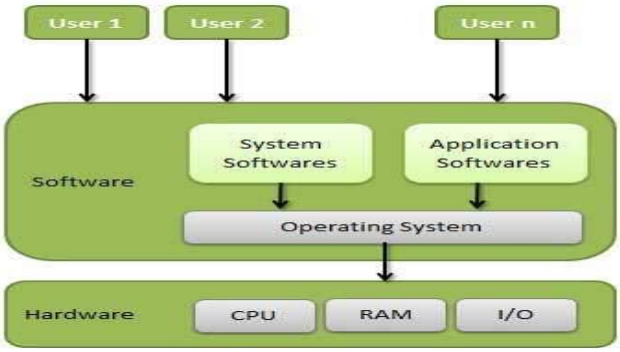
Operating system – Unit 1

An operating system (OS) is a collection of software that manages computer hardware resources and provides common services for computer programs. The operating system is a vital component of the system software in a computer system. An operating System (OS) is an intermediary between users and computer hardware. It provides users an environment in which a user can execute programs conveniently and efficiently. An operating System controls the allocation of resources and services such as memory, processors, devices and information. **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.



Following are some of the components of system

Memory Management: Memory management refers to management of Primary Memory or Main Memory. Main memory provides a fast storage that can be access directly by the CPU. So for a program to be executed, it must in the main memory.

Processor Management: In multiprogramming environment, OS decides which process gets the processor when and how much time. This function is called process scheduling.

Device Management: OS manages device communication via their respective drivers. Operating System keeps tracks of all devices. It decides which process gets the device when and for how much time. It also allocates the device in the efficient way and de-allocates devices.

File Management: Operating System keeps track of information, location, uses, status etc. The collective facilities are often known as file system. It decides who gets the resources. Allocates the resources and de-allocates the resources.

Other Important Activities of Operating System

Security -- By means of password and similar other techniques, preventing 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 software and users -- Coordination and assignment of compilers, interpreters, assemblers and other software to the various users of the computer systems.

User view: The user's view of the computer system varies according to the interface being used. Most computer users sit in front of a PC, consisting of a monitor, mouse, keyboard and a system unit. Such systems are designed for one user to monopolize its resources. In these type of systems, OS is designed to focus on ease of use. These systems don't pay attention to resource allocation because only one user can't busy CPU all the time. While in mainframes OS is design to focus on computing as compared to the time sharing (CPU Time) between different application programs and users because users have very less interaction with systems, mostly.

System view: From the computer's point of view, the operating system is the program most intimately involved with the hardware. We can view an operating system as a **resource allocator**. A computer system may have many resources to solve a problem: CPU time, memory space, I/O devices, and so on. The operating system acts as a manager of these resources. An operating system is a **control program**. A control program manages the execution of user programs to prevent errors and improper use of the computer resources. It is especially concerned with the operation and control of I/O devices.

What are the goals or objective of operating system ?

There are many goals or objective of an Operating system . But there are two main goals of Operating System –

1. Efficient use of a computer system:

- Use Computer hardware efficiently
- Allow sharing of hardware and software resource .
- Make application software portable and versatile .
- Improve overall system reliability such as error confinement , fault , tolerance reconfiguration.

2. User Convenience :

- Simplify the execution of user program and make solving user problems easily
- Provide isolation , security and protection among user programs.
- Good service means speedy response to computational requests.
- User friendly system , easy to use commands , graphical user interface(GUI)

Types of operating system

Operating systems are there from the very first computer generation. Operating systems keep evolving over the period of time. Following are few of the important types of operating system which are most commonly used.

Batch operating system

The users of 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. Thus, the programmers left their programs with the operator. The operator then sorts programs into batches with similar requirements. The problems with Batch Systems are following.

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

Multiprogramming operating system

A multiprogramming operating system is one that allows end-users to run more than one program at a time. The technology works by allowing the central processing unit (CPU) of a computer to switch between two or more running tasks when the CPU is idle.

A multiprogramming operating system acts by analyzing the current CPU activity in the computer. When the CPU is idle — when it is between tasks — it has the opportunity to use that downtime to run tasks for another program. In this way, the functions of several programs may be executed sequentially. For example, when the CPU is waiting for the end-user to enter numbers to be calculated, instead of being entirely idle, it may run load the components of a web page the user is accessing.

The main benefit of this functionality is that it can reduce wasted time in the system's operations. As in a business, efficiency is the key to generating the most profit from an enterprise. Using this type of operating system eliminates waste in the system by ensuring that the computer's CPU is running at maximum capacity more of the time.

The multiprogramming operating system has been largely replaced by a new generation of operating system known as multitasking operating systems. In a multitasking operating system, the system does not have to wait for the completion of a task before moving to work on an active program. Instead, it can interrupt a running program at any time in order to shift its CPU resources to a different active program. This provides for a more dynamic approach to handling concurrent programs.

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, objective is to maximize processor use, whereas in Time-Sharing Systems 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 receives an immediate response. Operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time.

Advantages of Timesharing operating systems are following

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

Disadvantages of Timesharing operating systems are following.

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

Real Time operating System

Real time system is defines 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 response time. So in this method response time is very less as compared to the 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. Real-time operating system has well-defined, fixed time constraints otherwise system will

fail.For example Scientific experiments, medical imaging systems, industrial control systems, weapon systems, robots, and home-applicance controllers, Air traffic control system etc.

Operating System Services

Operating System provides services to both the users and to the programs.

- It provides programs, an environment to execute.
- It provides users, services to execute the programs in a convenient manner.

Following are few common services provided by operating systems.

Program execution: Operating system handles many kinds of activities from user programs to system programs like printer spooler, name servers, file server etc. Each of these activities is encapsulated as a process.

A process includes the complete execution context (code to execute, data to manipulate, registers, OS resources in use). Following are the major activities of an operating system with respect to program management.

- Loads a program into memory.
- Executes the program.
- Handles program's execution.
- Provides a mechanism for process synchronization.
- Provides a mechanism for process communication.
- Provides a mechanism for deadlock handling.

I/O Operation: I/O subsystem comprised of I/O devices and their corresponding driver software. Drivers hides the peculiarities of specific hardware devices from the user as the device driver knows the peculiarities of the specific device.

Operating System manages the communication between user and device drivers. Following are the major activities of an operating system with respect to I/O Operation.

- I/O operation means read or write operation with any file or any specific I/O device.
- Program may require any I/O device while running.
- Operating system provides the access to the required I/O device when required.

File system manipulation: A file represents a collection of related information. Computer can store files on the disk (secondary storage), for long term storage purpose. Few examples of storage media are magnetic tape, magnetic disk and optical disk drives like CD, DVD. Each of these media has its own properties like speed, capacity, data transfer rate and data access methods.

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions. Following are the major activities of an operating system with respect to file management.

- Program needs to read a file or write a file.
- The operating system gives the permission to the program for operation on file.
- Permission varies from read-only, read-write, denied and so on.
- Operating System provides an interface to the user to create/delete files.
- Operating System provides an interface to the user to create/delete directories.
- Operating System provides an interface to create the backup of file system.

Communication: Following are the major activities of an operating system with respect to communication.

- Two processes often require data to be transferred between them.

- The both processes can be on the one computer or on different computer but are connected through computer network.
- Communication may be implemented by two methods either by Shared Memory or by Message Passing.

Error handling: Error can occur anytime and anywhere. Error may occur in CPU, in I/O devices or in the memory hardware. Following are the major activities of an operating system with respect to error handling.

- OS constantly remains aware of possible errors.
- OS takes the appropriate action to ensure correct and consistent computing.

Resource Management: In case of multi-user or multi-tasking environment, resources such as main memory, CPU cycles and files storage are to be allocated to each user or job. Following are the major activities of an operating system with respect to resource management.

- OS manages all kind of resources using schedulers.
- CPU scheduling algorithms are used for better utilization of CPU.

Protection: Considering a computer systems having multiple users the concurrent execution of multiple processes, then the various processes must be protected from each another's activities.

Protection refers to mechanism or a way to control the access of programs, processes, or users to the resources defined by a computer systems. Following are the major activities of an operating system with respect to protection.

- OS ensures that all access to system resources is controlled.
- OS ensures that external I/O devices are protected from invalid access attempts.
- OS provides authentication feature for each user by means of a passwo

System Calls

System calls provide an interface to the services made available by an operating system. These calls are generally available as routines written in C, C++, and assembly language instructions. Types of system calls are system calls can be grouped roughly into five major categories: process control, file management, device management, information maintenance and communications.

Process control: A running program needs to be able to halt its execution either normally (end) or abnormally (abort). If a system call is made to terminate the currently running program abnormally, or if the program runs in to a problem and causes an error trap, a dump of memory is taken and an error message generated.

File Management: we need to able to create and delete files.once file created, we need to open it, write, read, reposition and close the file.

Device Management: A process requires several resources to execute. If the resources are available, they can be granted, and the control can be returned to the user process. Otherwise, the process will have to wait until sufficient resources are available.

Information Maintenance: Many system calls exist simply for the purpose of transferring information between the user program and the operating system. Some system calls return information about the system, processes.

Communication: communication is essential for the processes to exchange information with one another. There are two models of communication, message passing and shared memory.

(also refer types of system calls given in the class)

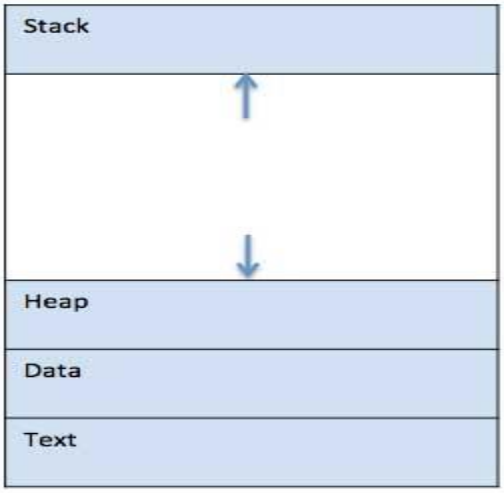
System program

A modern system is a collection of system programs. System program provide a convenient environment for program development and execution. Some of them are simply user interfaces to the system calls; others are more complex. They can be divided into these categories:

- File Management: these programs create, delete, copy, rename, print, dump, list and generally manipulate files and directories.
- Status Information: some programs ask the system for the date, time, amount of available memory or disk space, number of users and some other status information. That information is then formatted and is printed to the terminal or other output device or file.
- File Modification: several text editors may be available to create and modify the content of files stored on disk.
- Programming Language support: compilers, assemblers and interpreters for common programming languages are often provided to the user with the operating system. Some of these programs are now priced and provided separately.
- Program loading and execution: once a program is assembled or compiled, it must be loaded in to memory to be executed. The system may provide absolute loaders, relocatable loaders, linkage editors and overlay loaders. Debugging systems for either higher level languages or machine level languages are also needed.
- Communication: these programs provide the mechanism for creating virtual connections among processes, users and different computer systems. They allow users to send messages to one another's screens, to browse web pages, to send electronic mail messages, to log in remotely or to transfer files from one machine to another.

Process

A process is 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 −

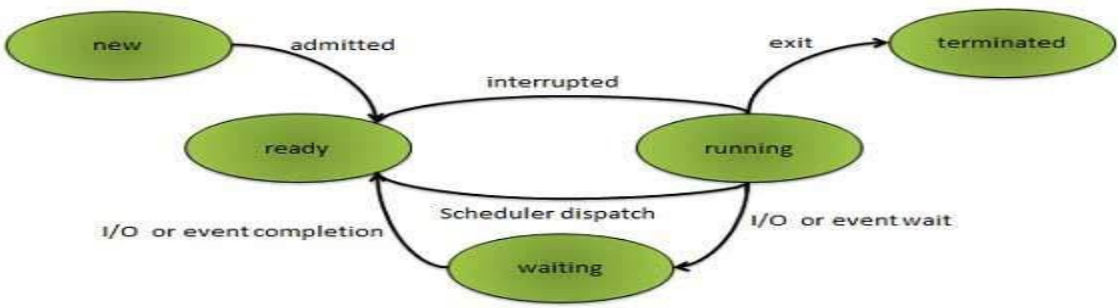


Component & Description
Stack: The process Stack contains the temporary data such as method/function parameters, return address and local variables.
Heap: This is dynamically allocated memory to a process during its run time.
Text: This includes the current activity represented by the value of Program Counter and the contents of the processor's registers.
Data: This section contains the global and static variables.

Process States (life cycle of a process)

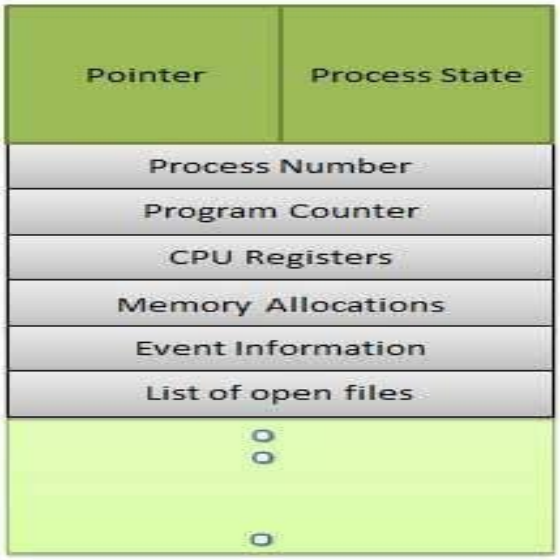
As a process executes, it changes state. The state of a process is defined as the current activity of the process. Process can have one of the following five states at a time.

New: The process is being created.
Ready: The process is waiting to be assigned to a processor. When the process is allocated with the processor, it will move to running state and begins its execution.
Running: Process instructions are being executed (i.e. The process that is currently being executed). While executing the process may get interrupted and moved to ready state or the process may require I/O devices and moved to wait state.
Waiting: The process is waiting for some event to occur (such as the completion of an I/O operation). Once if the process completes I/O or event, it will be moved to ready state to continue its execution.
Terminated: when the process has finished its execution, it will be terminated.



Process Control Block (PCB)

Each process is represented in the operating system by a process control block (PCB) also called a task control block. PCB is the data structure used by the operating system. Operating system groups all information that needs about particular process. PCB contains many pieces of information associated with a specific process which is described below.



Pointer: Pointer points to another process control block. Pointer is used for maintaining the scheduling list.
Process State: Process state may be new, ready, running, waiting and so on.
Program Counter: Program Counter indicates the address of the next instruction to be executed for this process.
CPU registers: CPU registers include general purpose register, stack pointers, index registers and accumulators etc. number of register and type of register totally depends upon the computer architecture.
Memory management information: This information may include the value of base and limit registers, the page tables, or the segment tables depending on the memory system used by the operating system. This information is useful for de-allocating the memory when the process terminates.

Accounting information: This information includes the amount of CPU and real time used, time limits, job or process numbers, account numbers etc.

Process control block includes CPU scheduling, I/O resource management, file management information etc. The PCB serves as the repository for any information which can vary from process to process. Loader/linker sets flags and registers when a process is created. If that process get suspended, the contents of the registers are saved on a stack and the pointer to the particular stack frame is stored in the PCB. By this technique, the hardware state can be restored so that the process can be scheduled to run again.

Process scheduling

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 system. Such operating systems allow more than one process to be loaded into the executable memory at a time and 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.

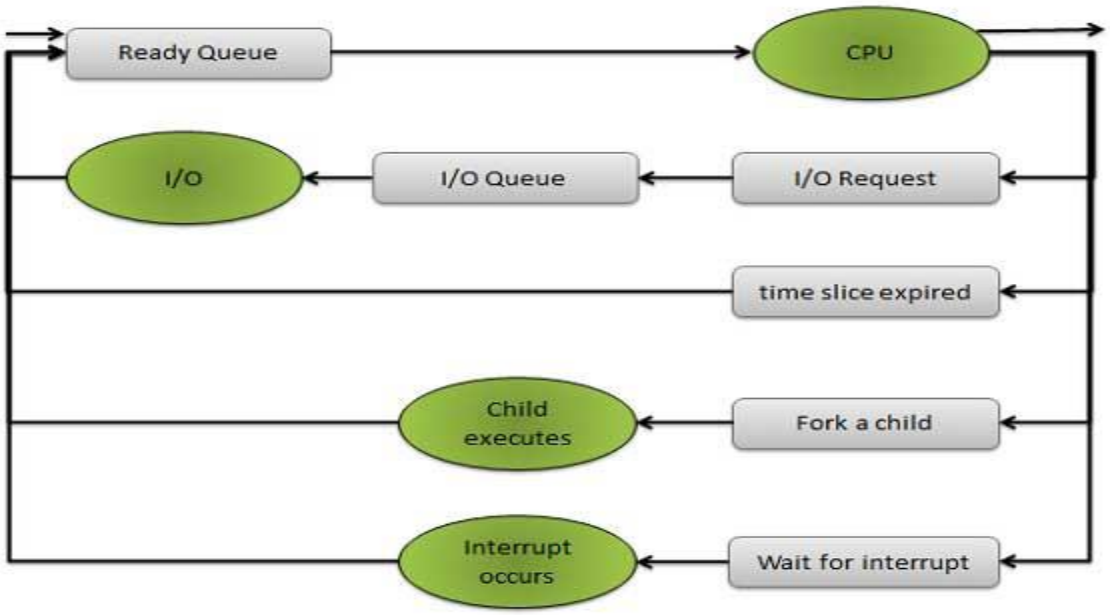
The Operating System maintains the following important process scheduling queues −

- **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 different queues. Scheduling queues refers to queues of processes or devices. When the process enters into the system, then this process is put into a job queue. This queue consists of all processes in the system. The operating system also maintains other queues such as device queue. Device queue is a queue for which multiple processes are waiting for a particular I/O device. Each device has its own device queue.

The below figure shows the queuing diagram of process scheduling. Queue is represented by rectangular box. The circles represent the resources that serve the queues. The arrows indicate the process flow in the system. A new process is initially put in the ready queue. It waits in the ready queue until it is selected for execution. Once the process is assigned to the CPU and is executing, one of the several events could occur:

- The process could issue an I/O request, and then be placed in an I/O queue.
- The process could create a new subprocess and wait for its termination.
- The process could be removed forcibly from CPU, as a result of an interrupt, and be put back in the ready queue.
  In the first two cases, the process eventually switches from the waiting state to the ready state and then put back to ready queue. The process continues this cycle until it terminates.



Schedulers

Schedulers are special system software which handles 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 job scheduler. Long term scheduler determines which programs are admitted to the system for processing. Job scheduler 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 process changes the state from new to ready, then there is use of long term scheduler.

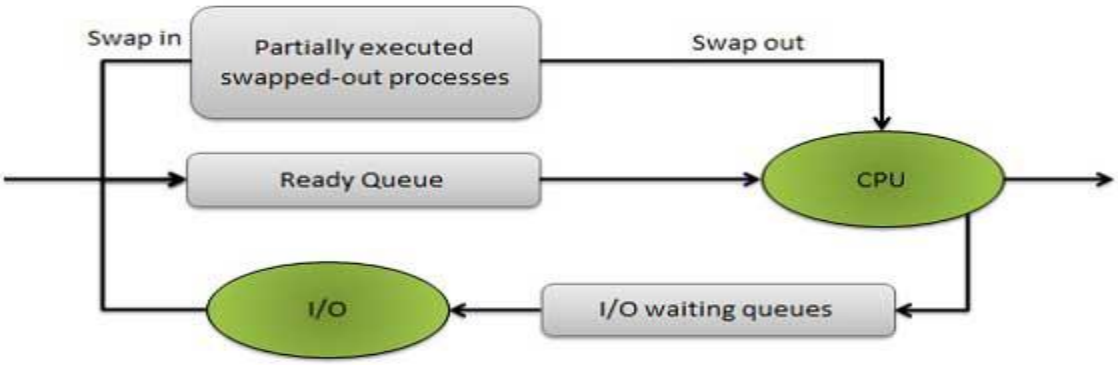
Short Term Scheduler

It is also called CPU scheduler. Main objective is increasing 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 process among the processes that are ready to execute and allocates CPU to one of them.

Short term scheduler also known as dispatcher, execute most frequently and makes the fine grained decision of which process to execute next. Short term scheduler is faster than long term scheduler.

Medium Term Scheduler

Medium term scheduling is part of the 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.



Running process may become suspended if it makes an I/O request. Suspended processes cannot make any progress towards completion. In this condition, to remove the process from memory and make space for other process, 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.

Comparison between Scheduler

S.N.	Long Term Scheduler	Short Term Scheduler	Medium Term Scheduler
1	It is a job scheduler	It is a CPU scheduler	It is a process swapping scheduler.
2	Speed is lesser than short term scheduler	Speed is fastest among other two	Speed is in between both short and long term scheduler.
3	It controls the degree of multiprogramming	It provides lesser control over degree of multiprogramming	It reduces the degree of multiprogramming.
4	It is almost absent or minimal in time sharing system	It is also minimal in time sharing system	It is a part of Time sharing systems.
5	It selects processes from pool and loads them into memory for execution	It selects those processes which are ready to execute	It can re-introduce the process into memory and execution can be continued.

Context Switch: A context switch is the mechanism to store and restore the state or context of a CPU in Process Control block so that a process execution can be resumed from the same point at a later time. Using this technique a context switcher enables multiple processes to share a single CPU. Context switching is an essential part of a multitasking operating system features.

Cooperating process: a process is co-operating, if it can affect or be affected by the other processes executing in the system. Any process that shares data with other processes is a cooperating process. There are several reasons for providing an environment that allows process cooperation:

- Information sharing: since several users may be interested in the same piece of information (ex: file), we must provide an environment to allow concurrent access to these types of resources.
- Computation speedup: if we want a particular task to run faster, we must break it into subtasks, each of which will be executing in parallel with the others. Such a speedup can be achieved only if the computer has multiple processing elements.
- Modularity: we may want to construct the system in a modular fashion, dividing the system functions into separate processes or threads.
- Convenience: even an individual user may have many tasks on which to work at one time.

Concurrent execution of cooperating processes requires mechanisms that allow processes to communicate with one another and to synchronize their actions. There are two fundamental models for Cooperating processes to communicate: Shared memory and Message passing.

In the shared memory model, a region of memory that is shared by cooperating processes is established. Processes can exchange information by reading and writing data to the shared region. Shared memory is faster than message passing.

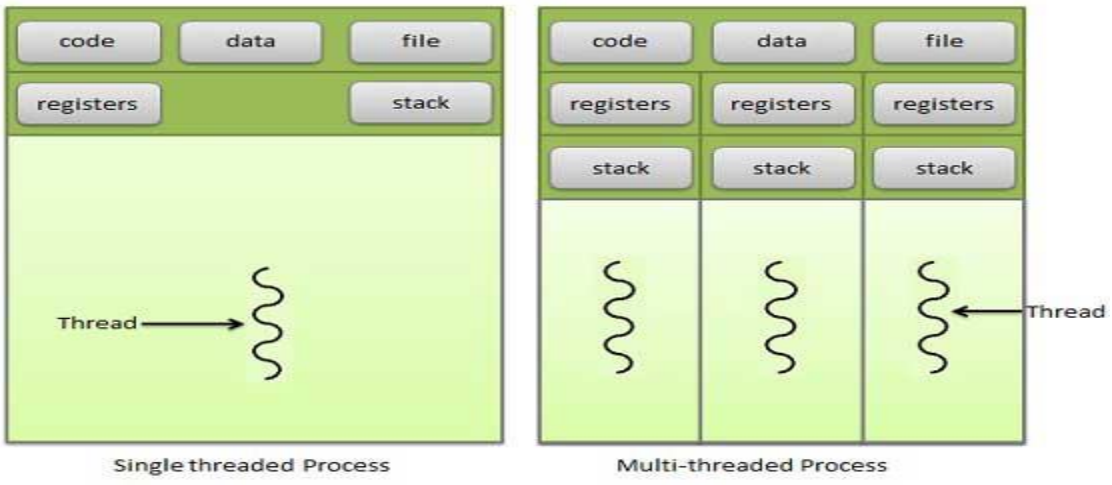
In the message passing model, communication takes place by means of messages exchanged between the cooperating processes. It is useful for exchanging smaller amount of data. It is also easier to implement than shared memory for interprocess communication.

Inter-process Communication: cooperating process requires an interprocess communication (IPC) mechanism that will allow them to exchange data and information without sharing the same address space. IPC is particularly useful in a distributed environment where the communicating processes may reside on different computers connected with a network. For example: chatting. IPC is best provided by a message passing system, and message systems can be defined in many ways.

Thread

A thread is a flow of execution through the process code, with its own program counter, system registers and stack. A thread is also called a light weight process. Threads provide a way to improve application performance through parallelism. Threads represent a software approach to improving performance of operating system by reducing the overhead thread is equivalent to a classical process.

Each thread belongs to exactly one process and no thread can exist outside a process. Each thread represents a separate flow of control. Threads have been successfully used in implementing network servers and web server. They also provide a suitable foundation for parallel execution of applications on shared memory multiprocessors. Following figure shows the working of the single and multithreaded processes.



Difference between Process and Thread

Process	Thread
Process is heavy weight or resource intensive.	Thread is light weight taking lesser resources than a process.
Process switching needs interaction with operating system.	Thread switching does not need to interact with operating system.
In multiple processing environments each process executes the same code but has its own memory and file resources.	All threads can share same set of open files, child processes.
If one process is blocked then no other process can execute until the first process is unblocked.	While one thread is blocked and waiting, second thread in the same task can run.
Multiple processes without using threads use more resources.	Multiple threaded processes use fewer resources.
In multiple processes each process operates independently of the others.	One thread can read, write or change another thread's data.

Advantages of Thread

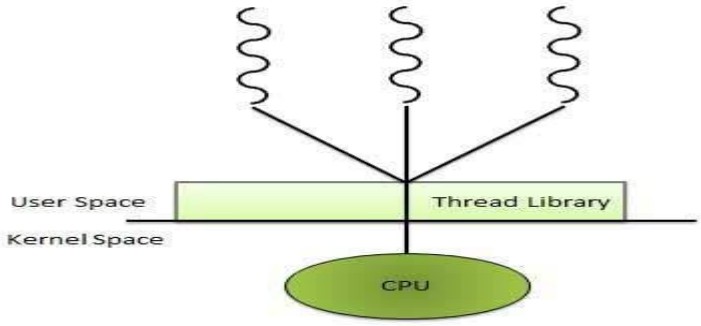
- Thread minimize context switching time.
- Use of threads provides concurrency within a process.
- Efficient communication.
- Economy- It is more economical to create and context switch threads.
- Utilization of multiprocessor architectures to a greater scale and efficiency.

Types of Thread

- **User Level Threads** -- User managed threads
- **Kernel Level Threads** -- Operating System managed threads acting on kernel, an operating system core.

User Level Threads

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



Advantages

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

Disadvantages

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

Kernel Level Threads

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

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

Advantages

- Kernel can simultaneously schedule multiple threads from the same process on multiple processes.

- If one thread in a process is blocked, the Kernel can schedule another thread of the same process.
- Kernel routines themselves can multithreaded.

Disadvantages

- Kernel threads are generally slower to create and manage than the user threads.
- Transfer of control from one thread to another within same process requires a mode switch to the Kernel.

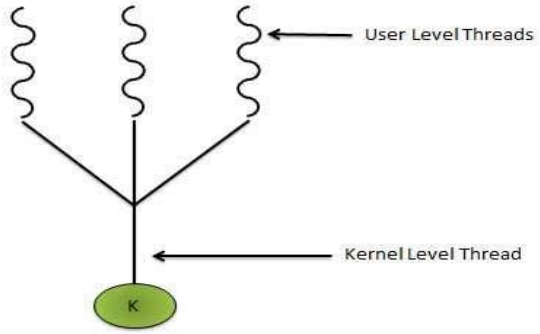
Multithreading Models

Some operating system provide a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach. In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process. Multithreading models are three types

- Many to many relationship.
- Many to one relationship.
- One to one relationship.

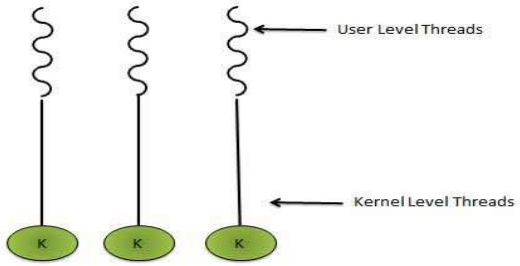
Many to One Model: Many to one model maps many user level threads to one Kernel level thread. Thread management is done in user space. When thread makes a blocking system call, the entire process will be blocked. Only one thread can access the Kernel at a time, so multiple threads are unable to run in parallel on multiprocessors.

If the user level thread libraries are implemented in the operating system in such a way that system does not support them then Kernel threads use the many to one relationship modes.



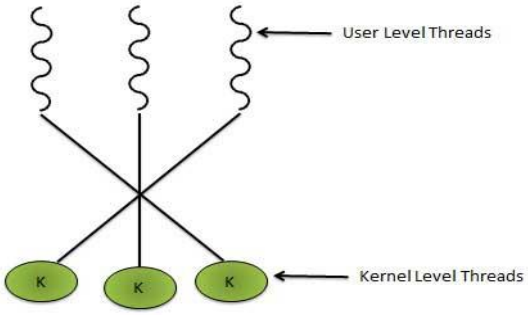
One to One Model: There is one to one relationship of user level thread to the kernel level thread.This model provides more concurrency than the many to one model. It also another thread to run when a thread makes a blocking system call. It support multiple thread to execute in parallel on microprocessors.

Disadvantage of this model is that creating user thread requires the corresponding Kernel thread. OS/2, windows NT and windows 2000 use one to one relationship model.



Many to Many Model: In this model, many user level threads multiplexes to the Kernel thread of smaller or equal numbers. The number of Kernel threads may be specific to either a particular application or a particular machine.

Following diagram shows the many to many model. In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallels on a multiprocessor.



Difference between User Level & Kernel Level Thread

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

Threading Issues

1. Thread cancellation: it means terminating a thread before it has finished working. There can be two approaches for this, one is asynchronous cancellation, which terminates the target thread immediately. The other is deferred cancellation allows the target thread to periodically check if it should be cancelled.
2. Signal Handling: signals are used in Unix system to notify a process that a particular event has occurred. Now in when a multithreaded process receives a signal, to which thread it must be delivered? It can be deliver to all, or a single thread.
3. fork( ) system call: it is a system call executed in the kernel through which a process creates a copy of itself. Now the problem in multithreading process is, if one thread forks will the entire process be copied or not?
4. Security issues: this arises because of extensive sharing of resources between multiple threads.

CPU scheduling

CPU scheduling is the basis of multiprogrammed operating system. In a single processor system, only one process can run at a time, any others must wait until the CPU is free. In multiprocessor system several processes are kept in memory at a time. When one process has to wait, the operating system takes the CPU away from that process and gives the CPU to another process. Scheduling of this kind is a fundamental operating function. Almost all computer resources are scheduled before use.

The success of CPU scheduling depends on an observed property of process: process execution consists of a cycle of CPU execution and I/O wait. The process alternates between these two states. Process execution begins with CPU burst followed by I/O burst.

CPU scheduler:

Whenever the CPU becomes idle, the operating system must select one of the processes in the ready queue to be executed. The selection process is carried out by the CPU scheduler or short term scheduler. The scheduler selects the processes in the memory that are ready to executer and allocate the CPU to that process.

Pre-emptive and non pre-emptive:

Non pre-emptive algorithm is designed so that once a process enters the running state, it is not removed from the processor until it has completed its service time.

Pre-emptive algorithm is driven by the notation of prioritized computation. The process with the highest priority should always be the one currently using the processor. If a process is currently using the processor and a new process with higher priority enters the ready list, the process on the processor should be removed and returned to the ready list until it is once again the highest priority process in the system.

Preemptive Scheduling	Non Preemptive Scheduling
The resources are allocated to a process for a limited time.	Once resources are allocated to a process, the process holds it till it completes its burst time or switches to waiting state.
Process can be interrupted in between.	Process can not be interrupted till it terminates or switches to waiting state.
If a high priority process frequently arrives in the ready queue, low priority process may starve.	If a process with long burst time is running CPU, then another process with less CPU burst time may starve.
Preemptive scheduling has overheads of scheduling the processes.	Non-preemptive scheduling does not have overheads.
Preemptive scheduling is flexible.	Non-preemptive scheduling is rigid.
Preemptive scheduling is cost associated.	Non-preemptive scheduling is not cost associative.

Scheduling criteria:

Different CPU scheduling algorithms have different properties, the choice of a particular algorithm may favor one class of processes over another. Many criteria have been suggested for comparing CPU scheduling algorithm.

1. **CPU utilization:** we want to keep the CPU as busy as possible. CPU utilization can range from 0 to 100 percent. In a real system, it should range from 40 percent to 90 percent.
2. **Throughput:** if the CPU is busy executing processes, then work is being done. One measure of work is the number of processes that are completed per time unit called throughput. For long process this rate may be one process per hour, for short process this may be 10 processes per second.
3. **Turnaround time:** the important criterion is how long it takes to execute that process. The interval from the time of submission of a process to the time of completion is the turnaround time. Turnaround time is the sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU, and doing I/O.
4. **Waiting time:** the CPU scheduling algorithm does not affect the amount of time during which a process executes or does I/O; it affects only the amount of time that a process spends waiting in the ready queue. Waiting time the sum of the periods spent waiting in the ready queue.
5. **Response time:** the measure of time from the submission of a request until the first response is produced is called response time.

It is desirable to maximize CPU utilization and throughput and to minimize turnaround time, waiting time and response time.

Scheduling Algorithms

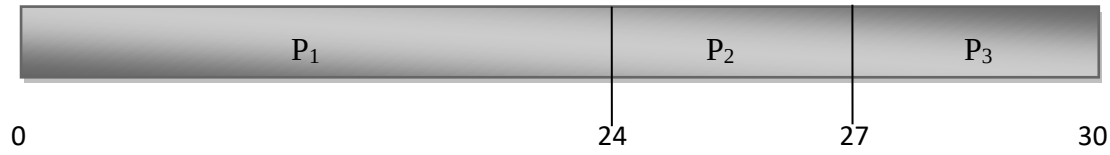
The four major scheduling algorithms are: First Come First Serve (FCFS) Scheduling, Shortest-Job-First (SJF) Scheduling, Priority Scheduling and Round Robin(RR) Scheduling

First Come First Serve (FCFS)

The simplest CPU scheduling algorithm is the first come first serve algorithm. In this scheme, the process that requests the CPU first is allocated the CPU first. The implementation of the FCFS is easily managed with a FIFO queue. This algorithm is simple to write and understand. The average waiting time under the FCFS policy, is quite long. Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds:

process	Burst time
P_1	24
P_2	3
P_3	3

If the processes arrive in the order P_1, P_2, P_3 are served in FCFS, we get following result shown in the Gantt chart.



Wait Time: Service Time - Arrival Time

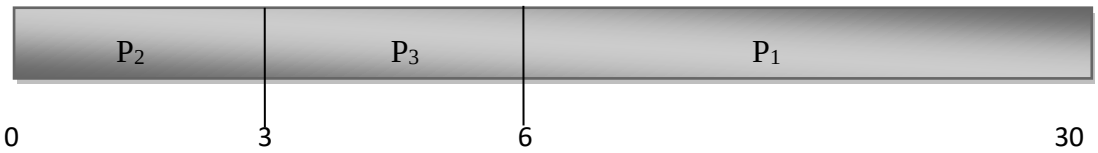
Process	Burst time	Arrival time	Service time
P_1	24	0	0
P_2	3	0	24
P_3	3	0	27

Wait time of each process is following

P_1 0 – 0 = 0 P_2 24 – 0 = 24 P_3 27 – 0 = 27

Average Wait Time: (0+24+27) / 3 = 17ms.

If the processes arrive in the order P_2, P_3, P_1 are served in FCFS, we get following result shown in the Gantt chart.



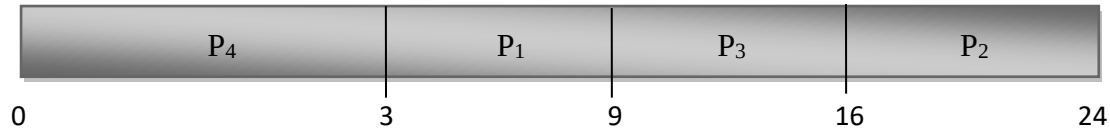
Now, the average Wait Time: (6+0+3) / 3 = 3ms. This reduction is substantial. The average waiting time vary if the process CPU burst times vary greatly. This algorithm is non preemptive. It is troublesome for time sharing systems, where user needs to get a share of the CPU at regular intervals.

Shortest Job First (SJF)

This is also known as **shortest job first**, or SJF. In this scheme, the CPU is assigned to the process that has the smallest CPU burst. If two processes have the same CPU burst, FCFS scheduling is used to break the tie. Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds.

process	Burst time
P_1	6
P_2	8
P_3	7
P_4	3

The Gantt chart is,



Process	Burst time	Arrival time	Service time
P_1	6	0	3
P_2	8	0	16
P_3	7	0	9
P_4	3	0	0

Wait time of each process is following

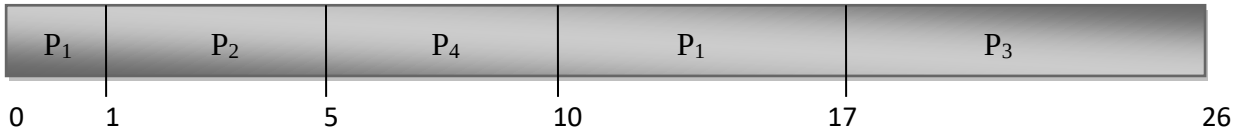
P_1 3 – 0 = 3 P_2 16 – 0 = 16 P_3 9 – 0 = 9 P_4 0 – 0 = 0

Average Wait Time: (3+16+9+0) / 4 = 7 ms.

The SJF can be either pre-emptive or non pre-emptive. In preemptive SJF algorithm, jobs are put in to ready queue as they arrive, but as a process with short burst time arrives, the existing process is preempted. Consider another example,

Process	Burst time	Arrival time
P_1	8	0
P_2	4	1
P_3	9	2
P_4	5	3

In pre-emptive case, the Gantt chart is,

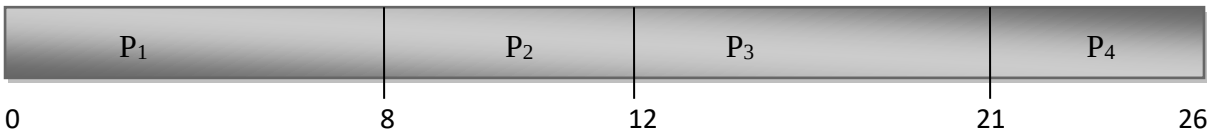


Wait time of each process is following

P_1 (0 – 0) + (10 – 1) = 9 P_2 1 – 1 = 0 P_3 17 – 2 = 15 P_4 5 – 3 = 2

Average Wait Time: (9+0+15+2) / 4 = 6.5 ms.

In non pre-emptive case, the Gantt chart is,



Average Wait Time: (0+8+12+21) / 4 = 7.75 ms.

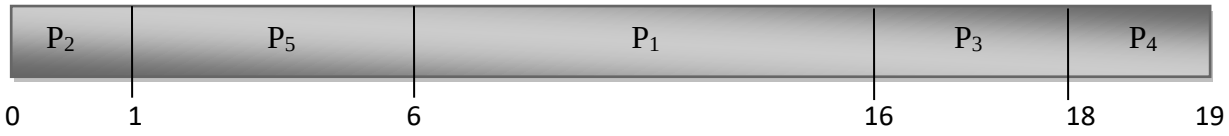
This algorithm is best to minimize waiting time. It is easy to implement in Batch systems where required CPU time is known in advance. But there is no way to know the CPU time required by the process, so it is impossible to implement in interactive systems.

Priority Based Scheduling

A priority is associated with each process, and the CPU is allocated to the process with the highest priority. Equal priority processes are scheduled in FCFS order. Priority can be decided based on memory requirements, time requirements or any other resource requirement. Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds.

process	Burst time	Priority
P_1	10	3
P_2	1	1
P_3	2	4
P_4	1	5
P_5	5	2

The Gantt chart,



Average Wait Time = 8.2 ms.

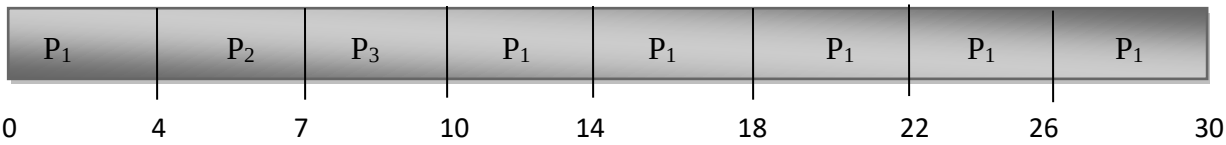
Priority scheduling may be either pre-emptive or non pre-emptive. Major problem with this algorithm is indefinite blocking or starvation. A process that is ready to run but waiting for the CPU can be considered as blocked. Priority scheduling can leave some low priority processes waiting indefinitely. The solution for this problem is aging. Aging is a technique of gradually increasing the priority of the process that wait in the system for a long time.

Round Robin Scheduling

It is designed for time sharing system. It is similar to FCFS, but pre-emption is added to switch between processes. A small unit of time, called a time quantum is defined. A time quantum is generally from 10 to 100 millisecond. The ready queue is treated as circular queue, once a process is executed for given time period that process is preempted and other process executes for given time period. Round Robin is the preemptive process scheduling algorithm. Consider the following set of processes that arrive at time 0, with the length of the CPU burst given in milliseconds and the time quantum is 4.

process	Burst time
P_1	24
P_2	3
P_3	3

Time Quantum = 4 ms. The Gantt chart,



Waiting time,

P_1 (0-0) + (10-4) + (14-14) + (18-18) + (22-22) + (26-26) = 6, P_2 4-0 = 4, P_3 7-0 = 7

Average Wait Time: (6+4+7) / 3 = 17/3 = 5.66 ms.

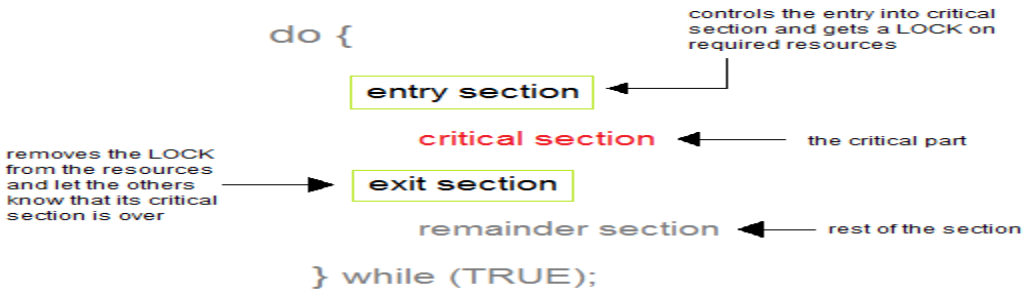
Unit-2

Process Synchronization

Process Synchronization means sharing system resources by processes in such a way that, Concurrent access to shared data is handled thereby minimizing the chance of inconsistent data. Maintaining data consistency demands mechanisms to ensure synchronized execution of cooperating processes. It was introduced to handle problems that arose while multiple process executions. Some of the problems are discussed below.

Critical Section Problem

A Critical Section is a code segment that accesses shared variables and has to be executed as an atomic action. It means that in a group of cooperating processes, at a given point of time, only one process must be executing its critical section. If any other process also wants to execute its critical section, it must wait until the first one finishes. Critical section problem is to design a protocol that the processes can use to co-operate, each process must request permission to enter its critical section. This request is implemented in "entry section", followed by critical section and exit section. The remaining code is the remainder section.



Solution to Critical Section Problem must satisfy the following three conditions :

1. **Mutual Exclusion:** Out of a group of cooperating processes, only one process can be in its critical section at a given point of time. If process P_1 is executing in its critical section, then no other processes can be executing in their critical section.
2. **Progress:** If no process is in its critical section, and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder section can participate in the decision on which will enter its critical section.
3. **Bounded Waiting:** There exists a bound on the number of times that other processes are allowed to enter their critical section after a process has made a request to enter its critical section and before that request is granted.

Bakery Algorithm

Bakery algorithm is a software approach to mutual exclusion, the reason for this name for the algorithm is in bakeries and other shops every customer receives a numbered ticket on arrival, allowing each to be served in turn. Unfortunately, the bakery algorithm cannot guarantee that two processes do not receive the same number. In the case of tie, the process with the lowest name is served first i.e., if P_i and P_j

receive the same number and if i<j, then P_i is served first. The structure of process P_i used in the bakery algorithm is

```
boolean choosing[n];
int num[n];
while(true)
{
choosing[i]=true;
num[i]=1+getmax(num[ ],n);
choosing[i]=false;
for(int j=0;j<n;j++)
{
while(choosing[j])
{}
while ( (num[j]!=0) && (num[j], j) < (num[i], i) )
{}
}
/*critical section*/
num[i]=0;
/*remainder section*/
}
```

The array choosing and num are initialized to false and zero respectively. We define the following notation,

(a,b) < (c,d) if a < c or if a = = c and b < d

If P_i is interested in entering critical section then it sets choosing[i] = true, and gets num[i] and resets choosing[i] = false. It has to check whether any other process is interested. If ((num[j]!=0) && (num[j], j) < (num[i], i)) is false then P_i enters critical section.

Semaphores

In 1965, Dijkstra proposed a new and very significant technique for managing concurrent processes by using the value of a simple integer variable to synchronize the progress of interacting processes. This integer variable is called **semaphore**. So it is basically a synchronizing tool shared between processes and is accessed only through two low standard atomic operations, wait and signal originally termed as P() and V() respectively.

The classical definition of wait and signal are:

- **Wait:** decrement the value of its argument S as soon as it would become non-negative.

```
wait(S)
```

```
{
while(s≤0)
;    //no operation
S - - ;
}
```

- **Signal:** increment the value of its argument, S as an individual operation.

```
Signal(S)
{
S + + ;
}
```

When one process modifies the semaphore value, no other process can simultaneously modify that same semaphore value.
For example, consider two concurrently running processes, P_1 with a statement S_1 and P_2 with S_2. Suppose that we require S_2 be executed only after S_1 has completed. Then P_1 and P_2 share a common semaphores "synch", initialized to 0 and inserting the statements.

S_1;	wait(synch);
signal(synch);	S_2;
(in process P_1)	**(in process P_2)**

P_2 will execute S_2 only after P_1 has invoked signal(synch) which is after S_1.

The implementation of semaphores with a waiting queue may result in a situation where two or more process are waiting indefinitely for an event that can be caused only by one of the waiting processes. The event is the execution of a signal operation. When such state is reached, these processes are said to be deadlocked. Let P_1 and P_2 accessing two semaphores S and Q,

P_1	P_2
wait(S);	wait(Q);
wait(Q);	wait(S);
.	.
.	.
.	.
signal(S);	signal(Q);
signal(Q);	signal(S);

P_1 executes wait(S) and P_1 executes wait(Q). When P_1 executes wait(Q), it must wait until P_2 executes signal(Q). Similarly when P_2 executes wait(S), it must wait until P_1 executes signal(S). since these signal operations cannot be executed P_1 and P_2 are deadlocked.

Types of Semaphores

Semaphores are mainly of two types: Binary Semaphore and Counting Semaphores.

- **Binary Semaphore:** It is a special form of semaphore used for implementing mutual exclusion; hence it is often called Mutex. A binary semaphore is initialized to 1 and only takes the value 0 and

1 during execution of a program. Binary semaphore is used to deal with the critical section problem for multiple processes. The n processes share a semaphore, mutex.

- **Counting Semaphores:** These are unrestricted domain used to control access to given resource consisting of a finite number of instances. Semaphore is initialized to the number of resources available. Each process that wishes to use a resource performs a wait( ) operation on the semaphore (decrement the count). When a process releases a resource, it performs a signal operation (increment count). When the semaphore goes to 0, all resources are being used. The process that wish to use will be blocked until count becomes greater than 0.

Classical Problems of Synchronization

The different synchronization problems are,

- **The Bounded Buffer Problem**

  Assume that the pool consists of n buffers, each capable of holding one item. The mutex semaphore provides mutual exclusion for access to the buffer pool and is initialized to the value 1. The empty and full semaphores count the number of empty and full buffers, respectively. The semaphore empty is initialized to the value n; the semaphore full is initialized to the value 0.

  The code for producer process is,

```
do{
……
Produce an item
……
wait (empty);
wait (mutex);
……
Add to buffer
……
signal (mutex);
signal (full);
}while(1);
```

  The code for consumer process is,

```
do{
wait (full);
wait (mutex);
……
Remove an item
……
signal (mutex);
signal (empty);
```

…….
Consume an item
…….
}while(1);

- **The Reader-Writer Problem**

A data object such as file or record is to be shared among several concurrent processes. Some of these processes may want only to read the content of the shared object, whereas others may want to update the shared object. We distinguish between these two types of processes by referring to those processes that are interested in only reading as readers, and to the rest as writers. Obviously, if two readers access the shared data object simultaneously, no adverse effects will result. However if a writer and some other process either a reader or a writer access the shared object simultaneously, then problem arises. To ensure that these difficulties do not arise, we require that the writers have exclusive access to the shared object. This synchronization problem is called as readers-writers problem.

The code for writer process is,

wait (wrt);
……
writing
……
signal (wrt);

The code for reader process is,

wait (mutex);
readcount++;
if(readcount = = 1)
wait (wrt);
signal (mutex);
……
Reading
……
wait (mutex);
readcount--;
if(readcount = = 0)
signal (wrt);
signal (mutex);

The semaphores mutex and wrt are initialized to 1; readcount is initialized to 0. The semaphore wrt is used to ensure mutual exclusion semaphore for writers. It is also used by the first or last reader that enters or exits the critical section. The mutex semaphore is used to ensure mutual exclusion. The readcount keeps track of how many processes are currently reading the object.

- **The Dinning-Philosopher Problem**

Consider five philosophers who spend their lives thinking and eating. The philosophers share a common circular table surrounded by five chairs, each belonging to one philosopher. In the center of the table is a bowl of rice, and the table is laid with five plates and chopsticks. When a philosopher thinks he does not interact with his colleagues. From time to time a philosopher gets hungry and tries to pick up the two chopsticks that are close to him. The philosopher may pick up only one chopstick at a time and when he has both chopsticks at the same time, he eats. When he finishes eating, he puts down both of the chopsticks and starts thinking.

The solution is to represent each chopstick by a semaphore. A philosopher tries to grab the chopstick by executing a wait operation on that semaphore; he releases his chopstick by executing the signal operation on the semaphore. All elements of the semaphore chopstick are initialized to 1. The structure of philosopher i is,

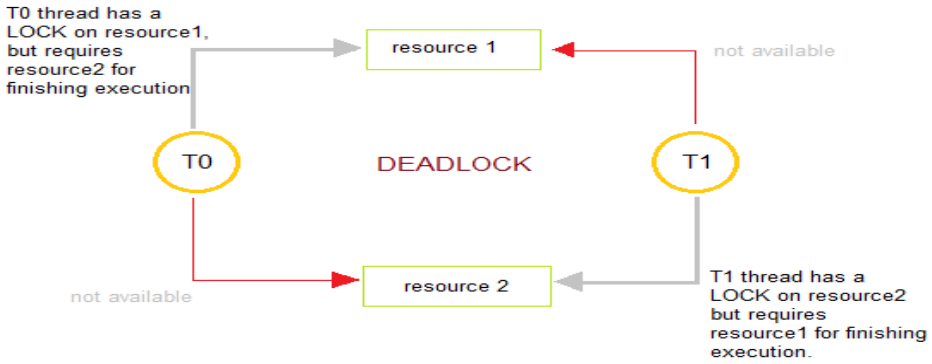
```
do{
wait (chopstick[i]);
wait (chopstick[(i+1) % 5]);
……
eat
……
signal (chopstick[i]);
signal (chopstick[(i+1) % 5]);
…..
Think
……
}while(1);
```

Although this solution guarantees that no two neighbors are eating simultaneously. Suppose that all five philosophers become hungry simultaneously and each grabs his left chopstick. All the elements of chopstick will now be equal to 0. When each philosopher tries to grab his right chopstick, he will be delayed, deadlock arises. The remedies for deadlock problem are,

1. Allow at most four philosophers to be sitting simultaneously at the table.
2. Allow a philosopher to pick up his chopsticks only if both the chopsticks are available.
3. Use an asymmetric solution, that is an odd philosopher picks up first his left chopstick and then right one, whereas an even philosopher picks up first his right chopstick and then left one.

Deadlock

In a multiprogramming environment several processes compete for a finite number of resources. A process requests resources; if the resources are not available at that time, the process enters a wait state. Waiting process may never again change its state, because the resources they have requested are held by other waiting processes. This situation is called a deadlock.



Deadlock Characterization:

1. **Necessary Condition:** A deadlock situation can arise when four conditions hold simultaneously in the system.

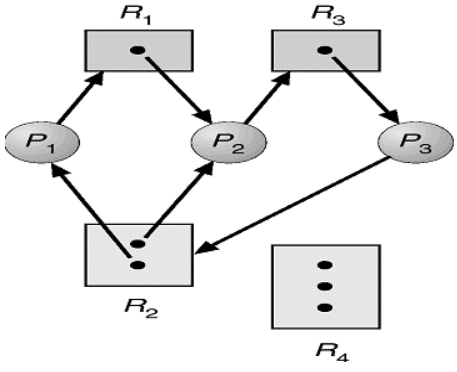
   a. Mutual Exclusion: At least one resource must be held in a non shareable mode; that is, only one process at a time can use the resource. If another process requests that resource the requesting process must be delayed until the resource has been released.
   b. Hold and wait: A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.
   c. No preemption: Resources cannot be preempted; that is , a resource can be released only voluntarily by the process holding it, after that process has completed its task.
   d. Circular wait: A set $\{P_0, P_1, P_2, \ldots\ldots, P_n\}$of waiting processes must exist such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by $P_2, \ldots$, P_{n-1} is waiting for a resource that is held by P_n, and P_n is waiting for a resource that is held by P_0.

2. **Resource-Allocation graph:** deadlock can be described more precisely in terms of directed graph called a **system resource allocation graph.** The graph consists of a set of vertices V and a set of edges E. the set of vertices are partitioned into two different types of nodes $P = \{P_1, P_2, \ldots\ldots, P_n\}$, the set consisting of all active processes in the system and $R = \{R_1, R_2, \ldots\ldots, R_n\}$, the set consisting of all resource types in the system.

   A directed edge from process P_i to resource type R_j is denoted by $P_i \rightarrow R_j$ called as request edge, it signifies that process P_i requested an instance of resource type R_j. A directed edge from resource type R_j to process P_i is denoted by $R_j \rightarrow P_i$ called as assignment edge it signifies that resource type R_j has been allotted to process P_i.

   The sets P, R, and E:

   - $P = \{P_1, P_2, P_3\}$
   - $R = \{ R_1, R_2, R_3, R_4\}$
   - $E = \{ P_1 \rightarrow R_1, P_2 \rightarrow R_3, P_3 \rightarrow R_2, R_1 \rightarrow P_2, R_2 \rightarrow P, R_2 \rightarrow P_2, R_3 \rightarrow P_3 \}$



If the graph contains no cycles, then no process in the system is deadlocked. If the graph does contain a cycle, then a deadlock may exist. If each resource type has exactly one instance, then a cycle implies that a deadlock has occurred. If each resource type has several instances, then a cycle does not necessarily imply that a deadlock has occurred.

Methods for Handling Deadlock:

We can deal with deadlock problem in one of the three ways:

- We can use a protocol to prevent or avoid deadlocks, ensuring that system will never enter a deadlock state.
- We can allow the system to enter a deadlock state, detect it, and recover.
- We can ignore the problem altogether, and pretend that deadlocks never occur in the system. This solution is used by most operating systems.

To ensure that deadlocks never occur, the system can use either deadlock prevention or deadlock avoidance scheme. If a system does not employ either deadlock prevention or a deadlock avoidance algorithm, then a deadlock situation may occur. In this environment, the system can provide an algorithm that examines the state of the system to determine whether a deadlock has occurred, and an algorithm to recover from the deadlock.

Deadlock Prevention:

For a deadlock to occur, each of the four necessary conditions must hold. By ensuring that at least one of these cannot hold, we can prevent the occurrence of a deadlock.

1. Mutual Exclusion: It must hold for non sharable resources, for example printer cannot shared simultaneously by several processes. Sharable resources doesn't involve in deadlock, for example read only files, no process needs to wait. In general we cannot prevent deadlock by denying the mutual exclusion condition, because some resources are non shareable.
2. Hold and Wait: To ensure this condition, we must guarantee that whenever a process request a resource, it does not hold any other resources.
   One protocol is to allow each process to request and be allocated all its resources before it begins execution. Alternative protocol is to allow a process to request resource only when it has none. Before it can request for additional resource, it must release all the resources that it is currently allocated.
3. No Preemption: If a process is holding some resource, and request another resource that cannot be immediately allocated to it then all resources currently being held are preempted. The preempted resources are added to the list of resources for which process is waiting. The process will be restarted only when it regain its old resources as well as the new one.
   Alternatively, if a process requests some resource, we first check whether they are allocated to some other process that is waiting for additional resources. If so preempt the resources from waiting and allocate it to requested process. Both conditions failed process has to wait.
4. Circular Wait: one way to ensure that circular wait never holds is to impose a total ordering of all resource types and to require that each process request resource in an increasing order. Let R= {R_1,

R_2, R_3, …., R_m} set of resource types. We assign a unique integer number to each resource type. We define one function

F: R → N, where N is set of natural numbers. For example, F (tape drive) = 1, F (Printer) =5.

The protocol to prevent deadlock is, initially a process can request any number of instances of a resource type say, R_i. after that the process can request instances of resource type R_j if and only if F (R_j) ≥ F (R_i).

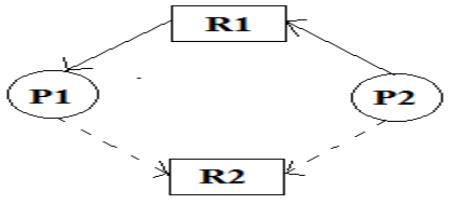
Deadlock Avoidance:

By ensuring that at least one of the four conditions cannot hold, we can prevent the occurrence of a deadlock, but the side effect is low device utilization and reduced system throughput. An alternative method for avoiding deadlock is to require additional information about how resources are to be requested. The complete sequence of request and releases for each process is needed, then the system can decide for each request whether to wait or not in order to avoid deadlock. In making this decision, the system consider the resources currently available, resources currently allocated to each process, and the future request and releases of each process.

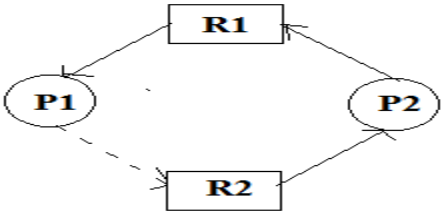
A state is safe if the system can allocate resources to each process (up to maximum) in some order and still avoid a deadlock. If a system is already in a safe state, we can try to stay away from an unsafe state and avoid deadlock. Deadlocks cannot be avoided in an unsafe state. A safe sequence of processes and allocation of resources ensures a safe state. Deadlock avoidance algorithms try not to allocate resources to a process if it will make the system in an unsafe state. Since resource allocation is not done right away in some cases, deadlock avoidance algorithms also suffer from low resource utilization problem.

A resource allocation graph is generally used to avoid deadlocks. If there are no cycles in the resource allocation graph, then there are no deadlocks. If there are cycles, there may be a deadlock. If there is only one instance of every resource, then a cycle implies a deadlock. Vertices of the resource allocation graph are resources and processes. The resource allocation graph has request edges and assignment edges. An edge from a process to resource is a request edge and an edge from a resource to process is an allocation edge. A calm edge denotes that a request may be made in future and is represented as a dashed line. Based on calm edges we can see if there is a chance for a cycle and then grant requests if the system will again be in a safe state.

Consider the image with calm edges as below:



If R_2 is allocated to P_2 and if P_1 request for R_2, there will be a deadlock.



The resource allocation graph is not much useful if there are multiple instances for a resource. In such a case, we can use **Banker's algorithm**. In this algorithm, every process must tell upfront the maximum resource of each type it need, subject to the maximum available instances for each type. Allocation of resources is made only, if the allocation ensures a safe state; else the processes need to wait. The Banker's algorithm can be divided into two parts: Safety algorithm if a system is in a safe state or not. The resource request algorithm make an assumption of allocation and see if the system will be in a safe state. If the new state is unsafe, the resources are not allocated and the data structures are restored to their previous state; in this case the processes must wait for the resource. Several data structure are needed for this, let n be the number of process and m be the number of resource types.

- Available: indicates the number of available resources of each type. If Available[j] = k, there are k instances of resource type R_j available.
- Max: an n x m matrix defines the maximum demand of each process. If Max[i][j] = k, then process P_i may request at most k instances of resource type R_j.
- Allocation: an n x m matrix defines the number of resources of each type currently allocated to each process. If Allocation[i][j] = k, then process P_i currently allocated k instances of resource type R_j.
- Need: an n x m matrix indicates the remaining resource need of each process. If Need[i][j] = k, then process P_i may need k more instances of resource type R_j to complete its task.note that Need[i][j] = Max[i][j] – Allocation[i][j]

Safety algorithm: We present algorithm for finding out whether or not a system is in a safe state. Described as

1. Work and Finish be vectors of length m and n respectively. Initially Work = Available Finish[i] = false.
2. Find an i such that both Finish[i] = = false, Need[i] ≤ Work

   If no such i exists, go to step 4.

3. Work = Work + Allocation[i]

   Finish[i] = true

   Go to step 2

4. If Finish[i] = = true for all i, then the system is in safe.

Resource request algorithm: Determines if request can be safely granted. Let Request[i] be request vector of P_i, if Request[i] = k then the process P_i wants k instances of resource type R_j.

1. If Request[i] ≤ Need[i], go to step 2 otherwise raise an error condition, since the process has exceeded its Max.
2. If Request[i] ≤ Available[i], go to step 3 otherwise P_i must wait, since the resources are not available..
3. The system pretend that requested resource is allocated to P_i by modifying

   Available = Available – Request[i]

   Allocation[i] = Allocation[i] + Request[i]

   Need[i] = Need[i] – Request[i]

If the resulting resource- allocation state is safe, the transaction is completed, and process P_i is allocated its resources. If it is unsafe, then P_i must wait for Request[i], and the old resource-allocation state is restored.

Example 1: Consider the table given below for a system, find the need matrix and the safety sequence, is the request from process P_1 (0, 1, 2) can be granted immediately.
Resource – 3 types, A – (10 instances) B – (5 instances) C – (7 instances)

Process	Allocation			Maximum			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	5	3	7	4	3	3	3	2
P_1	2	0	0	3	2	2	1	2	2			
P_2	3	0	2	9	0	2	6	0	0			
P_3	2	1	1	2	2	2	0	1	1			
P_4	0	0	2	4	3	3	4	3	1			

Solution: Banker's Algorithm

Step 1:

Safety for process P_0

$need_0$ = (7, 4, 3)

If $need_0 \leq$ Available

if [(7, 4, 3) ≤ (3, 3, 2)] (false)

Process P_0 must wait.

Step 2:

Safety for process P_1

$need_1$ = (1, 2, 2)

if $need_i \leq$ Available

if $[(1, 2, 2) \leq (3, 3, 2)]$

P_i will execute.

Available = Available + Allocation

= (3, 3, 2) + (2, 0, 0)

= (5, 3, 2)

Step 3:

Safety for process P_2

$need_2 = (6, 0, 0)$

if $need_2 \leq$ Available

if $[(6, 0, 0) \leq (5, 3, 2)]$ (false)

P_3 will execute.

Available = Available + Allocation

= (5, 3, 2) + (2, 1, 1)

= (7, 4, 3)

Step 5:

Safety for process P_4

$need_4 = (4, 3, 1)$

If $need_4 \leq$ Available

If $[(4, 3, 1) \leq (6, 4, 3)]$

P_4 will execute.

Available = Available + Allocation

= (7, 4, 3) + (0, 0, 2)

= (7, 4, 5)

Step 6:

Safety for process P_0

$need_0 = (7, 4, 3)$

if $need_0 \leq$ Available

if $[(7, 4, 3) \leq (7, 4, 5)]$

P_0 will execute.

Available = Available + Allocation

= (7, 4, 5) + (0, 1, 0)

= (7, 5, 5)

Step 7:

Safety for process P_2

$need_2 = (6, 0, 0)$

if $need_2 \leq$ Available

if $[(6, 0, 0) \leq (7, 5, 5)]$

P_2 will execute.

Available = Available + Allocation

= (7, 5, 5) + (3, 0, 2)

= (10, 5, 7)

Safety Sequence = < P_1, P_3, P_4, P_0, P_2>. The system is in a safe state.

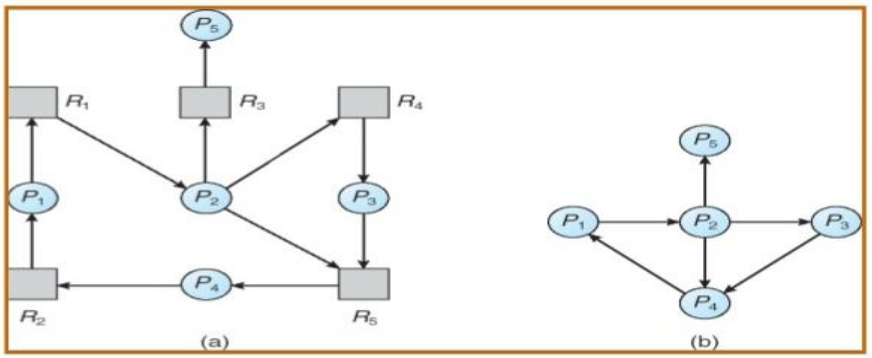
If P_1 requests (1,0,2), we decide whether this request can be immediately granted or not, by first checking that $Request_i \leq$ Available i.e., $(1,0,2) \leq (3,3,2)$ if it is true we then pretend that this request has been fulfilled, and we arrive at the following new state:

Process	Allocation			Need			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	7	4	3	2	3	0
P_1	3	0	2	0	2	0			
P_2	3	0	2	6	0	0			
P_3	2	1	1	0	1	1			
P_4	0	0	2	4	3	1			

We must determine whether this new system state is safe. To do so, we execute the safety algorithm and find that the sequence < P_1, P_3, P_4, P_0, P_2> satisfies our requirement. Hence we can immediately grant the request of process P_1.

Deadlock Detection: If deadlock prevention and avoidance are not done properly, as deadlock may occur and only things left to do is to detect the recover from the deadlock.

If all resource types has only single instance, then we can use a graph called wait-for-graph, which is a variant of resource allocation graph. Here, vertices represent processes and a directed edge from P_1 to P_2 indicates that P_1 is waiting for a resource held by P_2. Like in the case of resource allocation graph, a cycle in a wait-for-graph indicate a deadlock. So the system can maintain a wait-for-graph and check for cycles periodically to detect any deadlocks.



Resource-Allocation Graph Corresponding wait-for graph

The wait-for-graph is not much useful if there are multiple instances for a resource, as a cycle may not imply a deadlock. In such a case, we can use an algorithm similar to Banker's algorithm to detect deadlock. We can see if further allocations can be made on not based on current allocations. The data structures we use are,

- Available: indicates the number of available resources of each type. If Available[j] = k, there are k instances of resource type R_j available.
- Allocation: an n x m matrix defines the number of resources of each type currently allocated to each process. If Allocation[i][j] = k, then process P_i currently allocated k instances of resource type R_j.
- Request[i][j]: indicates the current request of each process. If Request[i][j] = k, then process P_i is requesting k more instances of resource type R_j.

Algorithm:

1. Initially Work = Available

   Finish[i] = false. If Allocation $\neq 0$

   Finish[i] = true. If Allocation = 0

2. Find an i such that both Finish[i] = = false, Request[i] $\leq$ Work

   If no such i exists, go to step 4.

3. Work = Work + Allocation[i]

   Finish[i] = true

   Go to step 2

4. If Finish[i] = = false, then the system is in deadlock.

Example 2: Resource – 3 types, A – (7 instances) B – (2 instances) C – (6 instances). Suppose at time T_0 we have the following resource allocation state:

Process	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	0	0	0	0	0	0
P_1	2	0	0	2	0	2			
P_2	3	0	3	0	0	0			
P_3	2	1	1	1	0	0			
P_4	0	0	2	0	0	2			

If we execute the algorithm we can said that the system is not in a deadlock state by finding the sequence < P_0, P_2, P_3, P_1, P_4> will result in Finish[i] = true for all i.

Suppose now that process P_2 makes one additional request (0 0 1), then

Process	Allocation			Request			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	1	0	0	0	0	0	0	0
P_1	2	0	0	2	0	2			
P_2	3	0	3	0	0	1			
P_3	2	1	1	1	0	0			
P_4	0	0	2	0	0	2			

After the execution of the process P_0 it releases all the resources which are held by it. The number of available resources is not sufficient to fulfill the requests of the other processes. Thus a deadlock exists, consisting of processes P_2, P_3, P_1 and P_4.

Recovery from deadlock

Once a deadlock is detected, you will have to break the deadlock. It can be done through different ways, including, aborting one or more processes to break the circular wait condition causing the deadlock and preempting resources from one or more processes which are deadlocked.

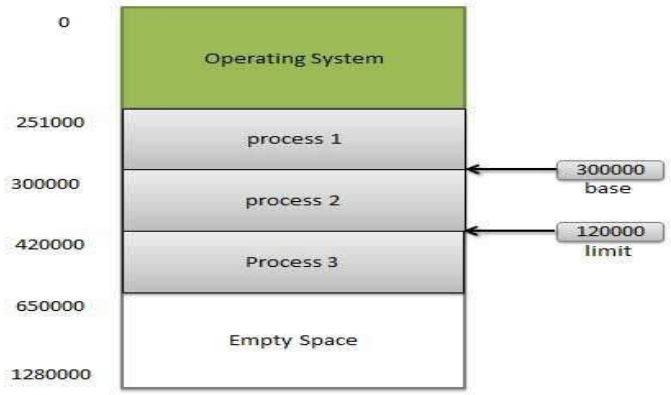
Process termination: to eliminate deadlock by aborting a process, we use one of two methods. In both, the system reclaim all resources allocated to the terminating process.

1. Abort all deadlocked processes – completely clears the deadlock problem but at great expense.
2. Abort one process at a time until the deadlock cycle is eliminated – after each process is aborted deadlock detection algorithm must be invoked to detect any process in deadlock state.

Resource pre-emption: pre-empt some resources from processes and give these resource to other processes until the deadlock cycle is broken. Here three issues must be addressed, selecting a victim (which process to select for preemption), roll back (if we preempt a resource from a process, roll back that process to some safe state and restart from there) and starvation (how can we guarantee that resources will not always preempted from same process).

Unit-3

Memory management



Memory management is the functionality of an operating system which handles or manages primary memory. Memory management keeps track of each and every memory location either it is allocated to some process or it is free. It checks how much memory is to be allocated to processes. It decides which process will get memory at what time. It tracks whenever some memory gets freed or unallocated and correspondingly it updates the status.

Memory management provides protection by using two registers, a base register and a limit register. The base register holds the smallest legal physical memory address and the limit register specifies the size of the range. For example, if the base register holds 300000 and the limit register is 120000, then the program can legally access all addresses from 300000 through 411999.

Instructions and data to memory addresses can be done in following ways

- **Compile time** -- When it is known at compile time where the process will reside, compile time binding is used to generate the absolute code.
- **Load time** -- When it is not known at compile time where the process will reside in memory, then the compiler generates re-locatable code.
- **Execution time** -- If the process can be moved during its execution from one memory segment to another, then binding must be delayed to be done at run time

Dynamic Loading

In dynamic loading, a routine of a program is not loaded until it is called by the program. All routines are kept on disk in a re-locatable load format. The main program is loaded into memory and is executed. Other routines methods or modules are loaded on request. Dynamic loading makes better memory space utilization and unused routines are never loaded.

Dynamic Linking

Linking is the process of collecting and combining various modules of code and data into a executable file that can be loaded into memory and executed. Operating system can link system level libraries to a program. When it combines the libraries at load time, the linking is called static linking and when this linking is done at the time of execution, it is called as dynamic linking.

In static linking, libraries linked at compile time, so program code size becomes bigger whereas in dynamic linking libraries linked at execution time so program code size remains smaller.

Definition of Logical Address

Address generated by CPU while a program is running is referred as Logical Address. The logical address is virtual as it does not exist physically. Hence, it is also called as Virtual Address. This address is used as a reference to access the physical memory location. The set of all logical addresses generated by a programs perspective is called Logical Address Space.

The logical address is mapped to its corresponding physical address by a hardware device called Memory-Management Unit. The address-binding methods used by MMU generates identical logical and physical address during compile time and load time. However, while run-time the address-binding methods generate different logical and physical address.

Definition of Physical Address

Physical Address identifies a physical location in a memory. MMU (Memory-Management Unit) computes the physical address for the corresponding logical address. MMU also uses logical address computing physical address. The user never deals with the physical address. Instead, the physical address is accessed by its corresponding logical address by the user. The user program generates the logical address and thinks that the program is running in this logical address. But the program needs physical memory for its execution. Hence, the logical address must be mapped to the physical address before they are used.

The logical address is mapped to the physical address using a hardware called Memory-Management Unit. The set of all physical addresses corresponding to the logical addresses in a Logical address space is called Physical Address Space.

The run-time 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.

Logical Address	Physical Address
It is the virtual address generated by CPU	The physical address is a location in a memory unit.
Set of all logical addresses generated by CPU in reference to a program is referred as Logical Address Space.	Set of all physical addresses mapped to the corresponding logical addresses is referred as Physical Address.
The user can view the logical address of a program.	The user can never view physical address of program
The user uses the logical address to access the physical address.	The user can not directly access physical address.
The Logical Address is generated by the CPU	Physical Address is Computed by MMU

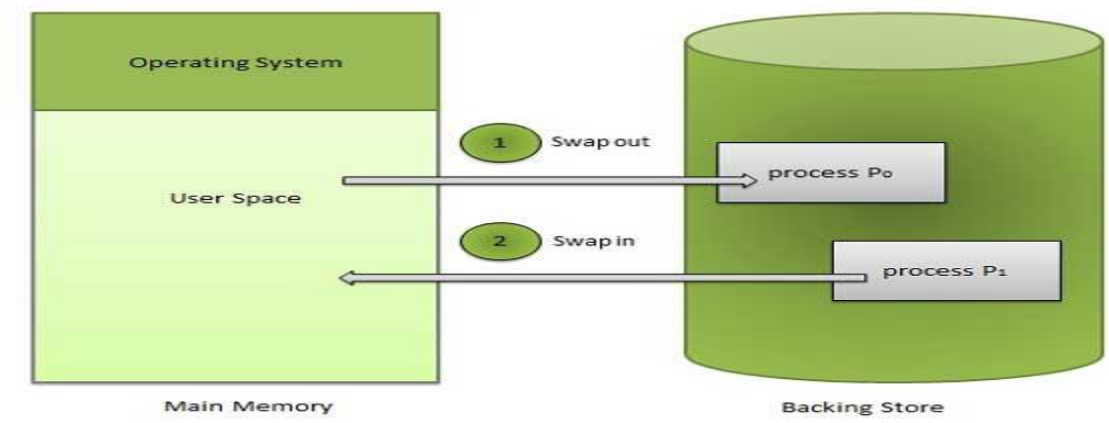
Swapping

Swapping is a mechanism in which a process can be swapped temporarily out of main memory to a backing store and then brought back into memory for continued execution.

Backing store is a usually a hard disk drive or any other secondary storage which fast in access and large enough to accommodate copies of all memory images for all users. It must be capable of providing direct access to these memory images.

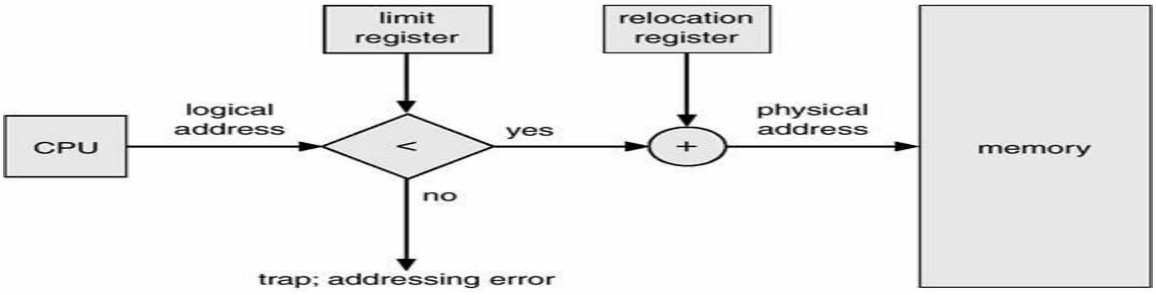
Major time consuming part of swapping is transfer time. Total transfer time is directly proportional to the amount of memory swapped. Let us assume that the user process is of size 100KB and the backing store is a standard hard disk with transfer rate of 1 MB per second. The actual transfer of the 100K process to or from memory will take

100KB / 1000KB per second
= 1/10 second
= 100 milliseconds



Contiguous memory allocation:

Contiguous memory allocation is a classical memory allocation model that assigns a process consecutive memory blocks (that is, memory blocks having consecutive addresses). Contiguous memory allocation is one of the oldest memory allocation schemes. When a process needs to execute, memory is requested by the process. The size of the process is compared with the amount of contiguous main memory available to execute the process. If sufficient contiguous memory is found, the process is allocated memory to start its execution. Otherwise, it is added to a queue of waiting processes until sufficient free contiguous memory is available.



Main memory usually has two partitions

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

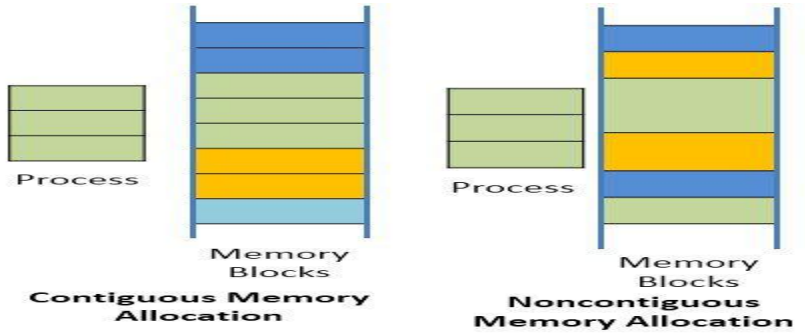
Operating system uses the following memory allocation mechanism.

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.
2	**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.

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.

Contiguous Memory Allocation	Noncontiguous Memory Allocation
Allocates consecutive blocks of memory to a process.	Allocates separate blocks of memory to a process.
Contiguous memory allocation does not have the overhead of address translation while execution of a process.	Noncontiguous memory allocation has overhead of address translation while execution of a process.
A process executes faster in contiguous memory allocation	A process executes quite slower comparatively in noncontiguous memory allocation.
The solution is memory space must be divided into the fixed-sized partition and each partition is allocated to a single process only.	The solution is divide the process into several blocks and place them in different parts of the memory according to the availability of memory space available.
A table is maintained by operating system which maintains the list of available and occupied partition in the memory space	A table has to be maintained for each process that carries the base addresses of each block which has been acquired by a process in memory.



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

- **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.
- **Internal fragmentation:** Memory block assigned to process is bigger. Some portion of memory is left unused as it cannot be used by another process.

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.

Internal Fragmentation	External Fragmentation
It occurs when fixed sized memory blocks are allocated to the processes.	It occurs when variable size memory spaces are allocated to the processes dynamically.
When the memory assigned to the process is slightly larger than the memory requested by the process this creates free space in the allocated block causing internal fragmentation.	When the process is removed from the memory, it creates the free space in the memory causing external fragmentation.
The solution is memory must be partitioned into variable sized blocks and assign the best fit block to the process.	The solution is compaction, paging and segmentation.

Paging

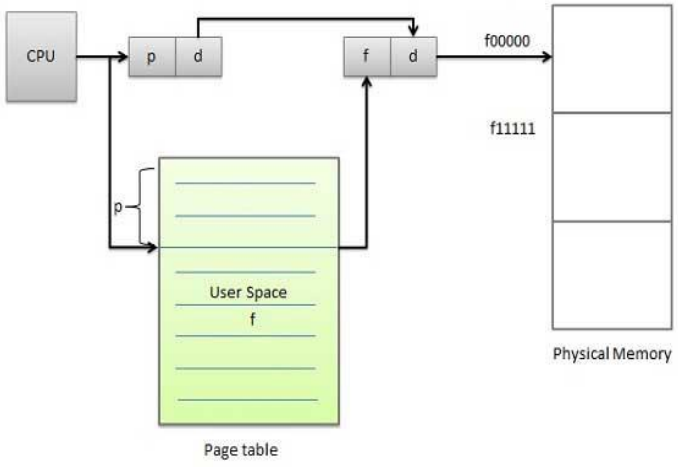
Paging is a memory management scheme used to avoid external fragmentation. Paging allows a process to be stored in a memory in a non-contiguous manner. Storing process in a non-contiguous manner solves the problem of external fragmentation.

For implementing paging the physical and logical memory spaces are divided into the same fixed-sized blocks. These fixed-sized blocks of physical memory are called frames, and the fixed-sized blocks of logical memory are called pages.

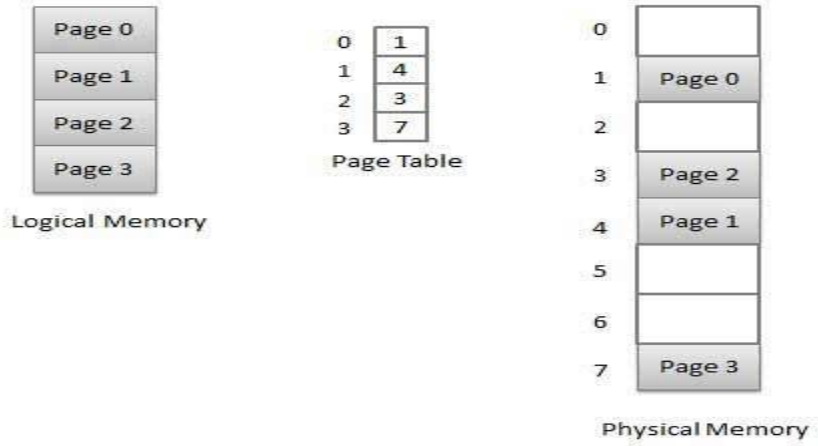
When a process needs to be executed the process pages from logical memory space are loaded into the frames of physical memory address space. Now the address generated by CPU for accessing the frame is divided into two parts i.e. page number and page offset.

Address generated by CPU is divided into

- **Page number (p)** -- page number is used as an index into a page table which contains base address of each page in physical memory.
- **Page offset (d)** -- page offset is combined with base address to define the physical memory address.



The **page table** uses page number as an index; each process has its separate page table that maps logical address to the physical address. The page table contains base address of the page stored in the frame of physical memory space. The base address defined by page table is combined with the page offset to define the frame number in physical memory where the page is stored.

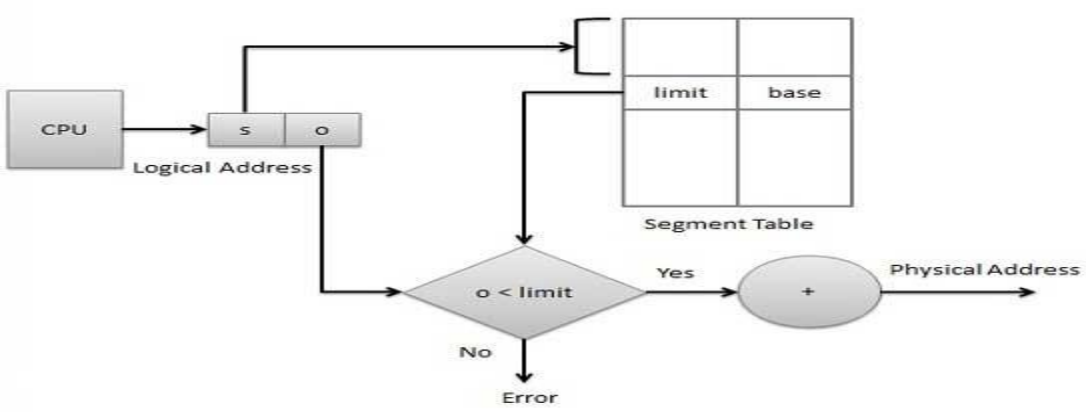


Segmentation

Like Paging, Segmentation is also a memory management scheme. It supports the user's view of the memory. The process is divided into the variable size segments and loaded to the logical memory address space.

The logical address space is the collection of variable size segments. Each segment has its name and length. For the execution, the segments from logical memory space are loaded to the physical memory space. Segmentation can be implemented using or without using paging. Address generated by CPU is divided into

- **Segment number (s)** -- segment number is used as an index into a segment table which contains base address of each segment in physical memory and a limit of segment.
- **Segment offset (o)** -- segment offset is first checked against limit and then is combined with base address to define the physical memory address.



The address specified by the user contain two quantities the segment name and the Offset. The segments are numbered and referred by the segment number instead of segment name. This segment number is used as an index in the segment table, and offset value decides the length or limit of the segment. The segment number and the offset together generate the address of the segment in the physical memory space.

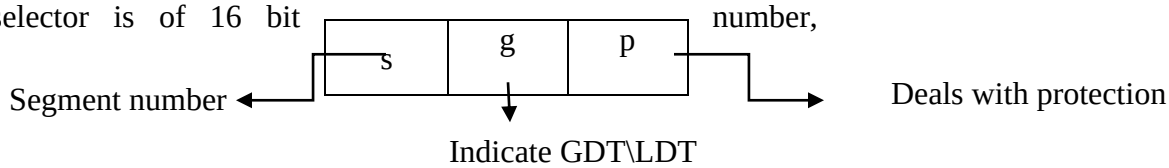
Paging	Segmentation
A page is of fixed block size.	A segment is of variable size.
Paging may lead to internal fragmentation.	Segmentation may lead to external fragmentation.
The user specified address is divided by CPU into a page number and offset.	The user specifies each address by two quantities a segment number and the offset (Segment limit).
The hardware decides the page size.	The segment size is specified by the user.
Paging involves a page table that contains base address of each page.	Segmentation involves the segment table that contains segment number and offset (segment length).

Segmentation with Paging

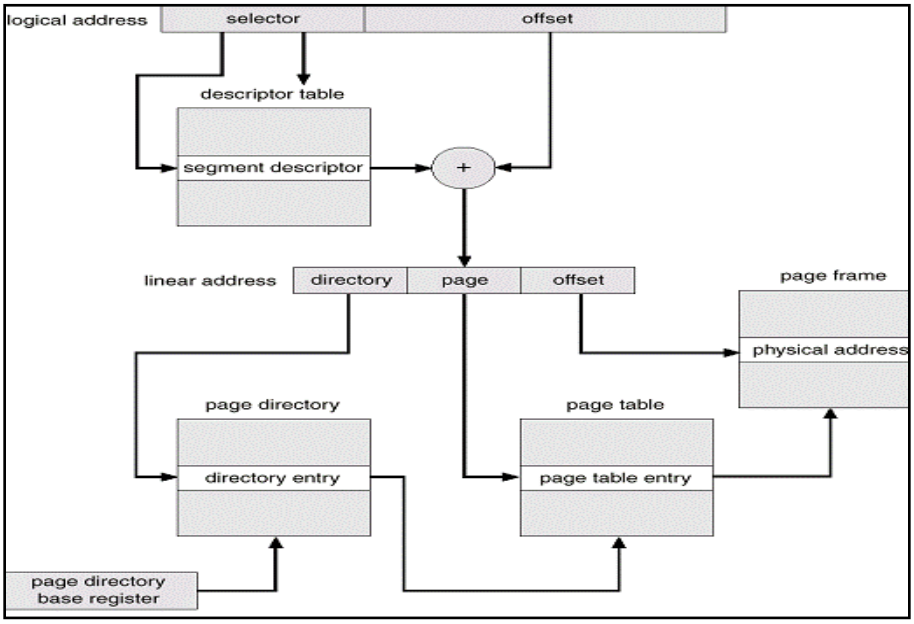
Both paging and segmentation have their advantages and disadvantages, it is better to combine these two schemes to improve on each. The combined scheme is known as 'Page the Elements'. Each segment in this scheme is divided into pages and each segment is maintained in a page table. The logical address space is divided into 2 partitions: first partition consists of up to 8KB segment private to that process (Local Descriptor table), second partition consists of 8kb segment shared among all the processes (Global Descriptor table).

The logical address is divided in to selector and offset

The selector is of 16 bit number,



The offset is of 32 bit number specifying the location of the byte (word) within the segment. The base and the limit information about the segment are used to generate linear address. The linear address is divided into page number and a page offset, further page number is divided in to page directory pointer and page table pointer.



Virtual memory

Memory is hardware that your computer uses to load the operating system and run programs. It consists of one or more RAM chips that each have several memory modules. The amount of real memory in a computer is limited to the amount of RAM installed. Common memory sizes are 256MB, 512MB, and 1GB.

Because your computer has a finite amount of RAM, it is possible to run out of memory when too many programs are running at one time. This is where virtual memory comes in. Virtual memory increases the available memory of your computer by enlarging the "address space," or places in memory where data can be stored. It does this by using hard disk space for additional memory allocation. However, since the hard drive is much slower than the RAM, data stored in virtual memory must be mapped back to real memory in order to be used.

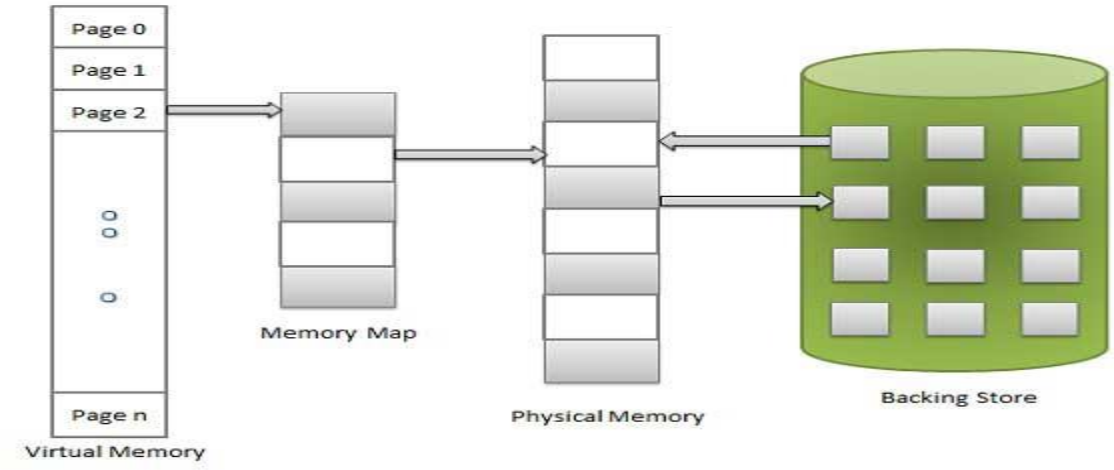
The process of mapping data back and forth between the hard drive and the RAM takes longer than accessing it directly from the memory. This means that the more virtual memory is used, the more it will slow your computer down.

Advantage: The primary advantage or objective of Virtual Memory systems is the ability to load and execute a process that requires a larger amount of memory than what is available by loading the process in parts and then executing them. The disadvantage is that Virtual Memory systems tend to be slow and require additional support from the system's hardware for address translations. It can be said that the execution speed of a process in a Virtual Memory system can equal, but never exceed, the execution speed of the same process with Virtual Memory turned off. Hence, we do not have an advantage with respect to the execution speed of the process. The advantage lies in the ability of the system to eliminate external fragmentation.

The other disadvantage of Virtual Memory systems is the possibility of Thrashing due to excessive Paging and Page faults. In may be noted that Trash Point is a point after which the execution of a process comes to a halt; the system is busier paging pages in and out of the memory than executing them.

Following are the situations, when entire program is not required to be loaded fully in main memory.

- User written error handling routines are used only when an error occured in the data or computation.
- Certain options and features of a program may be used rarely.
- Many tables are assigned a fixed amount of address space even though only a small amount of the table is actually used.
- The ability to execute a program that is only partially in memory would counter many benefits.
- Less number of I/O would be needed to load or swap each user program into memory.
- A program would no longer be constrained by the amount of physical memory that is available.
- Each user program could take less physical memory, more programs could be run the same time, with a corresponding increase in CPU utilization and throughput.



Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

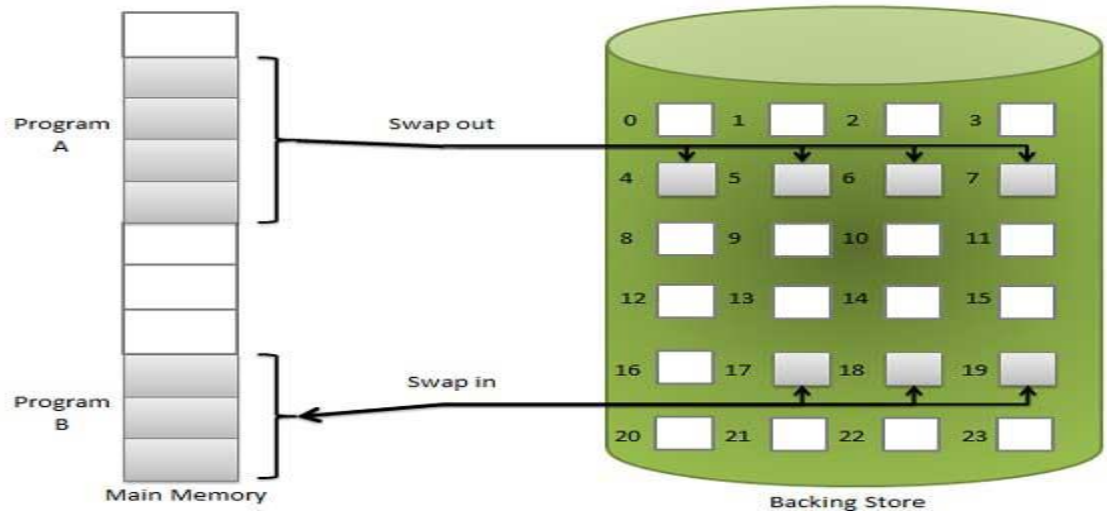
Demand Paging

A demand paging system is quite similar to a paging system with swapping. When we want to execute a process, we swap it into memory. Rather than swapping the entire process into memory, however, we use a lazy swapper called pager.

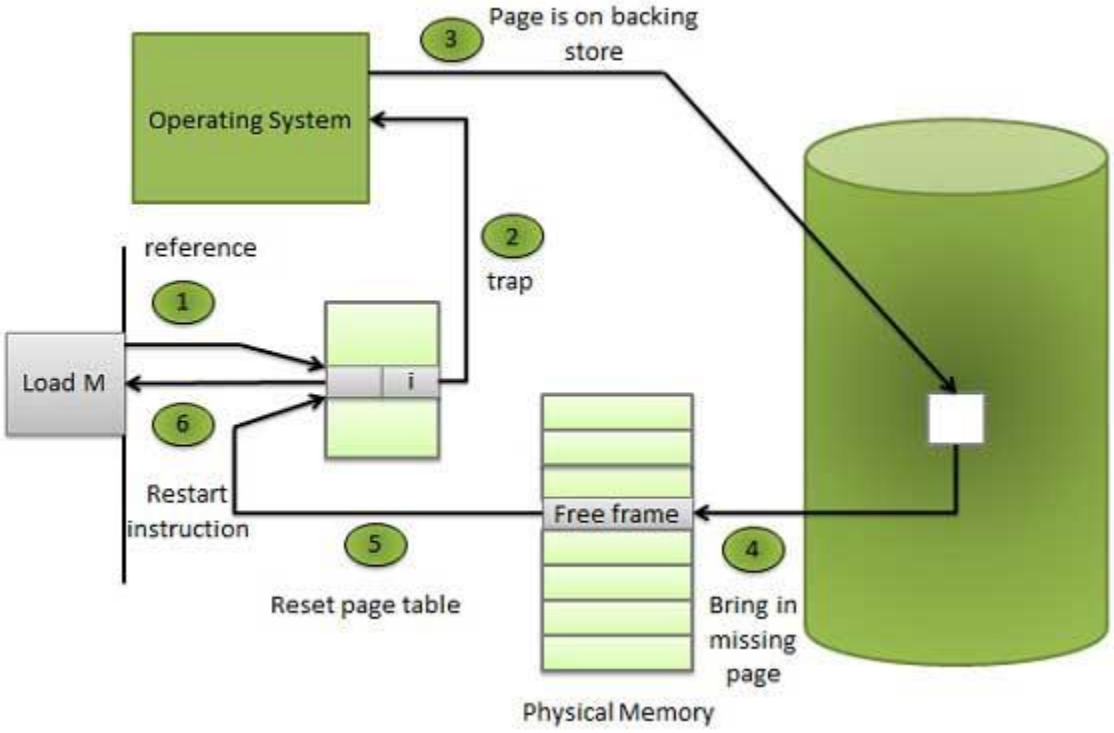
When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again. Instead of swapping in a whole process, the pager brings only those necessary pages into memory. Thus, it avoids reading into memory pages that will not be used in anyway, decreasing the swap time and the amount of physical memory needed.

Hardware support is required to distinguish between those pages that are in memory and those pages that are on the disk using the valid-invalid bit scheme. Where valid and invalid pages can be checked by checking the bit. Marking a page will have no effect if the process never attempts to access

the page. While the process executes and accesses pages that are memory resident, execution proceeds normally.



Access to a page marked invalid causes a **page-fault trap**. This trap is the result of the operating system's failure to bring the desired page into memory. But page fault can be handled as following



Step	Description
Step 1	Check an internal table for this process, to determine whether the reference was a valid or it was an invalid memory access.

Step 2	If the reference was invalid, terminate the process. If it was valid, but page have not yet brought in, page in the latter.
Step 3	Find a free frame.
Step 4	Schedule a disk operation to read the desired page into the newly allocated frame.
Step 5	When the disk read is complete, modify the internal table kept with the process and the page table to indicate that the page is now in memory.
Step 6	Restart the instruction that was interrupted by the illegal address trap. The process can now access the page as though it had always been in memory. Therefore, the operating system reads the desired page into memory and restarts the process as though the page had always been in memory.

Advantages

Following are the advantages of Demand Paging

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

Disadvantages

Following are the disadvantages of Demand Paging

- Number of tables and amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.
- Due to the lack of explicit constraints on jobs address space size.

Page Replacement Algorithm

In Demand Paging, only certain pages of a process are loaded initially into the memory. This allows us to get more number of processes into the memory at the same time. but what happens when a process requests for more pages and no free memory is available to bring them in. Following steps can be taken to deal with this problem:

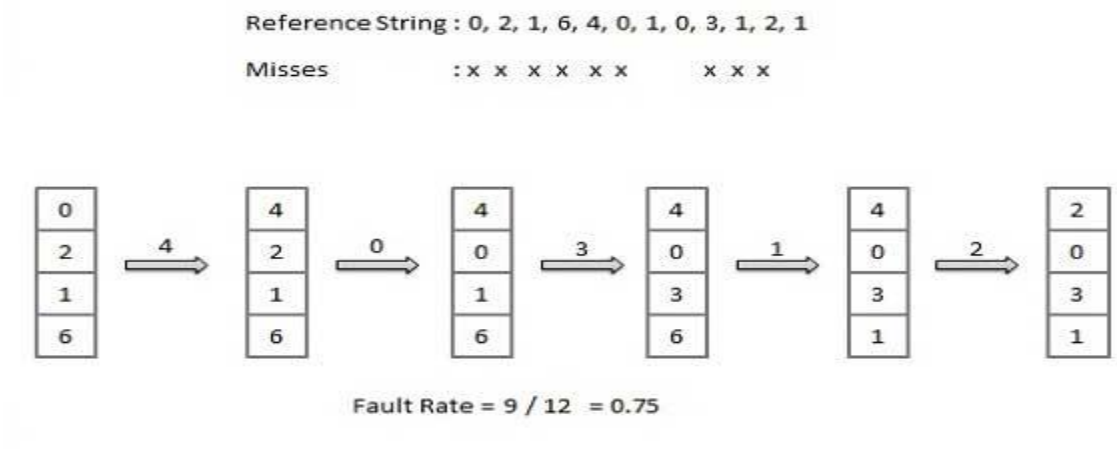
- Put the process in the wait queue, until any other process finishes its execution thereby freeing frames.
- Or, remove some other process completely from the memory to free frames.
- Or, find some pages that are not being used right now, move them to the disk to get free frames. This technique is called Page replacement and is most commonly used. We have some great algorithms to carry on page replacement efficiently.

Basic Page Replacement

- Find the location of the page requested by ongoing process on the disk.
- Find a free frame. If there is a free frame, use it. If there is no free frame, use a page-replacement algorithm to select any existing frame to be replaced, such frame is known as victim frame.
- Write the victim frame to disk. Change all related page tables to indicate that this page is no longer in memory.
- Move the required page and store it in the frame. Adjust all related page and frame tables to indicate the change.
- Restart the process that was waiting for this page.

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.

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.

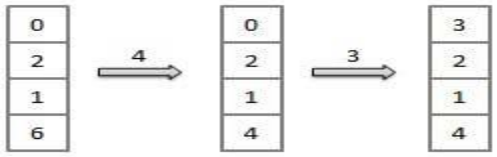


Belady's anomaly is the name given to the phenomenon where increasing the number of page frames results in an increase in the number of page faults for a given memory access pattern. This phenomenon is commonly experienced when using the First in First Out (FIFO) page replacement algorithm.

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 (looks for the time, when it again used). It is difficult to implement, because it requires future knowledge of the reference string.

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

Misses : x x x x x x

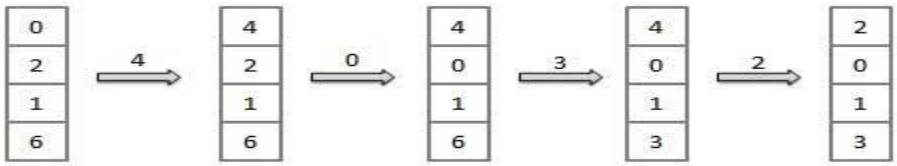


Fault Rate = 6 / 12 = 0.50

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



Fault Rate = 8 / 12 = 0.67

File System

File: A file is a named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks.In general, a file is a sequence of bits, bytes, lines or records whose meaning is defined by the files creator and user.

File Structure: File structure is a structure, which is according to a required format that operating system can understand.

- A file has a certain defined structure according to its type.
- A text file is a sequence of characters organized into lines.
- A source file is a sequence of procedures and functions.

- An object file is a sequence of bytes organized into blocks that are understandable by the machine.
- When operating system defines different file structures, it also contains the code to support these file structure. Unix, MS-DOS support minimum number of file structure.

File Type: File type refers to the ability of the operating system to distinguish different types of file such as text files source files and binary files etc. Many operating systems support many types of files. Operating system like MS-DOS and UNIX have the following types of files:

Ordinary files

- These are the files that contain user information.
- These may have text, databases or executable program.
- The user can apply various operations on such files like add, modify, delete or even remove the entire file.

Directory files

- These files contain list of file names and other information related to these files.

Special files:

- These files are also known as device files.
- These files represent physical device like disks, terminals, printers, networks, tape drive etc.

These files are of two types

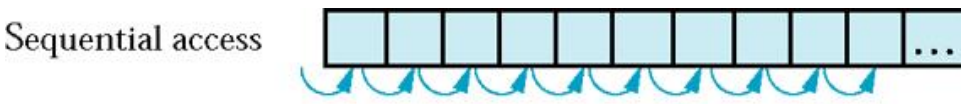
- **Character special files** - data is handled character by character as in case of terminals or printers.
- **Block special files** - data is handled in blocks as in the case of disks and tapes.

File Access Mechanisms: File access mechanism refers to the manner in which the records of a file may be accessed. There are several ways to access files

- Sequential access
- Direct/Random access
- Indexed sequential access

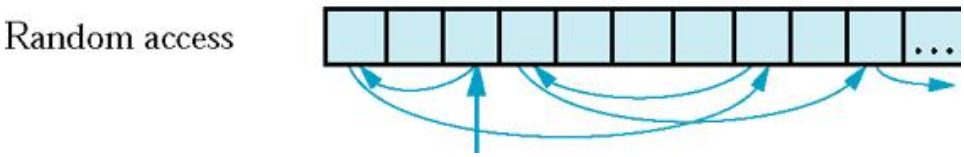
Sequential access

A sequential access is that in which the records are accessed in some sequence i.e the information in the file is processed in order, one record after the other. This access method is the most primitive one. Example: Compilers usually access files in this fashion.



Direct/Random access

- Random access file organization provides, accessing the records directly.
- Each record has its own address on the file with by the help of which it can be directly accessed for reading or writing.
- The records need not be in any sequence within the file and they need not be in adjacent locations on the storage medium.



Indexed sequential access

- This mechanism is built up on base of sequential access.
- An index is created for each file which contains pointers to various blocks.
- Index is searched sequentially and its pointer is used to access the file directly.

Directory: Information about files is maintained by Directories. A directory can contain multiple files. It can even have directories inside of them. In Windows we also call these directories as folders.

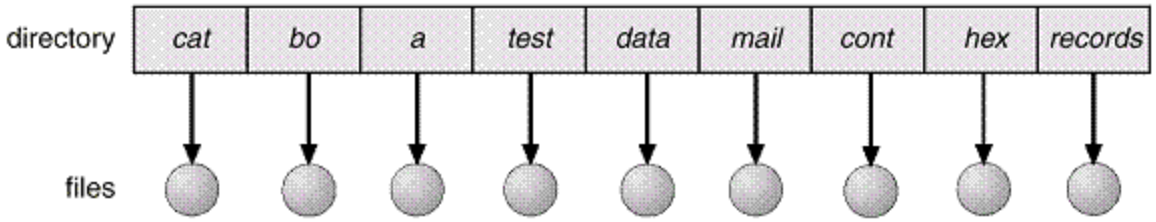
Directory structure: Directory structure which organizes and provides information about all the files in the system. The most common directory structures are:

- single-level directory
- two-level directory
- tree-structured directory
- acyclic directory

Single-Level Directory

In a single-level directory system, all the files are placed in one directory. This is very common on single-user OS's. A single-level directory has significant limitations, however, when the number of files increases or when there is more than one user. Since all files are in the same directory, they must have unique names. If there are two users who call their data file "test", then the unique-name rule is violated. Although file names are generally selected to reflect the content of the file, they are often quite limited in length.

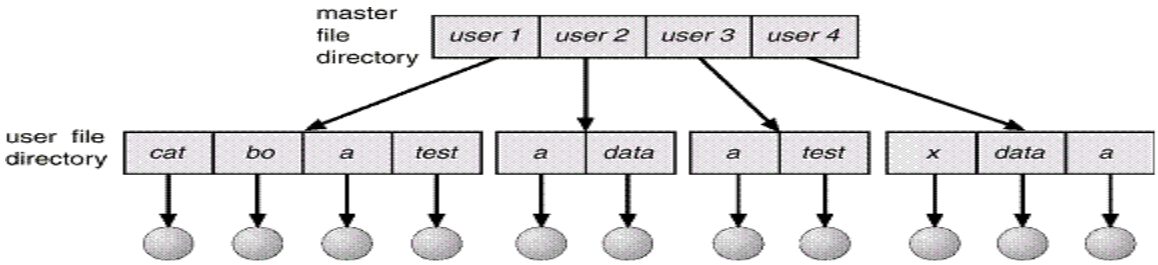
Even with a single-user, as the number of files increases, it becomes difficult to remember the names of all the files in order to create only files with unique names.



Two-Level Directory

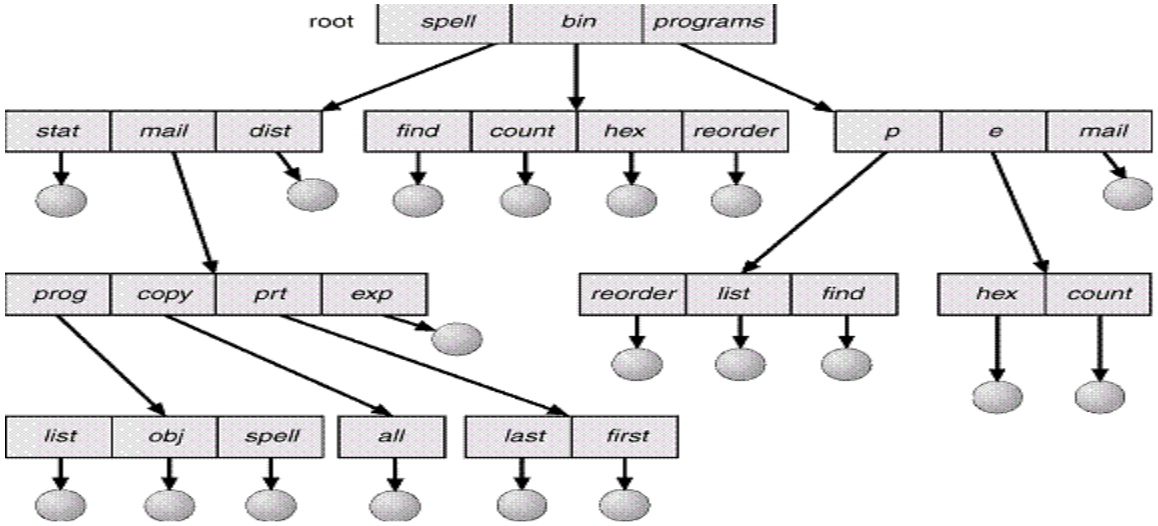
In the two-level directory system, the system maintains a master block that has one entry for each user. This master block contains the addresses of the directory of the users.

There are still problems with two-level directory structure. This structure effectively isolates one user from another. This is an advantage when the users are completely independent, but a disadvantage when the users want to cooperate on some task and access files of other users. Some systems simply do not allow local files to be accessed by other users.



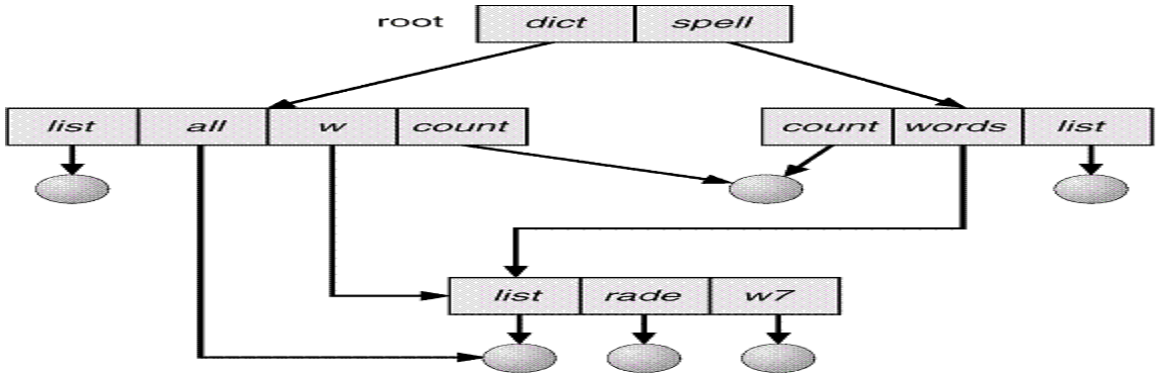
Tree-Structured Directories

In the tree-structured directory, the directory themselves are files. This leads to the possibility of having sub-directories that can contain files and sub-subdirectories.

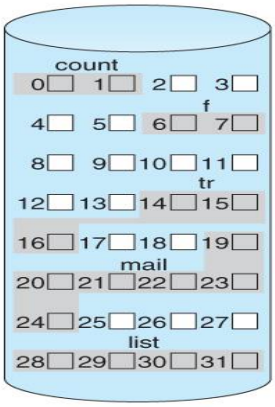


Acyclic-Graph Directories

The acyclic directory structure is an extension of the tree-structured directory structure. In the tree-structured directory, files and directories starting from some fixed directory are owned by one particular user. In the acyclic structure, this prohibition is taken out and thus a directory or file under directory can be owned by several users.



Space Allocation: Files are allocated disk spaces by operating system. Operating systems deploy following three main ways to allocate disk space to files.



directory

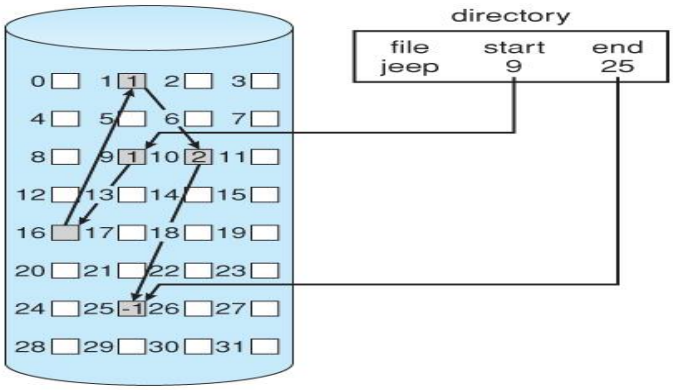
file	start	length
count	0	2
tr	14	3
mail	19	6
list	28	4
f	6	2

- Contiguous Allocation
- Linked Allocation
- Indexed Allocation

Contiguous Allocation

- Each file occupy a contiguous address space on disk.
- Assigned disk address is in linear order.
- Easy to implement.
- External fragmentation is a major issue with this type of allocation technique.

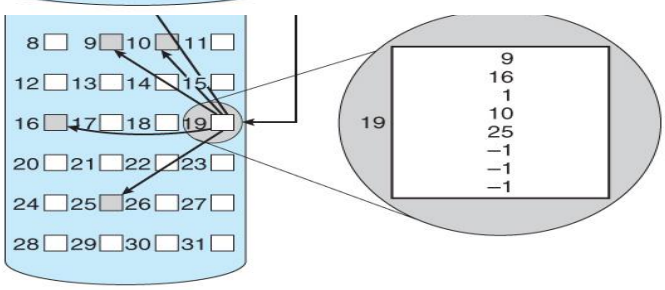
Linked Allocation



- Each file carries a list of links to disk blocks.
- Directory contains link / pointer to first block of a file.
- No external fragmentation
- Effectively used in sequential access file.
- Inefficient in case of direct access file.

Indexed Allocation

- Provides solutions to problems of contigous and linked allocation.



- A index block is created having all pointers to files.
- Each file has its own index block which stores the addresses of disk space occupied by the file.
- Directory contains the addresses of index blocks of files.

Free space management

Since there is only a limited amount of disk space, it is necessary to reuse the space from deleted files for new files. To keep track of free disk space, the system maintains a free-space list. The free-space list records all disk blocks that are free (i.e., are not allocated to some file). To create a file, the free-space list has to be searched for the required amount of space, and allocate that space to a new file. This space is then removed from the free-space list. When a file is deleted, its disk space is added to the free-space list.

Types of Operating Systems

1. Simple Batch System
2. Multiprogramming Batch System
3. Multiprocessor System
4. Desktop System
5. Distributed Operating System
6. Clustered System
7. Real time Operating System
8. Handheld System

Simple Batch Systems

- In this type of system, there is no direct interaction between user and the computer.
- The user has to submit a job (written on cards or tape) to a computer operator.
- Then computer operator places a batch of several jobs on an input device.
- Jobs are batched together by type of languages and requirement.
- Then a special program, the monitor, manages the execution of each program in the batch.
- The monitor is always in the main memory and available for execution.

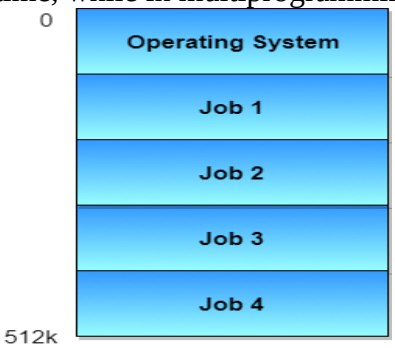
Following are some disadvantages of this type of system:

1. No interaction between user and computer.
2. No mechanism to prioritise the processes.

Multiprogramming Batch Systems

- In this the operating system picks up and begins to execute one of the jobs from memory.
- Once this job needs an I/O operation operating system switches to another job (CPU and OS always busy).
- Jobs in the memory are always less than the number of jobs on disk(Job Pool).
- If several jobs are ready to run at the same time, then the system chooses which one to run through the process of **CPU Scheduling**.
- In Non-multiprogrammed system, there are moments when CPU sits idle and does not do any work.
- In Multiprogramming system, CPU will never be idle and keeps on processing.

Time-Sharing Systems
They are very similar to Multiprogramming batch systems. In fact time sharing systems are an extension of multiprogramming systems. In time sharing systems the prime focus is on minimizing the response time, while in multiprogramming the prime focus is to maximize the CPU usage.



Multiprocessor Systems
A multiprocessor system consists of several processors that share a common physical memory. Multiprocessor system provides higher computing power and speed. In multiprocessor system all processors operate under single operating system. Multiplicity of the processors and how they do act together are transparent to the others.
Following are some advantages of this type of system.

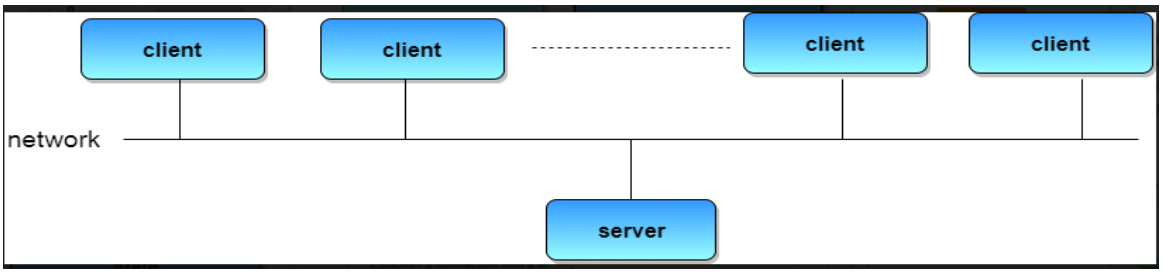
1. Enhanced performance
2. Execution of several tasks by different processors concurrently, increases the system's throughput without speeding up the execution of a single task.
3. If possible, system divides task into many subtasks and then these subtasks can be executed in parallel in different processors. Thereby speeding up the execution of single tasks.

Distributed Operating Systems
The motivation behind developing distributed operating systems is the availability of powerful and inexpensive microprocessors and advances in communication technology.
These advancements in technology have made it possible to design and develop distributed systems comprising of many computers that are inter connected by communication networks. The main benefit of distributed systems is its low price/performance ratio.
Following are some advantages of this type of system.

1. As there are multiple systems involved, user at one site can utilize the resources of systems at other sites for resource-intensive tasks.
2. Fast processing.
3. Less load on the Host Machine.

The two types of Distributed Operating Systems are: Client-Server Systems and Peer-to-Peer Systems.

Client-Server Systems: Centralized systems today act as server systems to satisfy requests generated by client systems. The general structure of a client-server system is depicted in the figure below:

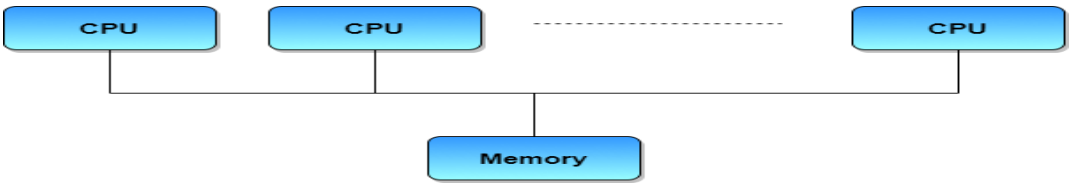


Server Systems can be broadly categorized as **compute servers** and **file servers**.

- **Compute-server systems** provide an interface to which clients can send requests to perform an action, in response to which they execute the action and send back results to the client.
- **File-server systems** provide a file-system interface where clients can create, update, read, and delete files.

Peer-to-Peer Systems: The growth of computer networks - especially the Internet and World Wide Web (WWW) – has had a profound influence on the recent development of operating systems. When PCs were introduced in the 1970s, they were designed for personal use and were generally considered standalone computers. With the beginning of widespread public use of the Internet in the 1980s for electronic mail and ftp many PCs became connected to computer networks.

In contrast to the **tightly coupled** systems, the computer networks used in these applications consist of a collection of processors that do not share memory or a clock. Instead, each processor has its own local memory. The processors communicate with one another through various communication lines, such as high-speed buses or telephone lines. These systems are usually referred to as loosely coupled systems (or distributed systems). The general structure of a client-server system is depicted in the figure below:



Real-Time Operating System

It is defined as an operating system known to give maximum time for each of the critical operations that it performs, like OS calls and interrupt handling.

Clustered Systems

- Like parallel systems, clustered systems gather together multiple CPUs to accomplish computational work.
- Clustered systems differ from parallel systems, however, in that they are composed of two or more individual systems coupled together.
- The definition of the term clustered is **not concrete;** the general accepted definition is that clustered computers share storage and are closely linked via LAN networking.
- Clustering is usually performed to provide **high availability**.
- A layer of cluster software runs on the cluster nodes. Each node can monitor one or more of the others. If the monitored machine fails, the monitoring machine can take ownership of its storage, and restart the application(s) that were running on the failed machine. The failed machine can remain down, but the users and clients of the application would only see a brief interruption of service.
- **Asymmetric Clustering -** In this, one machine is in hot standby mode while the other is running the applications. The hot standby host (machine) does nothing but monitor the active server. If that server fails, the hot standby host becomes the active server.
- **Symmetric Clustering -** In this, two or more hosts are running applications, and they are monitoring each other. This mode is obviously more efficient, as it uses all of the available hardware.
- **Parallel Clustering -** Parallel clusters allow multiple hosts to access the same data on the shared storage. Because most operating systems lack support for this simultaneous data access by multiple hosts, parallel clusters are usually accomplished by special versions of software and special releases of applications.

Clustered technology is rapidly changing. Clustered system use and features should expand greatly as **Storage Area Networks(SANs)**. SANs allow easy attachment of multiple hosts to multiple storage units. Current clusters are usually limited to two or four hosts due to the complexity of connecting the hosts to shared storage.

Handheld Systems

Handheld systems include **Personal Digital Assistants(PDAs)**, such as Palm-Pilots or Cellular Telephones with connectivity to a network such as the Internet. They are usually of limited size due to which most handheld devices have a small amount of memory, include slow processors, and feature small display screens.

- Many handheld devices have between **512 KB** and **8 MB** of memory. As a result, the operating system and applications must manage memory efficiently. This includes returning all allocated memory back to the memory manager once the memory is no longer being used.

- Currently, many handheld devices do **not use virtual memory** techniques, thus forcing program developers to work within the confines of limited physical memory.
- Processors for most handheld devices often run at a fraction of the speed of a processor in a PC. Faster processors require **more power**. To include a faster processor in a handheld device would require a **larger battery** that would have to be replaced more frequently.
- The last issue confronting program designers for handheld devices is the small display screens typically available. One approach for displaying the content in web pages is **web clipping**, where only a small subset of a web page is delivered and displayed on the handheld device.

Some handheld devices may use wireless technology such as **BlueTooth**, allowing remote access to e-mail and web browsing. **Cellular telephones** with connectivity to the Internet fall into this category. Their use continues to expand as network connections become more available and other options such as cameras and MP3 players, expand their utility.

Desktop Systems

Earlier, CPUs and PCs lacked the features needed to protect an operating system from user programs. PC operating systems therefore were neither **multiuser** nor **multitasking**. However, the goals of these operating systems have changed with time; instead of maximizing CPU and peripheral utilization, the systems opt for maximizing user convenience and responsiveness. These systems are called **Desktop Systems** and include PCs running Microsoft Windows and the Apple Macintosh. Operating systems for these computers have benefited in several ways from the development of operating systems for **mainframes**.

Microcomputers were immediately able to adopt some of the technology developed for larger operating systems. On the other hand, the hardware costs for microcomputers are sufficiently **low** that individuals have sole use of the computer, and CPU utilization is no longer a prime concern. Thus, some of the design decisions made in operating systems for mainframes may not be appropriate for smaller systems.