we write our computer programs in a text file and when we execute this program, it becomes a process which performs all the tasks mentioned in the program.

When a program is loaded into the memory and it becomes a process, it can be divided into four sections — stack, heap, text and data. The following image shows a simplified layout of a process inside main memory –



S.N.Component & Description¹

Stack

The process Stack contains the temporary data such as method/function parameters, return address and local variables.²

Heap

This is dynamically allocated memory to a process during its run time.³

Text

This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.⁴

Data

This section contains the global and static variables.

Process Life Cycle

When a process executes, it passes through different states. These stages may differ in different operating systems, and the names of these states are also not standardized.

In general, a process can have one of the following five states at a time.S.N.State & Description1

Start

This is the initial state when a process is first started/created.2

Ready

The process is waiting to be assigned to a processor. Ready processes are waiting to have the processor allocated to them by the operating system so that they can run. Process may come into this state after **Start** state or while running it by but interrupted by the scheduler to assign CPU to some other process.3

Running

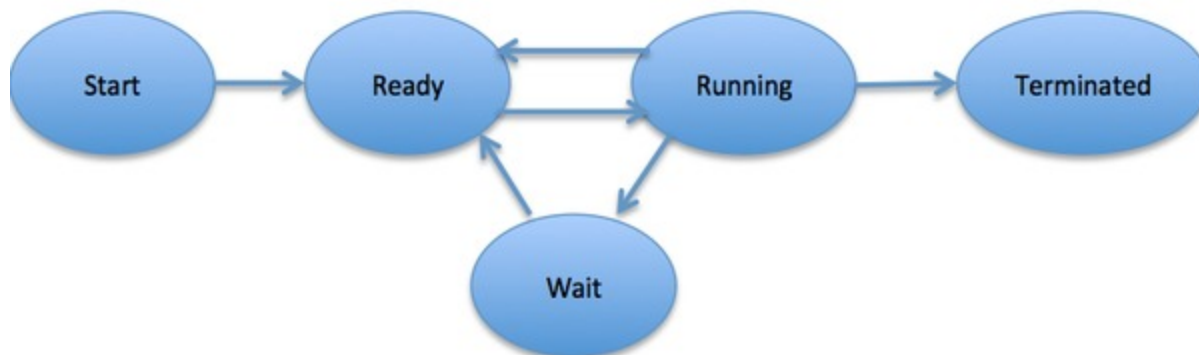
Once the process has been assigned to a processor by the OS scheduler, the process state is set to running and the processor executes its instructions.4

Waiting

Process moves into the waiting state if it needs to wait for a resource, such as waiting for user input, or waiting for a file to become available.5

Terminated or Exit

Once the process finishes its execution, or it is terminated by the operating system, it is moved to the terminated state where it waits to be removed from main memory.



Process Control Block (PCB)

A Process Control Block is a data structure maintained by the Operating System for every process. The PCB is identified by an integer process ID (PID). A PCB keeps all the information needed to keep track of a process as listed below in the table –S.N.Information & Description1

Process State

The current state of the process i.e., whether it is ready, running, waiting, or whatever.2

Process privileges

This is required to allow/disallow access to system resources.3

Process ID

Unique identification for each of the process in the operating system.4

Pointer

A pointer to parent process.5

Program Counter

Program Counter is a pointer to the address of the next instruction to be executed for this process.6

CPU registers

Various CPU registers where process need to be stored for execution for running state.7

CPU Scheduling Information

Process priority and other scheduling information which is required to schedule the process.8

Memory management information

This includes the information of page table, memory limits, Segment table depending on memory used by the operating system.9

Accounting information

This includes the amount of CPU used for process execution, time limits, execution ID etc.10

IO status information

This includes a list of I/O devices allocated to the process.

The architecture of a PCB is completely dependent on Operating System and may contain different information in different operating systems. Here is a simplified diagram of a PCB –

Process ID
State
Pointer
Priority
Program counter
CPU registers
I/O information
Accounting information
etc....

The PCB is maintained for a process throughout its lifetime, and is deleted once the process terminates.

Process Scheduling

Definition

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

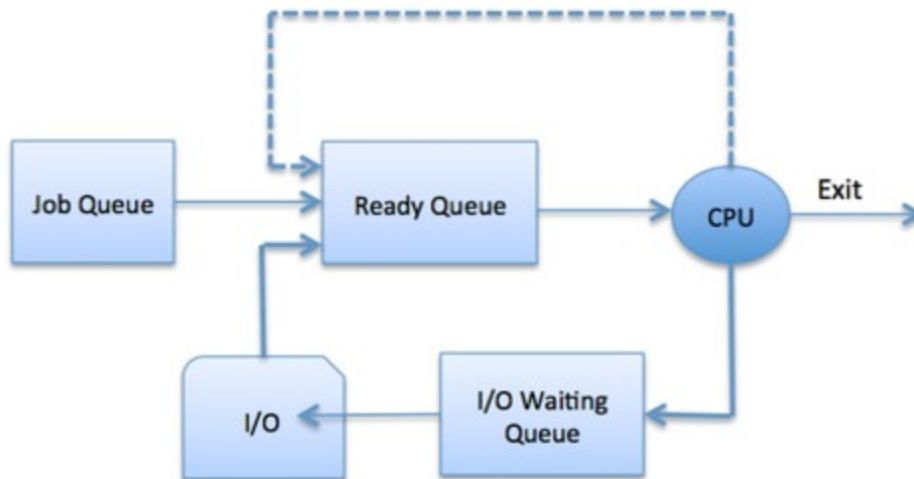
Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

Process Scheduling Queues

The OS maintains all PCBs in Process Scheduling Queues. The OS maintains a separate queue for each of the process states and PCBs of all processes in the same execution state are placed in the same queue. When the state of a process is changed, its PCB is unlinked from its current queue and moved to its new state queue.

The Operating System maintains the following important process scheduling queues –

- **Job queue** – This queue keeps all the processes in the system.
- **Ready queue** – This queue keeps a set of all processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.
- **Device queues** – The processes which are blocked due to unavailability of an I/O device constitute this queue.



The OS can use different policies to manage each queue (FIFO, Round Robin, Priority, etc.). The OS scheduler determines how to move processes between the ready and run queues which can only have one entry per processor core on the system; in the above diagram, it has been merged with the CPU.

Schedulers

Schedulers are special system software which handle process scheduling in various ways. Their main task is to select the jobs to be submitted into the system and to decide which process to run. Schedulers are of three types –

- Long-Term Scheduler
- Short-Term Scheduler
- Medium-Term Scheduler

Long Term Scheduler

It is also called a **job scheduler**. A long-term scheduler determines which programs are admitted to the system for processing. It selects processes from the queue and loads them into memory for execution. Process loads into the memory for CPU scheduling.

The primary objective of the job scheduler is to provide a balanced mix of jobs, such as I/O bound and processor bound. It also controls the degree of multiprogramming. If the degree of multiprogramming is stable, then the average rate of process creation must be equal to the average departure rate of processes leaving the system.

On some systems, the long-term scheduler may not be available or minimal. Time-sharing operating systems have no long term scheduler. When a process changes the state from new to ready, then there is use of long-term scheduler.

Short Term Scheduler

It is also called as **CPU scheduler**. Its main objective is to increase system performance in accordance with the chosen set of criteria. It is the change of ready state to running state of the process. CPU scheduler selects a process among the processes that are ready to execute and allocates CPU to one of them.

Short-term schedulers, also known as dispatchers, make the decision of which process to execute next. Short-term schedulers are faster than long-term schedulers.

Medium Term Scheduler

Medium-term scheduling is a part of **swapping**. It removes the processes from the memory. It reduces the degree of multiprogramming. The medium-term scheduler is in-charge of handling the swapped out-processes.

A running process may become suspended if it makes an I/O request. A suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other processes, the suspended process is moved to the secondary storage. This process is called **swapping**, and the process is said to be swapped out or rolled out. Swapping may be necessary to improve the process mix.

Operating System Scheduling algorithms

A Process Scheduler schedules different processes to be assigned to the CPU based on particular scheduling algorithms. There are six popular process scheduling algorithms which we are going to discuss in this chapter –

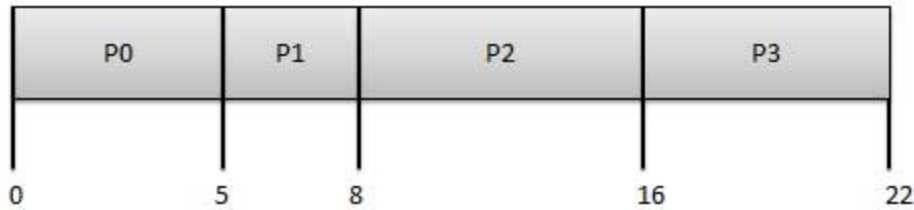
- First-Come, First-Served (FCFS) Scheduling
- Shortest-Job-Next (SJN) Scheduling
- Priority Scheduling
- Shortest Remaining Time
- Round Robin(RR) Scheduling
- Multiple-Level Queues Scheduling

These algorithms are either **non-preemptive or preemptive**. Non-preemptive algorithms are designed so that once a process enters the running state, it cannot be preempted until it completes its allotted time, whereas the preemptive scheduling is based on priority where a scheduler may preempt a low priority running process anytime when a high priority process enters into a ready state.

First Come First Serve (FCFS)

- Jobs are executed on first come, first serve basis.
- It is a non-preemptive, pre-emptive scheduling algorithm.
- Easy to understand and implement.
- Its implementation is based on FIFO queue.
- Poor in performance as average wait time is high.

Process	Arrival Time	Execute Time	Service Time
P0	0	5	0
P1	1	3	5
P2	2	8	8
P3	3	6	16



Wait time of each process is as follows –
 Process Wait Time : Service Time – Arrival Time
 $P0 - 0 = 0$
 $P1 - 1 = 4$
 $P2 - 2 = 6$
 $P3 - 3 = 13$

Average Wait Time: $(0+4+6+13) / 4 = 5.75$

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 processor should know in advance how much time process will take.

Given: Table of processes, and their Arrival time, Execution time
 Process Arrival Time Execution Time
 Service Time
 P0 0 5
 P1 1 3
 P2 2 8
 P3 3 6

 Shortest Job First Scheduling Algorithm

Waiting time of each process is as follows –
 Process Waiting Time
 $P0 - 0 = 0$
 $P1 - 1 = 4$
 $P2 - 2 = 12$
 $P3 - 3 = 5$


Average Wait Time: $(0 + 4 + 12 + 5) / 4 = 21 / 4 = 5.25$

Priority Based Scheduling

- Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems.
- Each process is assigned a priority. Process with highest priority is to be executed first and so on.
- Processes with same priority are executed on first come first served basis.
- Priority can be decided based on memory requirements, time requirements or any other resource requirement.

Given: Table of processes, and their Arrival time, Execution time, and priority. Here we are considering 1 is the lowest priority.

Process	Arrival Time	Execution Time	Priority	Service
P0	0	5	1	
P1	1	3	2	
P2	1	1	1	
P3	2	8	3	
P5	3	6	5	

 Priority Scheduling Algorithm

Waiting time of each process is as follows –

Process	Waiting Time
P0	0
P1	1
P2	1
P3	2
P5	3

Average Wait Time: $(0 + 1 + 1 + 2 + 3) / 5 = 7 / 5 = 1.4$

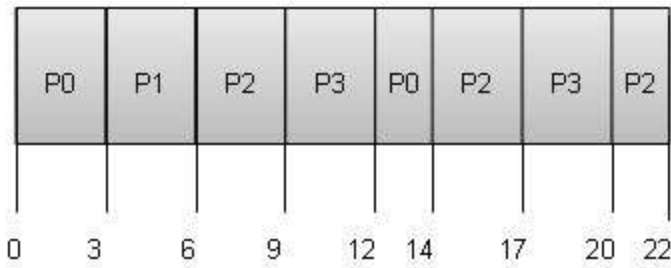
Shortest Remaining Time

- Shortest remaining time (SRT) is the preemptive version of the SJN algorithm.
- The processor is allocated to the job closest to completion but it can be preempted by a newer ready job with shorter time to completion.
- Impossible to implement in interactive systems where required CPU time is not known.
- It is often used in batch environments where short jobs need to give preference.

Round Robin Scheduling

- Round Robin is the preemptive process scheduling algorithm.
- Each process is provided a fix time to execute, it is called a **quantum**.
- Once a process is executed for a given time period, it is preempted and other process executes for a given time period.
- Context switching is used to save states of preempted processes.

Quantum = 3



Wait time of each process is as follows – Process Wait Time : Service Time – Arrival Time

P0: $(0 - 0) + (12 - 3) = 9$
 P1: $(3 - 1) = 2$
 P2: $(6 - 2) + (14 - 9) + (20 - 17) = 12$
 P3: $(9 - 3) + (17 - 12) = 11$

Average Wait Time: $(9+2+12+11) / 4 = 8.5$

Introduction of Process Synchronization

On the basis of synchronization, processes are categorized as one of the following two types:

- **Independent Process** : Execution of one process does not affect the execution of other processes.
- **Cooperative Process** : Execution of one process affects the execution of other processes.

Process synchronization problem arises in the case of Cooperative process also because resources are shared in Cooperative processes.

Race Condition

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

Several processes access and process the manipulations over the same data concurrently, then the outcome depends on the particular order in which the access takes place.

Critical Section Problem

Critical section is a code segment that can be accessed by only one process at a time. Critical section contains shared variables which need to be synchronized to maintain consistency of data variables.

In the entry section, the process requests for entry in the **Critical Section**.

Semaphores

A Semaphore is an integer variable, which can be accessed only through two operations *wait()* and *signal()*.

There are two types of semaphores : Binary Semaphores and Counting Semaphores

- Binary Semaphores : They can only be either 0 or 1. They are also known as mutex locks, as the locks can provide mutual exclusion. All the processes can share the same mutex semaphore that is initialized to 1. Then, a process has to wait until the lock becomes 0. Then, the process can make the mutex semaphore 1 and start its critical section. When it completes its critical section, it can reset the value of mutex semaphore to 0 and some other process can enter its critical section.
- Counting Semaphores: They can have any value and are not restricted over a certain domain. They can be used to control access to a resource that has a limitation on the number of simultaneous accesses. The semaphore can be initialized to the number of instances of the resource. Whenever a process wants to use that resource, it checks if the number of remaining instances is more than zero, i.e., the process has an instance available. Then, the process can enter its critical section thereby decreasing the value of the counting semaphore by 1. After the process is over with the use of the instance of the resource, it can leave the critical section thereby adding 1 to the number of available instances of the resource.

Classical problems of Synchronization with Semaphore Solution

In this article, we will see number of classical problems of synchronization as examples of a large class of concurrency-control problems. In our solutions to the problems, we use semaphores for synchronization, since that is the traditional way to present such solutions. However, actual implementations of these solutions could use mutex locks in place of binary semaphores.

These problems are used for testing nearly every newly proposed synchronization scheme. The following problems of synchronization are considered as classical problems:

1. Bounded-buffer (or Producer-Consumer) Problem,
2. Dining-Philosophers Problem,
3. Readers and Writers Problem,
4. Sleeping Barber Problem

These are summarized, for detailed explanation, you can view the linked articles for each.

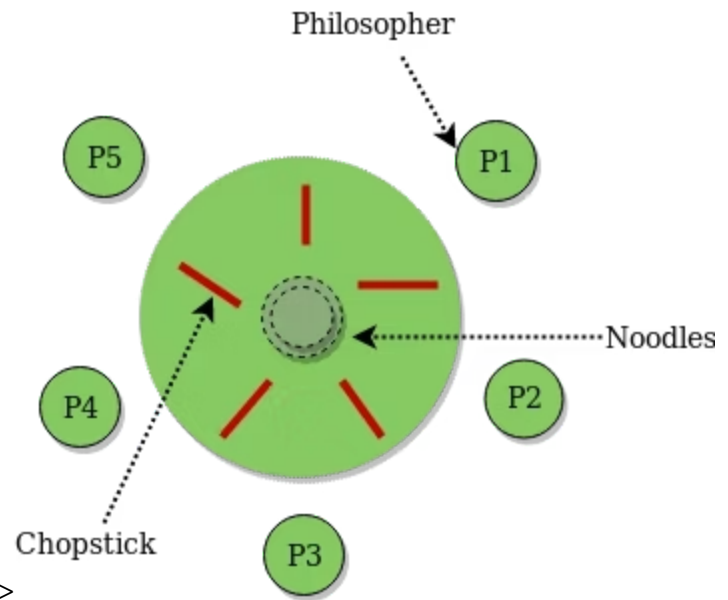
1. :Producer consumer problem

Bounded Buffer problem is also called producer consumer problem. This problem is generalized in terms of the Producer-Consumer problem. Solution to this problem is, creating two counting semaphores “full” and “empty” to keep track of the current number of full and empty buffers respectively. Producers produce a product and consumers consume the product, but both use of one of the containers each time.

1. Dining philosopher problem:

The Dining Philosopher Problem states that K philosophers seated around a circular table with one chopstick between each pair of philosophers. There is one chopstick between each

philosopher. A philosopher may eat if he can pickup the two chopsticks adjacent to him. One chopstick may be picked up by any one of its adjacent followers but not both. This problem involves the allocation of limited resources to a group of processes in a deadlock-free and starvation-free manner.



src="data:image/svg+xml; charset=utf-8," />

1. Readers and writers problem:

Suppose that a database is to be shared among several concurrent processes. Some of these processes may want only to read the database, whereas others may want to update (that is, to read and write) the database. We distinguish between these two types of processes by referring to the former as readers and to the latter as writers. Precisely in OS we call this situation as the readers-writers problem. Problem parameters:

- One set of data is shared among a number of processes.
- Once a writer is ready, it performs its write. Only one writer may write at a time.
- If a process is writing, no other process can read it.
- If at least one reader is reading, no other process can write.
- Readers may not write and only read.

2. Sleeping (<https://www.geeksforgeeks.org/operating-system-sleeping-barber-problem/>) Barber Problem:

Barber shop with one barber, one barber chair and N chairs to wait in. When no customers the barber goes to sleep in barber chair and must be woken when a customer comes in. When barber is cutting hair new customers take empty seats to wait, or leave if no vacancy.



[A WordPress.com Website.](#)

B.C.A study

Unit -3 :Deadlock

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

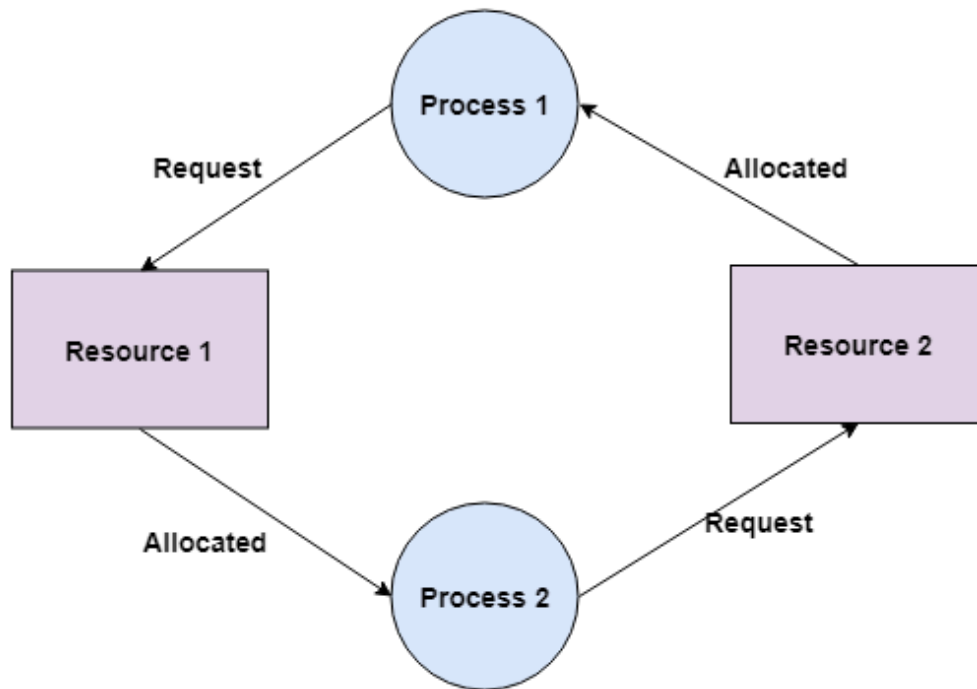
Coffman Conditions

A deadlock occurs if the four Coffman conditions hold true. But these conditions are not mutually exclusive.

The Coffman conditions are given as follows –

- **Mutual Exclusion** There should be a resource that can only be held by one process at a time. In the diagram below, there is a single instance of Resource 1 and it is held by Process 1 only.
- **Hold and Wait** A process can hold multiple resources and still request more resources from other processes which are holding them. In the diagram given below, Process 2 holds Resource 2 and Resource 3 and is requesting the Resource 1 which is held by Process 1.
- **No Preemption** A resource cannot be preempted from a process by force. A process can only release a resource voluntarily. In the diagram below, Process 2 cannot preempt Resource 1 from Process 1. It will only be released when Process 1 relinquishes it voluntarily after its execution is complete.
- **Circular Wait** A process is waiting for the resource held by the second process, which is waiting for the resource held by the third process and so on, till the last process is waiting for a resource held by the first process. This forms a circular chain. For example: Process 1 is allocated Resource 2 and it is requesting Resource 1. Similarly, Process 2 is allocated Resource 1 and it is requesting Resource 2.

This forms a circular wait loop.



Methods for Handling Deadlocks

- Generally speaking there are three ways of handling deadlocks:
 1. Deadlock prevention or avoidance – Do not allow the system to get into a deadlocked state.
 2. Deadlock detection and recovery – Abort a process or preempt some resources when deadlocks are detected.
 3. Ignore the problem all together – If deadlocks only occur once a year or so, it may be better to simply let them happen and reboot as necessary than to incur the constant overhead and system performance penalties associated with deadlock prevention or detection. This is the approach that both Windows and UNIX take.

Deadlock Detection

A deadlock can be detected by a resource scheduler as it keeps track of all the resources that are allocated to different processes. After a deadlock is detected, it can be resolved using the following methods:

- All the processes that are involved in the deadlock are terminated. This is not a good approach as all the progress made by the processes is destroyed.
- Resources can be preempted from some processes and given to others till the deadlock is resolved.

Deadlock Prevention

Deadlock prevention algorithms ensure that at least one of the necessary conditions (Mutual exclusion, hold and wait, no preemption, and circular wait) does not hold true. We do this by facing each of the four conditions on separate occasions. However, most prevention algorithms have poor resource utilization and hence result in reduced throughputs.

Facing the Mutual Exclusion condition

If no resource were ever assigned exclusively to a single process, then we would never have deadlocks. What this means is that all resources should be shared between all processes, and that will never lead to a deadlock. However, if two processes start printing on a shared printer together, then the output would be unreadable. In this model, the only process that actually requests the printer resource is the printer daemon. A daemon is a background process that handles all requests and provides services.

The daemon is strictly programmed to start printing when the complete output file is ready and not when it is being edited.

So how do we face the mutual exclusion condition? We avoid assigning a resource when that is not necessary. And we also try to make sure that as few processes as possible might claim the resource.

Facing the Hold and Wait condition

This condition is slightly more promising. Here, if the process is allocated all the resources it would require beforehand, then it won't need to ask for more resources, and there would be no deadlock. If, for some reason, the resource can't be allocated to the process due to unavailability, then the process just waits until it gets the resource, and then it is processed.

A slightly different variant to break the hold and wait condition is to make the process drop all of the resources it is currently holding, whenever requesting for a new resource. Then it can grab hold of all the required resources again. This is also not an optimal use of processing power.

Facing the No Preemption condition

Facing this condition is also a possibility to avoid deadlocks. Take, for example, the printer as the resource. If a process has been assigned the printer and is in the middle of printing its output, forcibly taking away the printer because a needed plotter isn't readily available is tricky at best and impossible at worst. However, some resources can be virtualized to avoid this situation. Spooling printer output to the disk and allowing only the printer daemon access to the real printer eliminates deadlocks regarding the printer, but creating one for disk space.

Facing the Circular Wait condition

To avoid the circular wait, resources may be ordered, and we can ensure that each process can request resources only in increasing order of these numbers. The algorithm may itself increase complexity and may also lead to poor resource utilization.

Deadlock Avoidance

To avoid deadlocks, we can make use of prior knowledge about the usage of resources by processes, including resources available, resources allocated, future requests, and future releases by processes. Most deadlock avoidance algorithms need every process to tell in advance the maximum number of resources of each type that it may need. Based on all information, we may decide if a process should wait for a resource or not, and thus avoid chances for the circular wait.

If a system is already in a safe state, we can try to stay away from an unsafe state and avoid deadlock. Deadlocks cannot be avoided in an unsafe state. A system can be considered to be in a safe state if it is not in a state of deadlock and can allocate resources to the maximum available. A safe sequence of processes and allocation of resources ensures a safe state. Deadlock avoidance algorithms try not to allocate resources to a process if it makes the system go into an unsafe state.

Banker's Algorithm to avoid Resource Deadlocks

Banker's algorithm was written by Dijkstra. It is a scheduling algorithm and is an extension of the deadlock detection algorithm.

How does the banker's algorithm work?

- Consider a banker going around town giving loans to customers.
- If granting a loan to a customer leads to an unsafe state, it is denied.
- If granting a loan to a customer leads to a safe state, it is carried out.

In this analogy, the loan is the number of resources, the customers are processes, and the banker is the operating system.

Now imagine a scenario where the banker has ten units of a loan (10 resources). There are four customers (processes), and they have all specified their maximum required units of the loan (resources).

Take a look at this initial table.

Process	Current resources	Maximum resources required
A	0	6
B	0	5
C	0	4
D	0	7

Before the Operating System has granted any resource. The maximum number of combined resources is 22, but the banker has only 10 to loan out. Luckily for us, these processes don't need all the resources at once. So we can allocate some resources to them unless they get pushed to an unsafe state.

Now take a look at this table, here we have allocated some resources already. 8 resources, to be precise.

Process	Current resources	Maximum resources required
A	1	6
B	1	5
C	2	4
D	4	7

The banker still has two resources to give out. How is this still a safe state? Suppose process A has asked for the remaining five resources. But the banker has only two remaining. So process A gets delayed, and the banker waits for process C to finish running, keeping the two unallocated resources with itself. This is because the banker does not know whether process C is going to ask for more or free up the resources. And it has to be ready when and if process C asks for two resources to complete running.

Once process C is finished, the banker has four resources with it now, and it can use these resources to complete other processes.

Let's take another look at an unsafe state when the banker takes some risk.

Process	Current resources	Maximum resources required
A	1	6
B	2	5
C	2	4
D	4	7
...		

[A WordPress.com Website.](#)

B.C.A study.

Unit-4: Device management

Device management in operating system known as the management of the I/O devices such as a keyboard, magnetic tape, disk, printer, microphone, USB ports, scanner, etc.as well as the supporting units like control channels.

Technique of device management in the operating system

An operating system or the OS manages communication with the devices through their respective drivers. The operating system component provides a uniform interface to access devices of varied physical attributes. For device management in operating system:

- Keep tracks of all devices and the program which is responsible to perform this is called I/O controller.
- Monitoring the status of each device such as storage drivers, printers and other peripheral devices.
- Enforcing preset policies and taking a decision which process gets the device when and for how long.
- Allocates and Deallocates the device in an efficient way.De-allocating them at two levels: at the process level when I/O command has been executed and the device is temporarily released, and at the job level, when the job is finished and the device is permanently released.
- Optimizes the performance of individual devices.

Types of devices

The OS peripheral devices can be categorized into 3: Dedicated, Shared, and Virtual. The differences among them are the functions of the characteristics of the devices as well as how they are managed by the Device Manager.

Dedicated devices:-

Such type of devices in the **device management in operating system** are dedicated or assigned to only one job at a time until that job releases them. Devices like printers, tape drivers, plotters etc. demand such allocation scheme since it would be awkward if several users share them at the same point of time. The disadvantages of such kind of devices is the inefficiency resulting from the allocation of the device to a single user for the entire duration of job execution even though the device is not put to use 100% of the time.

Shared devices:-

These devices can be allocated to several processes. Disk-DASD can be shared among several processes at the same time by interleaving their requests. The interleaving is carefully controlled by the Device Manager and all issues must be resolved on the basis of predetermined policies.

Virtual Devices:-

These devices are the combination of the first two types and they are dedicated devices which are transformed into shared devices. For example, a printer converted into a shareable device via spooling program which re-routes all the print requests to a disk. A print job is not sent straight to the printer, instead, it goes to the disk(spool) until it is fully prepared with all the necessary sequences and formatting, then it goes to the printers. This technique can transform one printer into several virtual printers which leads to better performance and use.

input/output devices

input/Output devices are the devices that are responsible for the input/output operations in a computer system.

Basically there are following two types of input/output devices:

- Block devices
- Character devices

Block Devices

A block device stores information in block with fixed-size and own-address.

It is possible to read/write each and every block independently in case of block device.

In case of disk, it is always possible to seek another cylinder and then wait for required block to rotate under head without mattering where the arm currently is. Therefore, disk is a block addressable device.

Character Devices

A character device accepts/delivers a stream of characters without regarding to any block structure.

Character device isn't addressable.

Character device doesn't have any seek operation.

There are too many character devices present in a computer system such as printer, mice, rats, network interfaces etc. These four are the common character devices.

Input/Output Devices Examples

Here are the list of some most popular and common input/output devices:

- Keyboard
- Mouse
- Monitor
- Modem
- Scanner
- Laser Printer
- Ethernet
- Disk

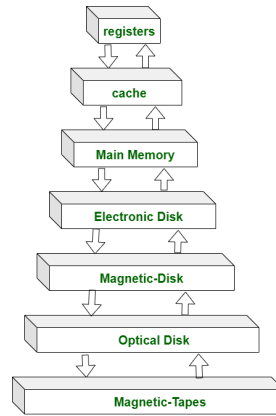
Storage devices

There are two types of storage devices:-

- **Volatile Storage Device –**
It loses its contents when the power of the device is removed.
- **Non-Volatile Storage device –**
It does not lose its contents when the power is removed. It holds all the data when the power is removed.

Secondary Storage is used as an extension of main memory. Secondary storage devices can hold the data permanently.

Storage devices consists of Registers, Cache, Main-Memory, Electronic-Disk, Magnetic-Disk, Optical-Disk, Magnetic-Tapes. Each storage system provides the basic system of storing a datum and of holding the datum until it is retrieved at a later time. All the storage devices differ in speed, cost, size and volatility. The most common Secondary-storage device is a Magnetic-disk, which provides storage for both programs and data.



Storage Device Hierarchy

In this hierarchy all the storage devices are arranged according to speed and cost. The higher levels are expensive, but they are fast. As we move down the hierarchy, the cost per bit generally decreases, where as the access time generally increases.

The storage systems above the Electronic disk are Volatile, where as those below are Non-Volatile. An Electronic disk can be either designed to be either Volatile or Non-Volatile. During normal operation, the electronic disk stores data in a large DRAM array, which is Volatile. But many electronic disk devices contain a hidden magnetic hard disk and a battery for backup power. If external power is interrupted, the electronic disk controller copies the data from RAM to the magnetic disk. When external power is restored, the controller copies the data back into the RAM.

The design of a complete memory system must balance all the factors. It must use only as much expensive memory as necessary while providing as much inexpensive, Non-Volatile memory as possible. Caches can be installed to improve performance where a large access-time or transfer-rate disparity exists between two components.

Buffering

A buffer is a memory area that stores data being transferred between two devices or between a device and an application.

Uses of I/O Buffering :

- Buffering is done to deal effectively with a speed mismatch between the producer and consumer of the data stream.

- A buffer is produced in main memory to heap up the bytes received from modem.
- After receiving the data in the buffer, the data get transferred to disk from buffer in a single operation.
- This process of data transfer is not instantaneous, therefore the modem needs another buffer in order to store additional incoming data.
- When the first buffer got filled, then it is requested to transfer the data to disk.
- The modem then starts filling the additional incoming data in the second buffer while the data in the first buffer getting transferred to disk.
- When both the buffers completed their tasks, then the modem switches back to the first buffer while the data from the second buffer get transferred to the disk.
- The use of two buffers disintegrates the producer and the consumer of the data, thus minimizes the time requirements between them.
- Buffering also provides variations for devices that have different data transfer sizes.

Types of various I/O buffering techniques :

1. Single buffer :

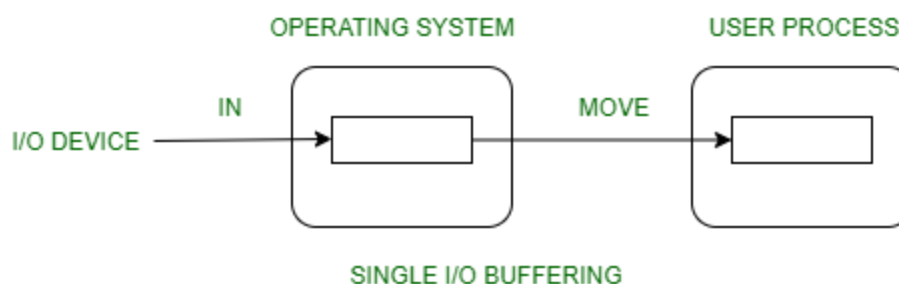
A buffer is provided by the operating system to the system portion of the main memory.

Block oriented device –

- System buffer takes the input.
- After taking the input, the block gets transferred to the user space by the process and then the process requests for another block.
- Two blocks works simultaneously, when one block of data is processed by the user process, the next block is being read in.
- OS can swap the processes.
- OS can record the data of system buffer to user processes.

Stream oriented device –

- Line- at a time operation is used for scroll made terminals. User inputs one line at a time, with a carriage return signaling at the end of a line.
- Byte-at a time operation is used on forms mode, terminals when each keystroke is significant.



2. Double buffer :

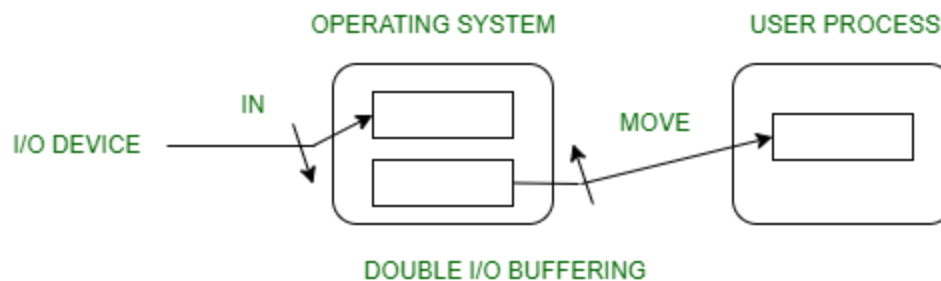
Block oriented –

- There are two buffers in the system.

- One buffer is used by the driver or controller to store data while waiting for it to be taken by higher level of the hierarchy.
- Other buffer is used to store data from the lower level module.
- Double buffering is also known as buffer swapping.
- A major disadvantage of double buffering is that the complexity of the process get increased.
- If the process performs rapid bursts of I/O, then using double buffering may be deficient.

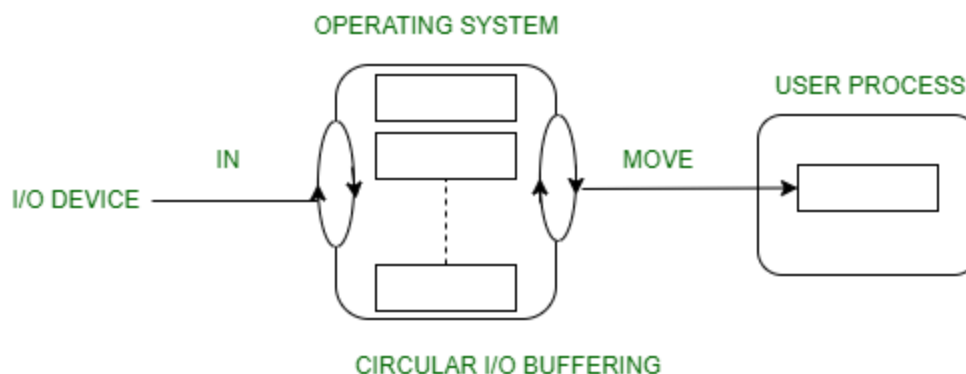
Stream oriented –

- Line- at a time I/O, the user process need not be suspended for input or output, unless process runs ahead of the double buffer.
- Byte- at a time operations, double buffer offers no advantage over a single buffer of twice the length.



3. Circular buffer :

- When more than two buffers are used, the collection of buffers is itself referred to as a circular buffer.
- In this, the data do not directly passed from the producer to the consumer because the data would change due to overwriting of buffers before they had been consumed.
- The producer can only fill up to buffer i-1 while data in buffer i is waiting to be consumed.



Secondary storage structure

Secondary storage devices are those devices whose memory is non volatile, meaning, the stored data will be intact even if the system is turned off. Here are a few things worth noting about secondary storage.

- Secondary storage is also called auxiliary storage.

- Secondary storage is less expensive when compared to primary memory like RAMs.
 - The speed of the secondary storage is also lesser than that of primary storage.
 - Hence, the data which is less frequently accessed is kept in the secondary storage.
 - A few examples are magnetic disks, magnetic tapes, removable thumb drives etc.
-

Magnetic Disk Structure

In modern computers, most of the secondary storage is in the form of magnetic disks. Hence, knowing the structure of a magnetic disk is necessary to understand how the data in the disk is accessed by the computer.



Structure of a magnetic disk

A magnetic disk contains several **platters**. Each platter is divided into circular shaped **tracks**. The length of the tracks near the centre is less than the length of the tracks farther from the centre. Each track is further divided into **sectors**, as shown in the figure.

Tracks of the same distance from centre form a cylinder. A read-write head is used to read data from a sector of the magnetic disk.

The speed of the disk is measured as two parts:

- **Transfer rate:** This is the rate at which the data moves from disk to the computer.
- **Random access time:** It is the sum of the seek time and rotational latency.

Seek time is the time taken by the arm to move to the required track. **Rotational latency** is defined as the time taken by the arm to reach the required sector in the track.

Even though the disk is arranged as sectors and tracks physically, the data is logically arranged and addressed as an array of blocks of fixed size. The size of a block can be **512** or **1024** bytes. Each logical block is mapped with a sector on the disk, sequentially. In this way, each sector in the disk will have a logical address.

Disk Scheduling Algorithms

On a typical multiprogramming system, there will usually be multiple disk access requests at any point of time. So those requests must be scheduled to achieve good efficiency. Disk scheduling is similar to process scheduling. Some of the disk scheduling algorithms are described below.

First Come First Serve

This algorithm performs requests in the same order asked by the system. Let's take an example where the queue has the following requests with cylinder numbers as follows:

98, 183, 37, 122, 14, 124, 65, 67

Assume the head is initially at cylinder **56**. The head moves in the given order in the queue i.e., **56→98→183→...→67**.

 First Come First Serve

Shortest Seek Time First (SSTF)

Here the position which is closest to the current head position is chosen first. Consider the previous example where disk queue looks like,

98, 183, 37, 122, 14, 124, 65, 67

Assume the head is initially at cylinder **56**. The next closest cylinder to **56** is **65**, and then the next nearest one is **67**, then **37, 14**, so on.

 Shortest Seek Time First

SCAN algorithm

This algorithm is also called the elevator algorithm because of its behavior. Here, first the head moves in a direction (say backward) and covers all the requests in the path. Then it moves in the opposite direction and covers the remaining requests in the path. This behavior is similar to that of an elevator. Let's take the previous example,

98, 183, 37, 122, 14, 124, 65, 67

Assume the head is initially at cylinder **56**. The head moves in backward direction and accesses **37** and **14**. Then it goes in the opposite direction and accesses the cylinders as they come in the path.

 SCAN algorithm

Disk management

The operating system is responsible for several aspects of disk management.

Disk Formatting

A new magnetic disk is a blank slate. It is just platters of a magnetic recording material. Before a disk can store data, it must be divided into sectors that the disk controller can read and write. This process is called low-level formatting (or **physical formatting**).

Low-level formatting fills the disk with a special data structure for each sector. The data structure for a sector consists of a header, a data area, and a trailer. The header and trailer contain information used by the disk controller, such as a sector number and an **error-correcting code (ECC)**.

To use a disk to hold files, the operating system still needs to record its own data structures on the disk. It does so in two steps. The first step is to **partition** the disk into one or more groups of cylinders. The operating system can treat each partition as though it were a separate disk. For instance, one partition can hold a copy of the operating system's executable code, while another holds user files. After partitioning, the second step is **logical formatting** (or creation of a file system). In this step, the operating system stores the initial file-system data structures onto the disk.

Boot block

When a computer is powered up or rebooted, it needs to have an initial program to run. This initial program is called the bootstrap program. It initializes all aspects of the system (i.e. from CPU registers to device controllers and the contents of main memory) and then starts the operating system.

To do its job, the bootstrap program finds the operating system kernel on disk, loads that kernel into memory, and jumps to an initial address to begin the operating-system execution.

For most computers, the bootstrap is stored in read-only memory (**ROM**). This location is convenient because ROM needs no initialization and is at a fixed location that the processor can start executing when powered up or reset. And since ROM is read-only, it cannot be infected by a computer virus. The problem is that changing this bootstrap code requires changing the ROM hardware chips.

For this reason, most systems store a tiny bootstrap loader program in the boot ROM, whose only job is to bring in a full bootstrap program from disk. The full bootstrap program can be changed easily: A new version is simply written onto the disk. The full bootstrap program is stored in a partition (at a fixed location on the disk) is called **the boot blocks**. A disk that has a boot partition is called a **boot disk or system disk**.

Bad Blocks

Since disks have moving parts and small tolerances, they are prone to failure. Sometimes the failure is complete, and the disk needs to be replaced, and its contents restored from backup media to the new disk.

More frequently, one or more sectors become defective. Most disks even come from the factory with bad blocks. Depending on the disk and controller in use, these blocks are handled in a variety of ways.

The controller maintains a list of bad blocks on the disk. The list is initialized during the low-level format at the factory and is updated over the life of the disk. The controller can be told to replace each bad sector logically with one of the spare sectors. This scheme is known as sector sparing or forwarding.

Swap-space management

Swapping is a memory management technique used in multi-programming to increase the number of process sharing the CPU. It is a technique of removing a process from main memory and storing it into secondary memory, and then bringing it back into main memory for continued execution. This action of moving a process out from main memory to secondary memory is called **Swap Out** and the action of moving a process out from secondary memory to main memory is called **Swap In**.

Swap-Space :

The area on the disk where the swapped out processes are stored is called swap space.

Swap-Space Management :

Swap-Space management is another low-level task of the operating system. Disk space is used as an extension of main memory by the virtual memory. As we know the fact that disk access is much slower than memory access, In the swap-space management we are using disk space, so it will significantly decrease system performance. Basically, in all our systems we require the best throughput, so the goal of this swap-space implementation is to provide the virtual memory the best throughput.

Swap-Space Use :

Swap-space is used by the different operating-system in various ways. The systems which are implementing swapping may use swap space to hold the entire process which may include image, code and data segments. Paging systems may simply store pages that have been pushed out of the main memory. The need of swap space on a system can vary from a megabytes to gigabytes but it also depends on the amount of physical memory, the virtual memory it is backing and the way in which it is using the virtual memory.

A swap space can reside in one of the two places –

1. Normal file system
2. Separate disk partition

Let, if the swap-space is simply a large file within the file system. To create it, name it and allocate its space **normal file-system** routines can be used. This approach, though easy to implement, is inefficient. Navigating the directory structures and the disk-allocation data structures takes time and extra disk

access. During reading or writing of a process image, **external fragmentation** can greatly increase swapping times by forcing multiple seeks.

There is also an alternate to create the swap space which is in a separate **raw partition**. There is no presence of any file system in this place. Rather, a swap space storage manager is used to allocate and de-allocate the blocks from the raw partition. It uses the algorithms for speed rather than storage efficiency, because we know the access time of swap space is shorter than the file system. By this **Internal fragmentation** increases, but it is acceptable, because the life span of the swap space is shorter than the files in the file system. Raw partition approach creates fixed amount of swap space in case of the **disk partitioning**.

Some operating systems are flexible and can swap both in raw partitions and in the file system space, example: **Linux**.

Disk reliability



[A WordPress.com Website.](#)

B.C.A study

Unit-5: Information management

Information management in operating systems refers to the process of collecting, organizing, and storing data in a structured and efficient manner. It is an essential aspect of any operating system as it plays a crucial role in maintaining the stability, security, and performance of the system. The goal of information management in operating systems is to ensure that the data is easily accessible, retrievable, and secure.

The operating system is responsible for managing different types of data such as system configurations, user files, applications, and system logs. The information management system in an operating system includes various components such as storage management, file systems, and access control mechanisms.

Storage management is the process of managing the physical storage devices and memory in the system. This includes managing the allocation and deallocation of storage space, ensuring data consistency and integrity, and managing the storage resources to maximize efficiency and performance.

File systems are the structures used to store and organize data in the operating system. It determines the layout of files and directories on the storage devices and provides the mechanism for accessing and retrieving the data. Different file systems have different features, such as support for large files, security, and reliability.

Access control mechanisms are used to manage user access to the data and system resources. This includes setting permissions for users and groups, managing authentication and authorization processes, and enforcing access policies.

In conclusion, information management in operating systems is a critical aspect of any operating system as it affects the stability, security, and performance of the system. By managing the storage, file systems, and access control mechanisms, the operating system can ensure that data is easily accessible, retrievable, and secure.

A Simple File system

A Simple File System (SFS) is a basic file system that is used to store and manage data on a storage device. It is a hierarchical file system that organizes data in a tree-like structure of directories and files. The main purpose of SFS is to provide an efficient and reliable method for organizing and storing data.

In an SFS, the root directory is the top-level directory that contains all other directories and files. Directories within the root directory are called subdirectories and can contain further subdirectories or files. The files in the system can be any type of data, such as text files, images, or audio files.

SFS uses file attributes to store information about each file, such as the file name, creation date, size, and ownership. It also uses file descriptors to manage the access and retrieval of data. The file descriptors are unique numbers assigned to each file and used to reference the file in the system.

SFS uses a file allocation table (FAT) to manage the allocation and deallocation of storage space. The FAT is a table that contains information about the location of each file on the storage device. When a file is created, the operating system allocates a portion of the storage space to the file and updates the FAT. When the file is deleted, the operating system frees up the storage space and updates the FAT accordingly.

In conclusion, SFS is a simple and efficient file system that provides a basic mechanism for organizing and storing data. It is suitable for small systems with limited storage requirements, and it is easy to implement and maintain. SFS is a good starting point for understanding file systems and can be used as a foundation for more advanced file systems.

General Model of a File System

A file system is a way of organizing and storing data on a storage medium (e.g. a hard disk, flash drive, etc.) so that it can be easily retrieved, updated and managed. The general model of a file system consists of the following components:

1. **Storage space:** This is the physical space on the storage medium where the data is stored.
2. **Files:** A file is a collection of data that is stored on the storage medium. Files can be of different types (e.g. text files, image files, audio files, etc.) and have different attributes (e.g. name, size, date of creation, etc.).
3. **Directories:** A directory is a special type of file that contains a list of other files and/or directories. Directories allow users to organize files into a hierarchical structure.
4. **File names:** A file name is a unique identifier that is assigned to each file in the file system. File names can be either simple (e.g. "file.txt") or hierarchical (e.g. "/dir1/dir2/file.txt").
5. **File metadata:** This is data that provides information about a file (e.g. its size, date of creation, etc.).
6. **File access methods:** These are the methods used to read and write data to and from files.

7. Allocation methods: These are the methods used to allocate storage space on the storage medium for files.

The general model of a file system provides a framework for organizing and managing data on a storage medium in a way that makes it easily accessible, retrievable and manageable

Symbolic File System

A symbolic file system, also known as a symbolic link or symlink, is a special type of file in a file system that acts as a pointer to another file or directory. Unlike a hard link, which is a direct link to the actual file, a symbolic link is a separate file that contains a reference to the original file.

When a program accesses a symbolic link, the operating system redirects the program to the file or directory that the link is pointing to, as if the program were accessing the original file directly. This allows programs to access the linked file or directory without having to know its actual location.

Symbolic links are useful for several purposes, including:

1. Creating aliases for files or directories: A symbolic link can be used to create an alias for a file or directory, allowing it to be accessed from multiple locations without having to copy the file or directory to each location.
2. Redirecting a file or directory to a new location: If a file or directory is moved to a new location, a symbolic link can be created in its original location that points to its new location, allowing programs that access the file or directory through the original location to continue working without modification.
3. Implementing file or directory hierarchies: A symbolic link can be used to create a file or directory hierarchy, where a link at one location points to a file or directory at another location

Basic File System

A file system is a way of organizing and storing digital information on a storage device, such as a hard drive or solid-state drive. The file system provides a way to name, store, retrieve, and manipulate the stored information.

There are several types of file systems, including:

1. FAT (File Allocation Table): This is an old file system that was used on floppy disks and older versions of Windows. It has limited capabilities and is not widely used today.
2. NTFS (New Technology File System): This is the file system used by recent versions of Windows. It provides advanced features such as file and folder permissions, encryption, and support for large storage devices.

3. exFAT (Extended File Allocation Table): This is a newer file system that is used for removable storage devices, such as memory cards and USB drives. It provides similar features to NTFS but is optimized for use on smaller, removable storage devices.
4. HFS (Hierarchical File System): This is the file system used by Apple's Mac OS X. It provides features similar to those found in NTFS, but is optimized for use with Mac OS X.
5. Ext2/Ext3/Ext4: These are file systems used by Linux. Ext4 is the latest and most widely used file system for Linux.

A file system is responsible for several key tasks, including:

1. Allocating space on the storage device to new files and directories.
2. Keeping track of which areas of the storage device are being used and which are free.
3. Ensuring that information is stored in a way that is easily recoverable in the event of a crash or other failure.
4. Providing a way for the operating system to access and manipulate the stored information.

Access control verification

Access control verification is the process of verifying that a user has the proper authorization to access a particular resource or system. This process is essential in ensuring the security and confidentiality of sensitive information.

Access control verification is typically performed using some combination of user authentication and authorization. User authentication is the process of verifying the identity of the user, such as by requiring a password or smart card. Authorization is the process of determining if the authenticated user is allowed to access the requested resource or system.

There are several different methods for implementing access control verification, including:

1. Role-based access control: Users are assigned to roles, and access to resources is granted based on the roles.
2. Rule-based access control: Access to resources is determined by a set of rules or conditions, such as the time of day or the user's location.
3. Discretionary access control: Access to resources is controlled by the owner of the resource, who grants access to specific users.
4. Mandatory access control: Access to resources is determined by security labels assigned to both the user and the resource, and access is only granted if the security labels match.

In summary, access control verification is an important aspect of security that helps to ensure that only authorized users can access sensitive information and systems. The method used for access control verification will depend on the specific security requirements and the resources being protected.

Logical File System

A Logical File System (LFS) is an abstract way of organizing and accessing data in a computer file system. It's a way of viewing and manipulating the underlying physical storage, such as hard drives, as a collection of files and directories, rather than just a block of raw storage.

In an LFS, files are organized into a hierarchical structure of directories and subdirectories, allowing for easy navigation and organization of data. The LFS provides a higher-level interface for managing files and directories, including operations like creating, reading, writing, and deleting files, as well as creating and removing directories.

One of the key features of an LFS is the ability to use file names and paths, which makes it easier for users to locate and access their files. The LFS also provides a mechanism for tracking changes to files and directories, such as updates or deletions, and for managing access control to prevent unauthorized access.

LFSs are implemented in operating systems, and are a fundamental component of modern computing. Some examples of LFSs include the file system used by Windows (NTFS), the file system used by Unix-based systems (such as ext4), and the file system used by Apple's macOS (HFS+).

Physical File system File System Interface

A Physical File System (PFS) is a low-level interface to the actual physical storage devices, such as hard drives, solid-state drives (SSDs), and flash memory. The PFS is responsible for reading and writing data to the physical storage media and for handling issues such as storage allocation, fragmentation, and wear leveling.

The Physical File System Interface is the set of operations and data structures that a file system uses to communicate with the physical storage media. This interface provides a way for the file system to interact with the physical storage devices, such as reading and writing data blocks, as well as performing maintenance operations such as error checking and correction, and wear leveling.

The Physical File System Interface is typically implemented by the storage controller, which is responsible for managing the physical storage media. This interface may include operations such as reading and writing blocks of data, as well as management operations such as formatting the storage media, configuring partitions, and checking the health of the storage devices.

The Physical File System Interface is an important component of a computer system, as it enables the file system to access the physical storage devices and manage the data stored on them. A well-designed Physical File System Interface can improve performance, reliability, and overall system stability by providing a low-level and efficient way of accessing the physical storage media

File Concept

A file is a collection of data that is stored in a computer's memory or on a storage device such as a hard drive, flash drive, or cloud storage service. Files can contain text, images, audio, video, or any other type of digital data.

The file concept is fundamental to modern computing and provides a convenient way to organize and access data. Files are usually given a name, which makes it easier for users to locate and access them. They can also be organized into directories and subdirectories, allowing for easy navigation and management of data.

Each file has a type or format that determines how the data it contains is organized and how it can be used. For example, a text file might contain plain text, while an image file might contain binary data that represents an image. Different types of files can be opened and edited with different software applications, such as a text editor for text files, or an image editor for image files.

In a computer's file system, files are stored as a series of bits and bytes, which represent the data they contain. The file system is responsible for managing the files, including allocating space for new files, updating existing files, and deleting files that are no longer needed.

Overall, the file concept is an important part of modern computing, providing a convenient and efficient way to store and manage data.

Access Methods

Access Methods are the techniques and algorithms used by computer systems to retrieve data stored in a database, file system, or other type of data storage. Access methods determine the way in which data is stored and organized, as well as the way in which it can be retrieved.

There are several different types of access methods, each with its own advantages and disadvantages, and each suited to different types of data and usage patterns. Some common access methods include:

1. **Sequential Access:** Data is stored and retrieved in a linear, sequential manner, similar to reading a book from front to back. This method is typically used for large data files where the data must be read in a specific order, such as log files.
2. **Direct Access:** Data is stored and retrieved based on its physical location on the storage device, allowing for fast access to specific data. This method is used for systems where data needs to be retrieved quickly, such as real-time systems.
3. **Indexed Access:** Data is stored and retrieved using an index, which acts as a lookup table to find the location of specific data in the storage system. This method is commonly used for databases, where data is organized into tables and indexes are used to find specific records quickly.

4. Hash-Based Access: Data is stored and retrieved using a hash function, which converts the data into a unique value that can be used to index and retrieve the data. This method is used for systems where data must be retrieved quickly, and where data is frequently searched for.

Access methods play a critical role in information management, as they determine the performance, reliability, and scalability of a system. The choice of access method depends on the type of data being stored, the usage patterns, and the performance requirements of the system.

Directory Structure

A directory structure is the way files and folders are organized and stored on a computer's file system. It provides a way to organize and categorize files and folders into a hierarchical structure, making it easier to find and access data.

Here's an example of a directory structure:

```
/
home/
user1/
documents/
resume.txt
cover_letter.txt
pictures/
vacation/
hawaii.jpg
grand_canyon.jpg
profile_pic.jpg
user2/
documents/
budget.xlsx
presentation.ppt
music/
pop/
justin_bieber.mp3
ariana_grande.mp3
rock/
metallica.mp3
acdc.mp3
etc/
config_files/
apache2.conf
ssh_config
tmp/
```

In this example, the root directory (/) contains several subdirectories, including `home`, `etc`, and `tmp`. The `home` directory contains user-specific data, with separate directories for `user1` and `user2`. Each user has a `documents` directory and a `pictures` directory, which contain their personal files. The `user1` directory contains a `pictures` directory with two subdirectories: `vacation` and `profile_pic`.

The `etc` directory contains system-wide configuration files, and the `tmp` directory is used for temporary storage.

This example demonstrates how a directory structure can be used to organize data into a hierarchical structure, making it easier to locate and access data. By categorizing and organizing data into meaningful directories, users can quickly find the files they need, even if they have a large number of files and folders stored on their computer

Protection

Protection in information management refers to the measures taken to ensure the confidentiality, integrity, and availability of data stored in a computer system or network. The goal of protection is to prevent unauthorized access, modification, or destruction of data, and to ensure that data is available when needed.

There are several ways to protect information:

1. **Access Control:** Access control refers to the measures taken to ensure that only authorized users can access specific data. This can include setting permissions on files and folders, using passwords or biometric authentication, and implementing firewalls to control access to a network.
2. **Encryption:** Encryption is the process of converting data into a form that can only be deciphered by authorized users. Encryption can be used to protect data while it is stored, transmitted, or processed.
3. **Backup and Recovery:** Backup and recovery refers to the process of creating and storing copies of data to ensure that it can be recovered in the event of a disaster or data loss. This includes regular backups of data to an external storage device, as well as disaster recovery plans to ensure that critical systems and data can be restored in the event of an outage.
4. **Physical Security:** Physical security refers to the measures taken to protect computer systems and data from theft, damage, or unauthorized access. This can include secure server rooms, locks and security cameras, and other physical measures to prevent unauthorized access to computer systems.
5. **Network Security:** Network security refers to the measures taken to protect a network from unauthorized access, malicious attacks, and data theft. This can include firewalls, intrusion detection systems, and encryption to protect data as it is transmitted over the network

Consistency Semantics File System Implementation

Consistency semantics in a file system refers to the guarantees about the state of the file system and the order of operations. There are several consistency models that a file system can implement, including:

1. Strong consistency: All operations on the file system appear to occur atomically and in the order they were issued.
2. Eventual consistency: After a sufficient amount of time, all nodes in the file system will eventually agree on the state of the data.
3. Causal consistency: Operations that are causally related are guaranteed to be seen in the same order by all nodes in the file system.

Implementing consistency semantics in a file system typically involves techniques such as locking, versioning, and distributed consensus protocols. The specific implementation will depend on the desired consistency guarantees and the underlying architecture of the file system

File – System Structure

A file system structure is the way that files and directories are organized on a storage device.

Here is an example of a file system structure

```
/
|- home/
|  |- user1/
|    |- Documents/
|      |- resume.doc
|      |- letter.pdf
|      |- Pictures/
|        |- vacation.jpg
|        |- Music/
|          |- playlist.mp3
|    |- user2/
|      |- Documents/
|        |- budget.xls
|        |- Pictures/
|          |- family.jpg
|- etc/
|- var/
|- bin/
```

In this example, the root directory / contains three subdirectories: `home/`, `etc/`, `var/`, and `bin/`. The `home/` directory contains two subdirectories, `user1/` and `user2/`, each of which represents a different user's home directory. Each user's home directory contains a `Documents/` directory, a `Pictures/` directory, and a `Music/` directory. The `Documents/` directory contains various documents, such as `resume.doc` and `letter.pdf`, while the `Pictures/` directory contains pictures such as `vacation.jpg` and `family.jpg`. The `Music/` directory contains audio files such as `playlist.mp3`.

File Allocation Methods

There are different kinds of methods that are used to allocate disk space. We must select the best method for the file allocation because it will directly affect the system performance and system efficiency. With the help of the allocation method, we can utilize the disk, and also files can be accessed.

There are various types of file allocations method:

1. Contiguous allocation
2. Extents
3. Linked allocation
4. Clustering
5. FAT
6. Indexed allocation
7. Linked Indexed allocation
8. Multilevel Indexed allocation
9. Inode

There are different types of file allocation methods, but we mainly use three types of file allocation methods:

1. Contiguous allocation
2. Linked list allocation
3. Indexed allocation

These methods provide quick access to the file blocks and also the utilization of disk space in an efficient manner.

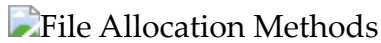
Contiguous Allocation: – Contiguous allocation is one of the most used methods for allocation. Contiguous allocation means we allocate the block in such a manner, so that in the hard disk, all the blocks get the contiguous physical block.

We can see in the below figure that in the directory, we have three files. In the table, we have mentioned the starting block and the length of all the files. We can see in the table that for each file, we allocate a contiguous block.

 File Allocation Methods

Example of contiguous allocation

We can see in the given diagram, that there is a file. The name of the file is 'mail.' The file starts from the 19th block and the length of the file is 6. So, the file occupies 6 blocks in a contiguous manner. Thus, it will hold blocks 19, 20, 21, 22, 23, 24.



Advantages of Contiguous Allocation

The advantages of contiguous allocation are:

1. The contiguous allocation method gives excellent read performance.
2. Contiguous allocation is easy to implement.
3. The contiguous allocation method supports both types of file access methods that are sequential access and direct access.
4. The Contiguous allocation method is fast because, in this method number of seeks is less due to the contiguous allocation of file blocks.

Disadvantages of Contiguous allocation

The disadvantages of contiguous allocation method are:

1. In the contiguous allocation method, sometimes disk can be fragmented.
2. In this method, it is difficult to increase the size of the file due to the availability of the contiguous memory block.

Linked List Allocation

The linked list allocation method overcomes the drawbacks of the contiguous allocation method. In this file allocation method, each file is treated as a linked list of disks blocks. In the linked list allocation method, it is not required that disk blocks assigned to a specific file are in the contiguous order on the disk. The directory entry comprises of a pointer for starting file block and also for the ending file block. Each disk block that is allocated or assigned to a file consists of a pointer, and that pointer point the next block of the disk, which is allocated to the same file.

Example of linked list allocation

We can see in the below figure that we have a file named 'jeep.' The value of the start is 9. So, we have to start the allocation from the 9th block, and blocks are allocated in a random manner. The value of the end is 25. It means the allocation is finished on the 25th block. We can see in the below figure that the block (25) comprised of -1, which means a null pointer, and it will not point to another block.

 File Allocation Methods

Advantages of Linked list allocation

There are various advantages of linked list allocation:

1. In linked list allocation, there is no external fragmentation. Due to this, we can utilize the memory better.
2. In linked list allocation, a directory entry only comprises of the starting block address.
3. The linked allocation method is flexible because we can quickly increase the size of the file because, in this to allocate a file, we do not require a chunk of memory in a contiguous form.

Disadvantages of Linked list Allocation

There are various disadvantages of linked list allocation:

1. Linked list allocation does not support direct access or random access.
2. In linked list allocation, we need to traverse each block.
3. If the pointer in the linked list break in linked list allocation, then the file gets corrupted.
4. In the disk block for the pointer, it needs some extra space.

Indexed Allocation

The Indexed allocation method is another method that is used for file allocation. In the index allocation method, we have an additional block, and that block is known as the index block. For each file, there is an individual index block. In the index block, the *i*th entry holds the disk address of the *i*th file block. We can see in the below figure that the directory entry comprises of the address of the index block.

 File Allocation Methods

Advantages of Index Allocation

The advantages of index allocation are:

1. The index allocation method solves the problem of external fragmentation.
2. Index allocation provides direct access.

Disadvantages of Index Allocation

The disadvantages of index allocation are:

1. In index allocation, pointer overhead is more.
2. We can lose the entire file if an index block is not correct.
3. It is totally a wastage to create an index for a small file.

A single index block cannot hold all the pointer for files with large sizes.

To resolve this problem, there are various mechanism which we can use:

1. Linked scheme
2. Multilevel Index
3. Combined Scheme

1. **Linked Scheme:** – In the linked scheme, to hold the pointer, two or more than two index blocks are linked together. Each block contains the address of the next index block or a pointer.
2. **Multilevel Index:** – In the multilevel index, to point the second-level index block, we use a first-level index block that in turn points to the blocks of the disk, occupied by the file. We can extend this up to 3 or more than 3 levels depending on the maximum size of the file.
3. **Combined Scheme:** – In a combined scheme, there is a special block which is called an information node (Inode). The inode comprises of all the information related to the file like authority, name, size, etc. To store the disk block addresses that contain the actual file, the remaining space of inode is used. In inode, the starting pointer is used to point the direct blocks. This means the pointer comprises of the addresses of the disk blocks, which consist of the file data. To indicate the indirect blocks, the next few pointers are used. The indirect blocks are of three types, which are single indirect, double indirect, and triple indirect.

free space management

Free space management is the process by which a file system manages unused disk space to ensure that it is available for future file storage. Here is an example of how free space management works:

1. Initial allocation: When a file system is first created, it allocates a portion of the disk space to be used for file storage. The remaining space is considered free space.
2. File creation: When a new file is created, the file system allocates a portion of the free space to store the file's data. The allocated space is now considered used space, and the remaining free space is updated.
3. File deletion: When a file is deleted, the space that was previously used to store the file's data becomes free space again. The file system updates the free space information to reflect this change.
4. File expansion: If a file grows in size, the file system may need to allocate additional disk space to store the file's new data. This additional space is taken from the free space pool.
5. File contraction: If a file decreases in size, the file system may choose to release the unused space back to the free space pool.
6. Defragmentation: Over time, as files are created, deleted, and modified, the file system may become fragmented, with free space scattered across the disk. To ensure that the free space is available for future file storage, the file system may perform a defragmentation operation, which reorganizes the disk to consolidate the free space into larger contiguous blocks.

The specific methods used for free space management will depend on the file system being used and its requirements for disk space usage and efficiency. Some file systems use a bitmap to keep track of the used and free disk blocks, while others use a linked list or a tree structure



[A WordPress.com Website.](#)