UNIT-1 (Introduction)

Operating System

- Operating system is a system program that act as an interface between hardware and user.
- It manages system resources.
- It provides a platform on which other application programs are installed.

- Core Part of Operating System is called kernel.

Goals of operating System

Primary Goal: To make system user friendly (convenient).

Secondary Goal: To make system efficient.

Role of Operating System: We can explore operating Systems from two viewpoints that of the user and that of the system.

User's View: In user's view operating system is designed to maximize the work (CPU utilization) that the user is performing. In this context OS is designed mostly for **ease of use** with some attention paid to performance and none paid to resource utilization.

System's View: In system point of view operating system is a program that works as a **resource allocator**. Every computer system has many resources like CPU Time, Memory Space, File Storage Space, I/O Device and so on. The operating system act as manager of these resources.

Functions of Operating System

Operating System is responsible for following activities.

Process Management: Operating System is responsible for following activities in connection with Process Management.

- ✓ Scheduling Processes and threads on the CPUs.
- ✓ Creating and deleting both user and system processes.
- ✓ Suspending and resuming processes.
- ✓ Providing mechanisms for process synchronization.
- ✓ Providing mechanism for process communication.

Memory Management: Operating System is responsible for following activities in connection with Memory Management.

- ✓ Keeping track of which parts of memory are currently being used and by whom.
- ✓ Deciding which processes and data to move into and out of memory.
- ✓ Allocating and de-allocating memory space as needed.

Disk Management: Operating System is responsible for following activities in connection with Disk Management.

- ✓ Free space management in disk.
- ✓ Storage Allocation in disk.
- ✓ Disk Scheduling.

File Management: Operating System is responsible for following activities in connection with File Management.

- ✓ Creating and deleting files.
- ✓ Creating and deleting directories to organize files.
- ✓ Supporting primitives for manipulating files and directories.
- ✓ Mapping files onto secondary storage.
- ✓ Backing up files on stable storage media.

I/O Device Management: Operating System is responsible for I/O Device management needed by various processes.

Network Management Operating System is responsible for network management.

Security and Protection: Operating System also provides security and protection to computer system.

Services of Operating System

An operating system provides an environment for execution of programs. It provides mainly following services to programs and to the users.

User Interface: Almost all operating systems provide user interface to the users.

Program Execution: Operating system provides the service of program execution. Program is loaded into the memory and CPU is assigned to it for its execution.

Access I/O Devices: Operating system provides I/O devices to running program.

File System Access: Operating system provides services like read and write files and directories, search for a given file and directory, access and deny to files and directories etc to the running program.

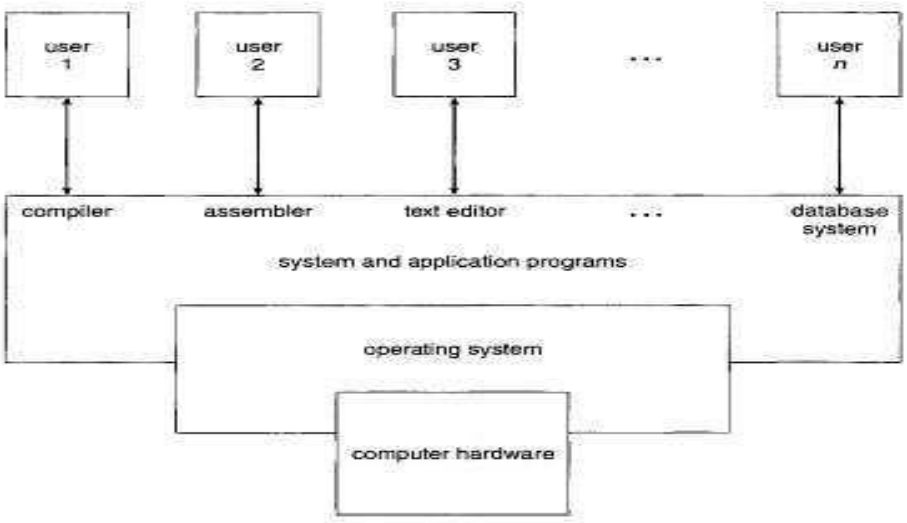
Error Detection and Response: Errors may be generated by the user programs or hardware. Operating system should take the appropriate action to ensure correct and consistent computing.

Communication: There are many circumstances in which one process needs to exchange information with another process. Such communication may occur between processes that are executing on same computer or between processes that are executing on different computer systems tied together by a computer network communication may be implemented via shared memory or through message passing by the operating system.

Components of Computer System

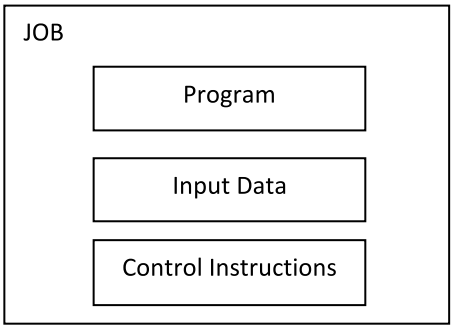
A computer system canbe roughly divided into following four components.

1. **Hardware:** Basic computing resources like CPU, Memory, I/O Devices etc.
2. **Application Programs:** Programs that are used to solve users' computing problems e.g. web browsers, compilers, word processors etc.
3. **Operating System:** It controls the hardware and coordinates its use among various application programs for different users. Operating system is similar to government; just like government it provides an environment within which other programs can do useful work.
4. **Users:** Users perform the computation with the help of application program.



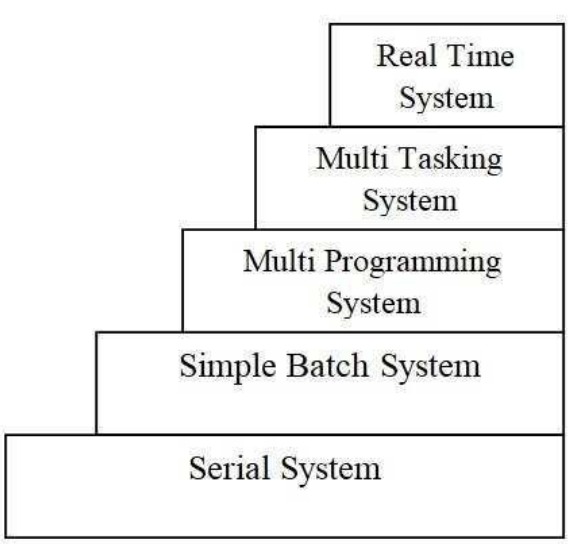
Abstract view of the components of a computer system.

JOB: When we combined program, input data and control instructions together, it is called Job.



Classification of Operating System

- Decreasing Memory Size
- Increasing waiting time And Response time
- Decreasing speed of CPU
- Decreasing CPU utilization
- Increasing CPU idle time

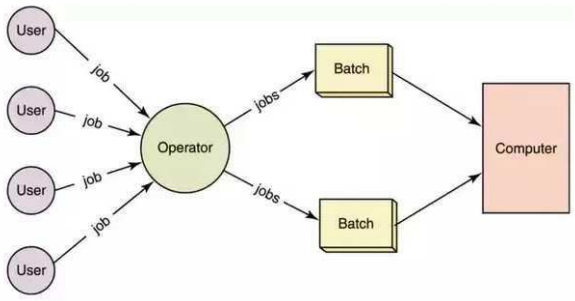


Serial System

In this system, Operating System is not needed. Programs interact directly with computer hardware.

Batch System

- Batch means set of jobs with similar need.
- In this system, Operator makes Batch of jobs and loads it into punch card.
- Similar types of jobs batch together and execute at a time.



Advantages

- ✓ Save time from activities like loading compiler.
- ✓ No manual intervention by user is needed.

Disadvantages

- ✓ Limited Memory

- ✓ Interaction of I/O directly with CPU.
- ✓ CPU is often idle.

Multiprogramming Operating System

- ❖ Main objective of multiprogramming is to maximize CPU utilization.
- ❖ More than one process can reside in main memory which are ready to execute i.e. more than one process is in ready state.
- ❖ In this system if any running process performs I/O or other event which does not require CPU, then instead of sitting idle, CPU performs context switching and picks another process for execution.

Advantages

- ✓ High CPU utilization.
- ✓ Less waiting time, response time etc.
- ✓ Useful in current scenario when load is high.

Disadvantages

- ✓ Process scheduling is difficult.
- ✓ Main memory management is required.
- ✓ Problems like memory fragmentation may occur.

Time Sharing or Multi-Tasking System

- ❖ It is also called fair share system or multi programming with round robin System.
- ❖ It is extension of multi Programming System.
- ❖ In this system CPU switches between processes so quickly that it gives an illusion that all executing at same time.
- ❖ This system can be used to handle multiple interactive tasks.

Advantages

- ✓ High CPU utilization.
- ✓ Less waiting time, response time etc.
- ✓ Useful in current scenario when load is high.

Disadvantages

- ✓ Process scheduling is difficult.
- ✓ Main memory management is required.
- ✓ Problems like memory fragmentation may occur.

Multiprocessing System

- ❖ A system is called multiprocessing system if two or more CPU within a single computer communicate with each other and share system bus memory and I/O devices.
- ❖ It provides true parallel execution of processes.

Type of Multiprocessing system

1. **Asymmetric Multiprocessing System:** In this system, each processor is assigned a specific task. A master processor controls the system. The other processors called slave processors either look to master for instruction or have predefined tasks i.e. master-slave relationship hold in this type of system.
2. **Symmetric Multiprocessing System:** In this system each processor performs all tasks within the operating system i.e. all processor are peers, no master-slave relationship exists between processors.

Advantages:

- ✓ **Increased Throughput:** By increasing number of processors, we expect to get more work done in less time.
- ✓ **Economy of Scale:** multiprocessor system can cost less than equivalent multiple single processor systems.
- ✓ **Increased Reliability:** Since work is distributed among several processors; failure of one processor will not halt the system only slow it down.

Disadvantages

- ✓ It is complex system.
- ✓ Process scheduling is difficult in this system.
- ✓ It required large size of Main memory.
- ✓ Overhead reduces throughput.

Real Time System

- A real time system has well-defined, fixed time constraints. Processing must be done within the defined constraints, or the system will fail.
- A real time system functions correctly only if it returns the correct result within its time constraints.
- This system is used when we require quick response against input.
- In this system, task is completed within specified time.
- It is classified in following two types-

1. **Hard Real Time System:** In hard Real Time system task will be fail if response time against input is more than specified time.
2. **Soft Real Time System:** In Soft Real Time system inaccurate result will be found if response time against input is more than specified time.

Interactive System

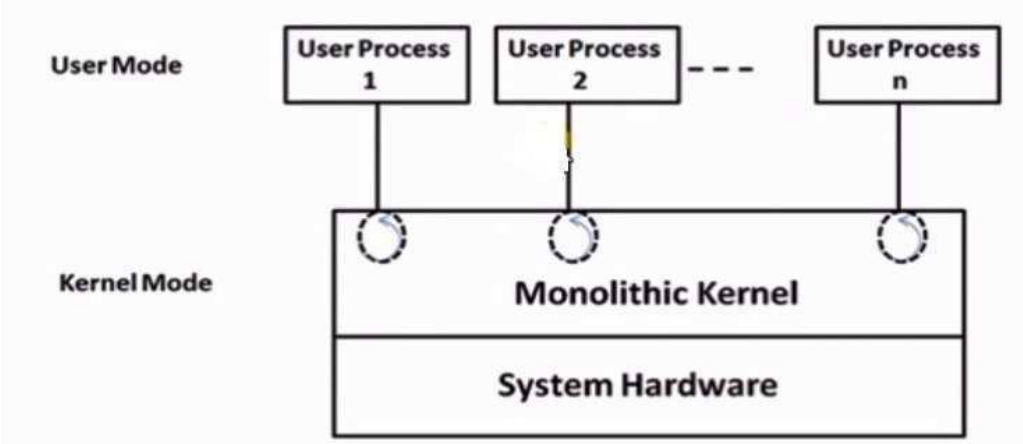
Time sharing requires an interactive computer system, which provides direct communication between the user and the system. The user gives instructions to the Operating System or to a program directly, using an input device such as a keyboard or a mouse, and waits for immediate results on an output device. Response time in interactive system should be short.

Multi-user operating system

Multi-user operating system is a computer operating system that allows multiple users on different computers or terminals to access a single system with one OS on it. These programs are often quite complicated and must be able to properly manage the necessary tasks required by the different users connected to it. The users will typically be at terminals or computers that give them access to the system through a network, as well as other machines on the system such as printers. **Example** Ubuntu,Unix etc.

Monolithic Kernel

- Monolithic means all in one piece.
- In monolithic kernel user services and kernel services are implemented under same address space, It increases size of the kernel thus increase size of operating system. Example: Linux



Advantage

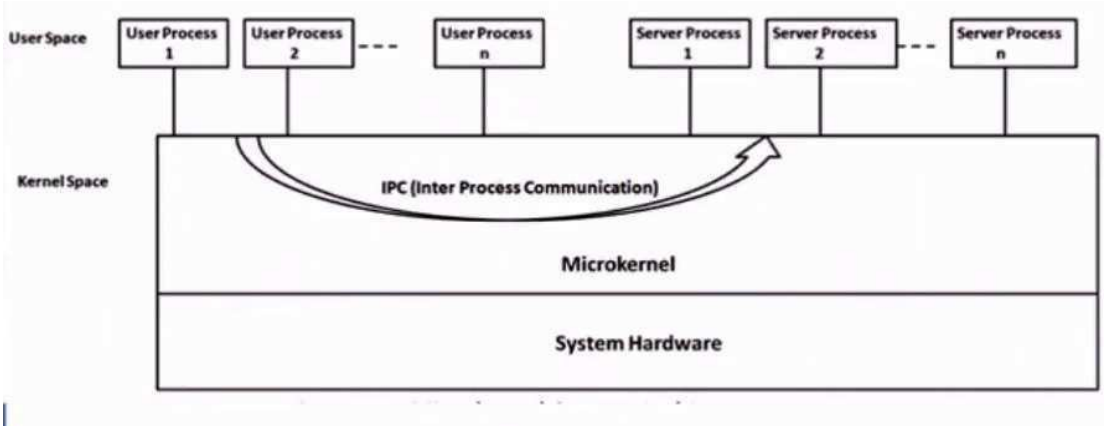
- ✓ Execution is faster.

Disadvantages

- ✓ Size is larger.
- ✓ Hard to extend.
- ✓ Hard to port.
- ✓ More prone to errors and bugs.

Microkernel

- In microkernel, user services and kernel services are implemented in different address space, therefore it also reduces the size of kernel as well as the size of Operating system.
- In this architecture, only the most important services are present inside kernel and rest of the operating system services are present inside system application program.
- Communication between client process and services running in user address space is established through message passing thus reduce the speed of execution.
  Example: Mach OS



Advantages

- ✓ Size is smaller.
- ✓ Easy to extend.
- ✓ Easy to port.
- ✓ Less prone to errors and bugs.

Disadvantage

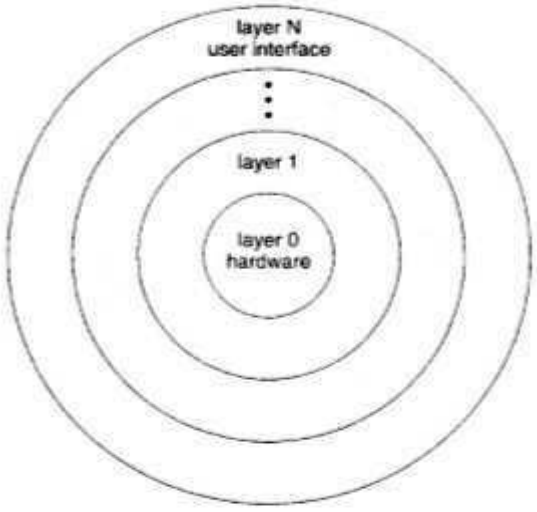
- ✓ Execution is slower.

Difference between microkernel and monolithic kernel

Parameter	**Microkernel**	**Monolithic Kernel**
Size	Small	Large
Speed of execution	Faster	Slower
Extendibility	Easy	Hard
Portability	Easy	Hard
Error prone capability	Less	More

Layered Kernel

- In layered approach, the OS consists of several layers where each layer has a well-defined functionality and each layer is designed, coded and tested independently.
- Lowest layer interacts and deal with underlying hardware and top most layer provides an interface to the application programs and to the user processes.
- Each layer relies on the service of the layer below it. Communication takes place between adjacent layers.
- Each layer knows what services are provided by the layer above it but the details about how the services are provided are hidden.
- Different layers can be file management layer, process management layer, memory management layer etc.



A layered operating system.

Advantages

- ✓ Easier to add new feature or make change in one layer without affecting the other layer.
- ✓ Easy to add new layer when required.
- ✓ A particular layer can be debugged correctly or redesigned without affecting other layers.

Disadvantages

- ✓ Overhead incurred to maintain is more, if no of layers is more.
- ✓ If functionality of layers are not properly divided it may cause low system performance.

Reentrant Kernel

- A reentrant kernel is one that consists of executable code stored in memory such that several processes can use this code at same time without hindering the working of other processes.
- Thus in reentrant kernel several processes may be executed in kernel mode at same time.
- Since multiple concurrent processes execute in kernel mode therefore they also share data that may affect the functionality of other processes. Hence, to avoid this condition, reentrant function and locking mechanism is used.
- Reentrant functions are one which allow processes to modify only the local variables and do not affect global data variables.
- In locking mechanism several processes can execute reentrant function concurrently but only one process at a time can execute non reentrant function, this will ensure that global data is modified by only one process at a time and thus each process has same copy of global data that is being shared among them.
  Example: UNIX.

Multithreading

Thread: Light weight Process is called Thread.

Thread Vs Process

Thread	Process
Light weight Process	Heavy Weight Process
Thread Switching does not need interaction with OS	Process Switching needs interaction with OS
Thread can share Code segment, Data Section, File Descriptor, address space, heap etc	Processes own their separate Code segment, Data Section, File Descriptor, address space, heap etc

Threads has its own

Threads of same process share

Note: Local variables of the Process are stored in Stack, Global variables are stored in Data Section and dynamically created variables are stored in Heap.

User Level Thread vs. Kernel Level Thread

User Level Thread	Kernel Level Thread
Implementation is easy	Implementation is difficult
OS don't recognized user level thread	Kernel Level Thread is recognized by OS
Context switch is fast because it requires no hardware support and less information to be stored during context switching.	Context switch is slow because it requires hardware support and more information to be stored during context switching.
If one user level thread performs blocking operation then entire process will be blocked.	If one kernel thread perform blocking operation then another thread can continue execution

Manavendra Pratap Singh-[M.Tech-IIIT Allahabad]

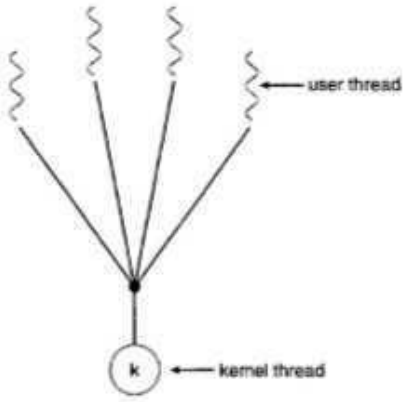
Benefits of Multithreading Programming

1. **Responsiveness**: Multithreading allow a program to continuerunning even part of it is blocked or is performing a lengthy operation, thereby increasing responsiveness to user.
2. **Resource Sharing:** Since in multithreading multiple threads of a process share code and data that enables an application to have several different threads of activity within the same address space.
3. **Economy:** Allocating memory and resources for process creation is costly because thread share resource of the process to which they belong, it is more economical to create and context switch thread.
4. **Scalability:** Benefit of multithreading can be greatly increased in a multiprocessor architecture, where threads may be running on different processors.

Multithreading Models

Many to One Model

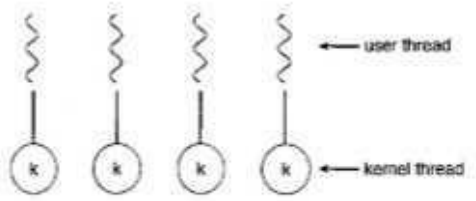
- It maps many user level threads to one kernel level thread.
- In this model Entire Process is blocked if a thread makes blocking system call.
- Since only one thread can access the kernel at a time multiple threads are unable to run in parallel on multiprocessor.



Many-to-one model.

One to One Model

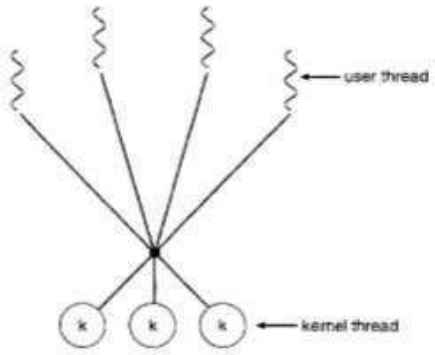
- It maps each user thread to a kernel thread.
- It provides more concurrency because it allows another thread to run in parallel on multiprocessor.
- It allows multiple threads to run in parallel on multiple processors.
- Creating a user thread requires creating the corresponding kernel thread.



One-to-one model.

Many to Many Model

- It maps many user level threads to a smaller or equal number of kernel level threads.
- Developer can create as many user level threads as required.
- Kernel level threads can run in parallel on a multiprocessor.
- When a thread performs blocking system call the kernel can schedule another thread for execution.



Many-to-many model.

UNIT I

Introduction : Operating system and functions, Classification of Operating systems- Batch, Interactive, Time sharing, Real Time System, Multiprocessor Systems, Multiuser Systems, Multiprocess Systems, Multithreaded Systems, Operating System Structure- Layered structure, System Components, Operating System services, Re-entrant Kernels, Monolithic and Microkernel Systems.

UNIT-2

Process Synchronization

On the basis of synchronization, processes are categorized as one of the following two types:
Independent Process- Execution of one process does not affect the execution of other processes.
Cooperative Process- Execution of one process affects the execution of other processes.

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

Example1-Let P1 and P2 are cooperative processes and x is a shared variable.

x=5← Shared variable

P1	P2
A=x; A++; sleep (1); x=A;	B=x; B--; sleep (1); x=B;

We are expecting the final value of xwill be 5 but due to race condition final value of x will be either 4 or 6.

Example2-Let P1 and P2 are cooperative processes and x is a shared variable.

x=5← Shared variable

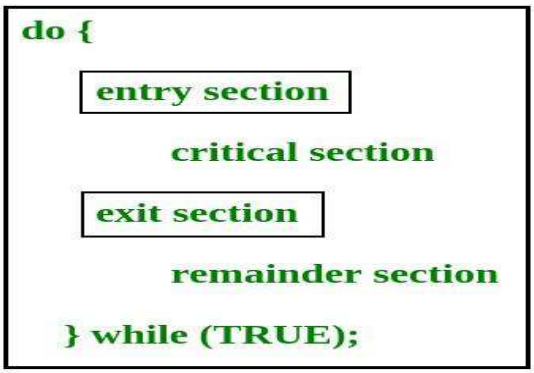
P1	P2
A=2*x Print(A)	B=x+2 Print(B)

We can get more than one output like {10,12},{7,14},{12,10} and {14,7} due to race condition for above cooperative processes.

Critical Section

The critical section is a code segment where the shared variables can be accessed. Atomic action is required in a critical section i.e. only one process can execute in its critical section at a time. All the other processes have to wait to execute in their critical sections.

The critical section is given as follows:



Entry section handles the entry into the critical section. It acquires the resources needed for execution by the process. In the entry section, the process requests for entry in the **Critical Section.**
Exit section handles the exit from the critical section. It releases the resources and also informs the other processes that critical section is free.

Requirements of solution to critical section Problem-

Any solution to the critical section problem must satisfy three requirements:

1. **Mutual Exclusion**
   Mutual exclusion implies that only one process can be inside the critical section at any time. If any other processes require the critical section, they must wait until it is free. It is essential requirement that must be satisfied by the solution of critical section problem.
2. **Progress**
   Progress means that if a process is not using the critical section, then it should not stop any other process from accessing it. In other words, any process can enter a critical section if it is free. It is essential requirement that must be satisfied by the solution of critical section problem.
3. **Bounded Waiting**
   A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

Dekker's algorithm

Dekker's algorithm is the first known correct solution to the mutual exclusion problem in concurrent programming.If two processes attempt to enter a critical section at the same time, the algorithm will allow only one process in, based on whose turn it is. If one process is already in the critical section, the other process will busy wait for the first process to exit. This is done by the use of two flags, wants_to_enter[0] and wants_to_enter[1], which indicate an intention to enter the critical section on the part of processes 0 and 1, respectively, and a variable turn that indicates who has priority between the two processes. Dekker's algorithm guarantees mutual exclusion, freedom from deadlock, and freedom from starvationOne advantage of this algorithm is that it doesn't require special test-and-set (atomic read/modify/write) instructions and is therefore highly portable between languages and machine architectures. One disadvantage is that it is limited to two processes.

1st Version of Dekker's Solution-

P1	P2
while(true) **{** **while(turn!=0);** CS **turn=1;** RS **}**	**while(true)** **{** **while(turn!=1);** CS **turn=0;** RS **}**

Mutual Exclusion is assured but Progress is not assured in this solution.

2nd Version of Dekker's Solution-

P1	P2
while(true)	**while(true)**
{	**{**
flag[0]=true;	**flag[1]=true;**
while(flag[1])	**while(flag[0])**
CS	CS
flag[0]=false;	**flag[1]=false;**
RS	RS
}	**}**

Mutual Exclusion is assured and Progress is assured in this solution but deadlock may occur in this solution.

Peterson's Solution

- Peterson's Solution is a classical **software based solution** to the critical section problem.
- In Peterson's solution, we have two shared variables:
  - boolean flag[i] :Initialized to FALSE, initially no one is interested in entering the critical section
  - int turn : The process whose turn is to enter the critical section.

P1	P2
while(true) **{** **flag[0]=true;** **turn=1;** **while(turn==1&&flag[1]==true);** CS **flag[0]=false;** RS **}**	**while(true)** **{** **flag[1]=true;** **turn=0;** **while(turn==0&&flag[0]==true);** CS **flag[1]=false;** RS **}**

Peterson's Solution preserves all three conditions:

- Mutual Exclusion is assured as only one process can access the critical section at any time.
- Progress is also assured, as a process outside the critical section does not block other processes from entering the critical section.
- Bounded Waiting is preserved as every process gets a fair chance.

Disadvantages of Peterson's Solution

- It involves Busy waiting
- It is limited to 2 processes.

Test and Set

Test and Set are a **hardware solution** to the synchronization problem. In Test and Set, we have a shared lock variable which can take either of the two values, 0 or 1.

0 Unlock
1 Lock

Before entering into the critical section, a process inquires about the lock. If it is locked, it keeps on waiting till it become free and if it is not locked, it takes the lock and executes the critical section.
In Test and Set, Mutual exclusion and progress are preserved but bounded waiting cannot be preserved.

The **test-and-set** instruction is an instruction used to write 1 (set) to a memory location and return its old value as a single atomic (i.e., non-interruptible) operation. If multiple processes may access the same memory location, and if a process is currently performing a test-and-set, no other process may begin another test-and-set until the first process's test-and-set is finished. A CPU may use a test-and-set instruction offered by another electronic component
A lock can be built using an atomic test-and-set instruction as follows:

```
function Lock(boolean *lock) {
   while (test_and_set(lock) == 1); }
```

The calling process obtains the lock if the old value was 0, otherwise the while-loop spins waiting to acquire the lock.

Semaphore

A semaphore S is an integer variable that, apart from initialization, is accessed only through two standard atomic operations: wait () and signal (). The wait () operation is also called P and signal () is also called V.

Definition of wait ()

```
wait(S) {
                while(S <= 0);
                S--;
        }
```

Definition of signal ()

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

All modifications to the integer value of the semaphore in the wait () and signal () operations must be executed indivisibly. That is, when one process modifies the semaphore value, no other process can simultaneously

Application of Semaphore

1. Solving Multi process Critical Section Problem
2. Resource allocation among various processes
3. Ordering Execution of processes.

Types of Semaphore

1. **Binary Semaphore -** This is also known as **mutex lock**. It can have only two values – 0 and 1. Its value is initialized to 1. It is used to implement solution of critical section problem with multiple processes.
2. **Counting Semaphore -** Its value can range over an unrestricted domain. It is used to control access to a resource that has multiple instances.

-
Producer Consumer Problem

Problem Statement -We have a buffer of fixed size. A producer can produce an item and can place in the buffer. A consumer can pick items and can consume them. We need to ensure that when a producer is placing an item in the buffer, then at the same time consumer should not consume any item. In this problem, buffer is the critical section.
To solve this problem, we need two counting semaphores – Full and Empty. "Full" keeps track of number of items in the buffer at any given time and "Empty" keeps track of number of unoccupied slots.

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

Solution for Producer -

```
do{

        //produce an item

        wait(empty);
        wait(mutex);

        //place in buffer

        signal(mutex);
        signal(full);

}while(true);
```

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

Solution for Consumer -

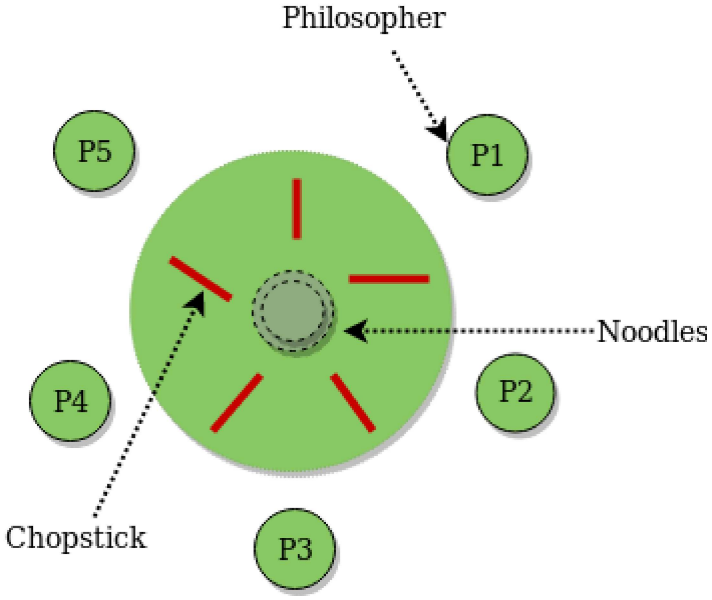
```
do{

        wait(full);
        wait(mutex);

        // remove item from buffer

        signal(mutex);
        signal(empty);

        // consumes item

}while(true);
```

As the consumer is removing an item from buffer, therefore the value of "full" is reduced by 1 and the value is mutex is also reduced so that the producer cannot access the buffer at this moment. Now, the

consumer has consumed the item, thus increasing the value of "empty" by 1. The value of mutex is also increased so that producer can access the buffer now.

The Dining Philosopher Problem

Problem Statement- The Dining Philosopher Problem states that K philosophers seated around a circular table with one chopstick between each pair of philosophers. There is one chopstick between each philosopher. A philosopher may eat if he can pick up the two chopsticks adjacent to him. One chopstick may be picked up by any one of its adjacent followers but not both.



Semaphore Solution to Dining Philosopher –
Each philosopher is represented by the following pseudocode:

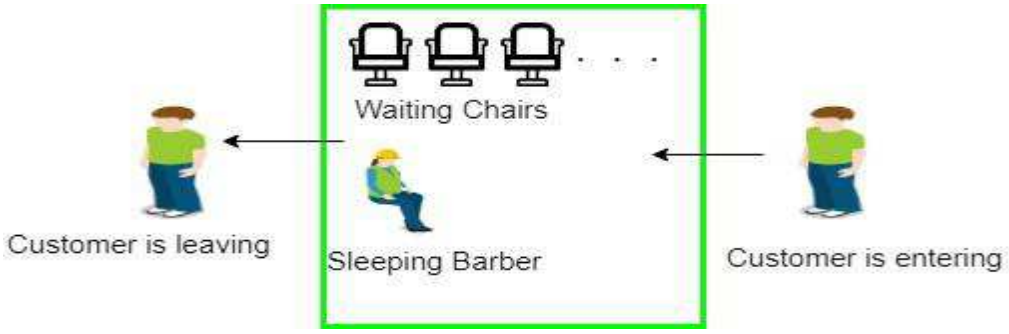
```
process P[i]
 while true do
{            THINK;
             PICKUP(CHOPSTICK[i], CHOPSTICK[i+1 mod 5]);
             EAT;
             PUTDOWN(CHOPSTICK[i], CHOPSTICK[i+1 mod 5])
  }
```

There are three states of philosopher: **THINKING, HUNGRY and EATING**. Here there are two semaphores: Mutex and a semaphore array for the philosophers. Mutex is used such that no two philosophers may access the pickup or putdown at the same time. The array is used to control the behavior of each philosopher. But, semaphores can result in deadlock due to programming errors.

Sleeping Barber Problem

Problem Statement- The analogy is based upon a hypothetical barber shop with one barber. There is a barber shop which has one barber, one barber chair, and n chairs for waiting for customers if there are any to sit on the chair.

- If there is no customer, then the barber sleeps in his own chair.
- When a customer arrives, he has to wake up the barber.
- If there are many customers and the barber is cutting a customer's hair, then the remaining customers either wait if there are empty chairs in the waiting room or they leave if no chairs are empty.



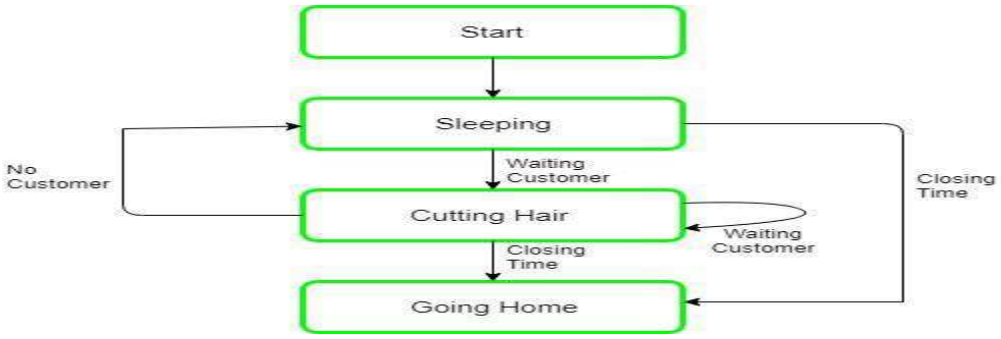
Solution - The solution to this problem includes three semaphores. First is for the customer which counts the number of customers present in the waiting room (customer in the barber chair is not included because he is not waiting). Second, the barber 0 or 1 is used to tell whether the barber is idle or is working, And the third mutex is used to provide the mutual exclusion which is required for the process to execute. In the solution, the customer has the record of the number of customers waiting in the waiting room if the number of customers is equal to the number of chairs in the waiting room then the upcoming customer leaves the barbershop.

When the barber shows up in the morning, he executes the procedure barber, causing him to block on the semaphore customers because it is initially 0. Then the barber goes to sleep until the first customer comes up.

When a customer arrives, he executes customer procedure the customer acquires the mutex for entering the critical region, if another customer enters thereafter, the second one will not be able to anything until the first one has released the mutex. The customer then checks the chairs in the waiting room if waiting customers are less than the number of chairs then he sits otherwise he leaves and releases the mutex.

If the chair is available then customer sits in the waiting room and increments the variable waiting value and also increases the customer's semaphore this wakes up the barber if he is sleeping.

At this point, customer and barber are both awake and the barber is ready to give that person a haircut. When the haircut is over, the customer exits the procedure and if there are no customers in waiting room barber sleeps.



Inter process communication

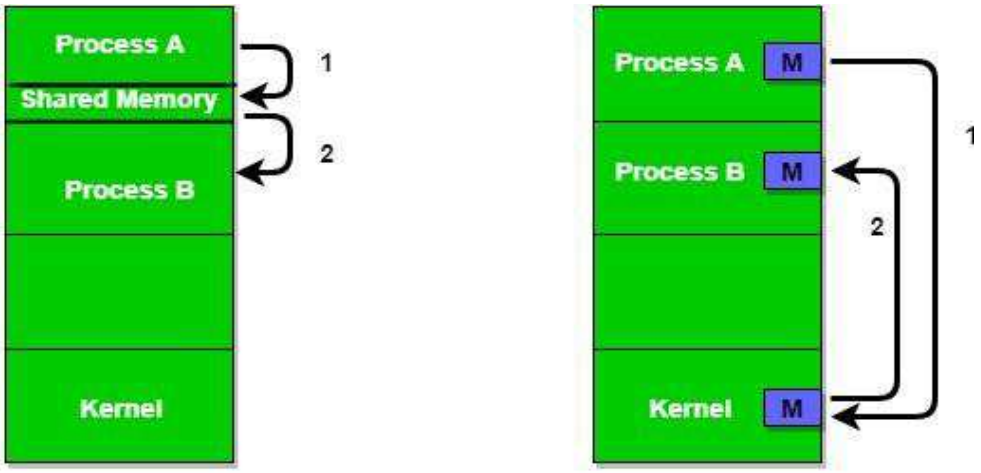
Inter process communication (IPC) is a mechanism which allows processes to communicate each other and synchronize their actions. Processes can communicate with each other using these two ways:

1. Shared Memory
2. Message passing

Shared Memory Method

Shared memory requires processes to share some variable and it completely depends on how programmer will implement it.

One way of communication using shared memory can be- Suppose process1 and process2 are executing simultaneously and they share some resources or use some information from other process, process1 generate information about certain computations or resources being used and keeps it as a record in shared memory. When process2 need to use the shared information, it will check in the record stored in shared memory and take note of the information generated by process1 and act accordingly. Processes can use shared memory for extracting information as a record from other process as well as for delivering any specific information to other process.

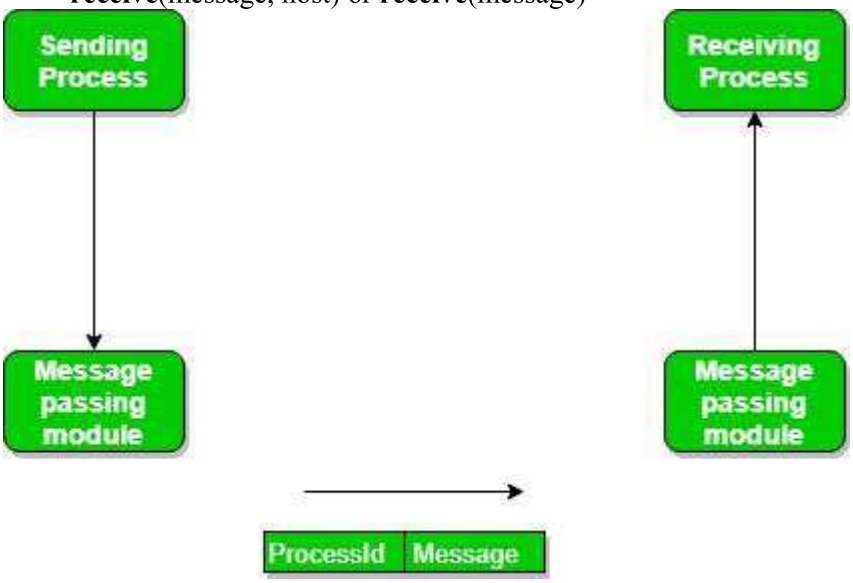


Messaging Passing Method

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

- Establish a communication link (if a link already exists, no need to establish it again.)

- Start exchanging messages using basic primitives.
  We need at least two primitives:
  – **send**(message, destinaion) or **send**(message)
  – **receive**(message, host) or **receive**(message)

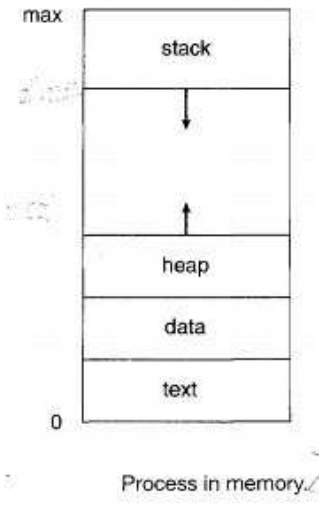


The message size can be of fixed size or of variable size. If it is of fixed size, it is easy for OS designer but complicated for programmer and if it is of variable size then it is easy for programmer but complicated for the OS designer. A standard message can have two parts: **header and body.** The **header part** is used for storing Message type, destination id, source id, and message length and control information. The control information contains information like what to do if runs out of buffer space, sequence number, priority. Generally, message is sent using FIFO style.

UNIT-3

Process

- Program in execution is called process.
- Process is active entity while program is passive entity.
- Process is smallest unit of work individually scheduled by operating system.



Process in memory.

Process Control Block/PCB

- PCB holds all the information needed to keep track of a process.
- It is a data structure maintained by Operating System.
- OS creates PCB for every process.
- It is useful in multi programming environment.

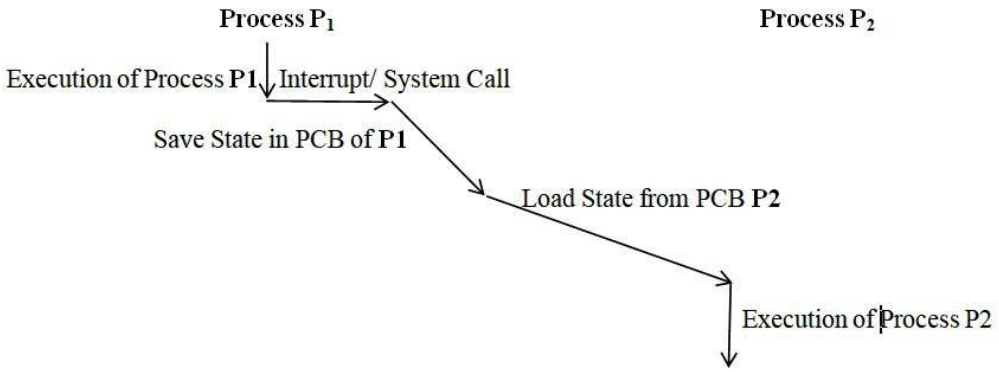
Pointer
Process ID
Program Counter
CPU Scheduling Information
Accounting Information
I/O Status Information
. .

Where:

- ✓ Pointer holds address of parent process
- ✓ Process ID holds unique identification number for process.

- ✓ Program counter holds address of next instructions to be executed.
- ✓ CPU scheduling information holds information like priority of process.
- ✓ Accounting Information holds the information of amount of CPU used.
- ✓ I/O status information contains the information of I/O devices allocated to process.

Process State switching Diagram

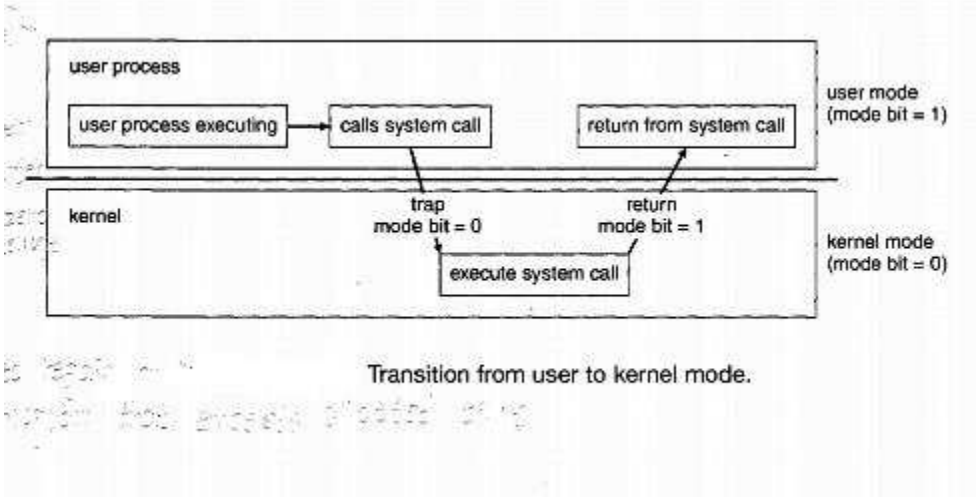


- ❖ Process P1 execution starts in user mode when it needs an event to occur like I/O Request, the process sends an interrupt to CPU, CPU stores the state of process P1 in its PCB and load state of P2 from its PCB. After loading state of P2,execution of process P2 begins.
- ❖ Execution of Process occurs in User Mode while Save and Load of PCB occurs in Kernel Mode.

Dual-Mode Operation

- Two separate modes of operations; user mode and kernel mode (also called supervisor mode or system mode or privileged mode) are used.
- Mode bit is added to the hardware of the computer to indicate the current mode. (0 for kernel mode 1 for user mode).
- When a computer system is executing on behalf of a user application, the system is in user mode, however when a user application requests a service from the operating system (via system call) it must transition from user to kernel mode to fulfill the request.
- The dual mode of operations provides us with the means for protecting the operating system from errant users and errant users from one another.
- Hardware allows privileged instructions to be executed only in kernel mode. If an attempt is made to execute privileged instruction in user mode, hardware does not execute the instruction but rather treats it asillegal and trap it to operating system.

  Example: At system boot time, hardware starts in kernel modeOS is then loaded and starts user applications in user mode. When a trap or interrupts occurs, the hardware switches from user mode to kernel mode (change the state of mode bit to 0). Thus, whenever OS gains control of the computer, it is in kernel mode (change the state of mode bit to 0) before passing control to a user program.



Transition from user to kernel mode.

Life Cycle of Process

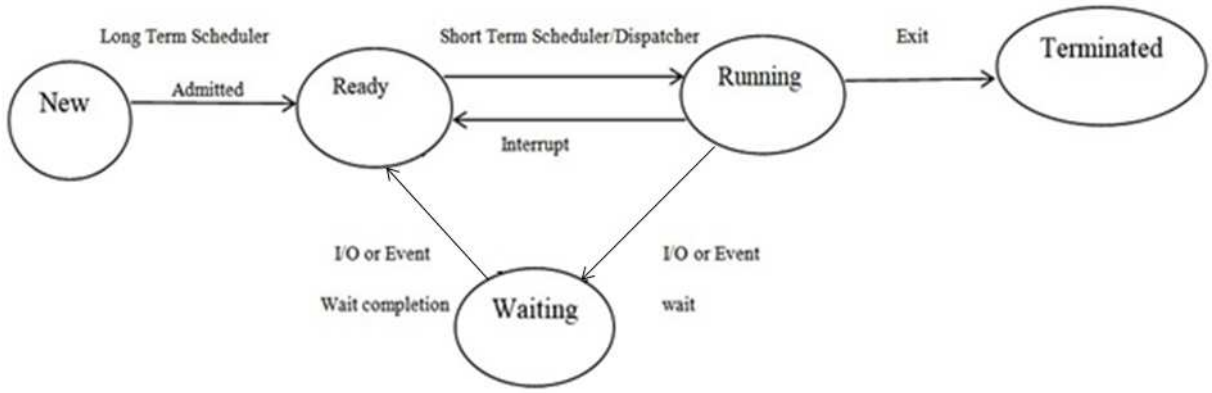
New State: When a process is created it is in new state. In new state process resides in Job Queue (Secondary Memory).

Ready State: State of process is called Ready State when Process resides in Ready Queue (Primary Memory) and waiting for CPU.

Running State: State of process is called Running state if CPU is assigned to process.

Waiting State: State of process is called Waiting State when Process resides in waiting Queue (Primary Memory) and waiting for some event like I/O to occur.

Terminated State: State of process is called Terminated State when the Process has completed its execution successfully.



Schedulers

Long Term Scheduler/Job Scheduler

- It selects process from Job Queue and assigns it to Ready Queue.
- It changes state of processes from New State to Ready State.

Short Term Scheduler/CPU Scheduler

- It selects process from Ready Queue and assigns it to CPU.
- It changes state of processes from Ready State to Running State.
- Dispatcher is responsible for saving the context of one process (i.e. content of PCB) and loading the context of another process.

Medium Term Scheduler

- It removes the processes from Main Memory.
- It reduces degree of multiprogramming.
- It is responsible for swapped out process.

Dispatcher: Dispatcher is part of Short Term Scheduler that performs following functions.

- Context switching
- Switching to User Mode
- Jumping to the proper location in the user program to restart that program

Queues used in Process Scheduling

Job Queue: It contains all processes of system.

Ready Queue: It contains all the processes that reside in Main Memory and ready to be executed.

Waiting Queue: It contains all the processes that are waiting for I/O.

Note: Job Queue, Ready Queue and waiting Queue all are implemented using Linked List.

CPU SCHEDULING

Removal of the running process from the CPU and selection of another process on the basis of particular strategy is called CPU Scheduling.

Non Preemptive Vs Preemptive

Non Preemptive Scheduling or Cooperative Scheduling: In non-preemptive scheduling, once the CPU is allocated to a process, keeps the CPU until it releases the CPU either by terminating or by switching to the waiting state i.e. when a process reaches to running state it can either switches to waiting state or terminates.

Preemptive Scheduling: Preemptive scheduling allows a process to switch from running state to ready state or waiting state to ready state. In Preemptive scheduling, once the CPU is allocated to a process, process can release the CPU before terminating the process.

Scheduling Performance Criteria

Criteria that are made for comparing scheduling algorithms are given below:

CPU Utilization: We want to keep the CPU as busy as possible. Conceptually CPU utilization range from 0 to 100 percent while in real system it should range from 40 to 90 percent.

Throughput: Number of processes that can completed per time Unit is called Throughput.

Turnaround Time: The interval from time of submission of a process to time of completion is called Turnaround Time of that process.

Waiting Time: Sum of periods spent waiting by a process in the ready queue is called waiting time of the process.

Response Time: Time interval from submission of a request until the first response is produced is called Response Time.

NOTE: It is desirable to:

- ✓ maximize CPU Utilization and Throughput
- ✓ minimize Response Time, Waiting Time and Turnaround Time

Manavendra Pratap Singh-[M.Tech-IIIT Allahabad]

Types of Scheduling Algorithm

1. First Come First Served Scheduling (FCFS)
2. Shortest Job First Scheduling (SJF)
3. Priority Scheduling
4. Round Robin
5. Multilevel Queue Scheduling
6. Multilevel Feedback Queue Scheduling

FCFS or First Come First Served Scheduling

- FCFS Scheduling is non Preemptive it means in FCFS Scheduling once the CPU has been allocated to a process, that process keeps the CPU until it releases the CPU either by terminating or by requesting I/O.
- In FCFS scheduling the process that request the CPU is allocated the CPU first.

Advantages:

- Easy to implement
- Easy to understand

Disadvantages:

- Average waiting time in FCFS Scheduling is often quite longer.
- Sometimes convey effect may occur in FCFS Scheduling.

Example: ……

Shortest Job First (Preemptive and Non-Preemptive)/

Shortest Remaining Time First(Preemptive) /

Shortest Next CPU Burst Scheduling(Preemptive)

- SJF scheduling can be either preemptive or non-preemptive.
- In SJF CPU is assigned to the process that has the smallest next CPU Burst.
- If the next CPU burst of two processes is the same FCFS Scheduling is used to break the tie.
- This approach gives minimum average waiting time for a given set of processes.

Example: ……

Priority Scheduling

- In this scheduling a priority is associated with each process and CPU is allocated to the process with the highest priority.
- Equal priority processes are scheduled in FCFS order.
- SJF scheduling is special case of priority scheduling where the priority is inverse of the next CPU burst.
- Priority scheduling can be either preemptive or non-preemptive.
- Major problem with Priority Scheduling is problem of starvation.
- Solution of the problem starvation is aging, where aging is a technique of gradually increasing the priority of the processes that wait in the system from long time.

Example: ……

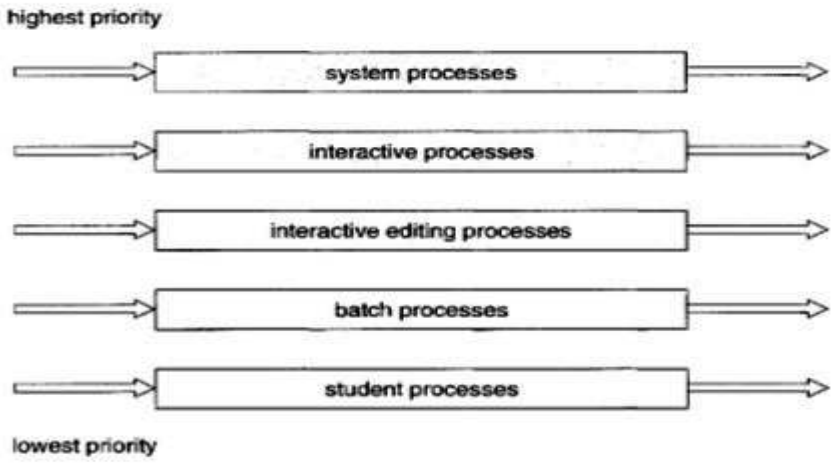
Round Robin Scheduling

- It is designed especially for time sharing system or multi-tasking system.
- It is similar to FCFS scheduling but preemption is added to enable the system to switch between processes.
- In this technique ready queue is treated as circular queue. CPU scheduler goes around the ready queue allocating the CPU to each process for a time interval up to 1 time-quantum (or time-slice).
- Round Robin scheduling is preemptive
- This approach gives minimum average response time for a given set of processes.

Example: ……

Multilevel Queue Scheduling

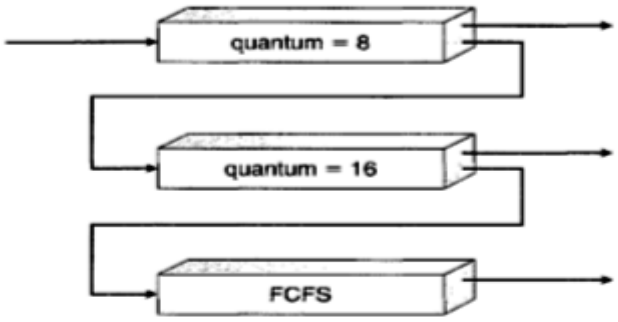
Multilevel Queue Scheduling partitions the ready queue into several separate queues. The processes are permanently assigned to one queue generally based on some properties of the process such as process type, process priority, memory size etc. Each queue has its own scheduling algorithm. CPU is assigned to the queues either on the basis of their priority or time-slice among the queues.



Multilevel queue scheduling.

Multilevel Feedback Queue Scheduling

Multilevel Feedback Queue Scheduling allows a process to move between queues. The idea is to separate processes according to the characteristics of their CPU burst. If a process uses too much CPU time it will be moved towards a lower priority queue and if a process that stayed too long in a lower priority queue may be moved to a higher priority queue this form of aging prevents starvation.



Multilevel feedback queues.

Manavendra Pratap Singh-[M.Tech-IIIT Allahabad]

Interrupt

Interrupt is a signal that is generated by any hardware or software component to CPU to indicate that some event has occurred.

There are two types of interrupt.

1. Hardware Interrupt: When interrupt is generated by any hardware component.
2. Software Interrupt: When interrupt is generated by any software component.

When an event occurs, an interrupt signal is sent to the CPU.CPU compares the priority of the interrupt with the currently executing process. If the priority of the interrupt is higher, then CPU invokes the corresponding Interrupt Service Routine (ISR) which handles the interrupt with the help of system calls.

System Call

System call is programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on.

Example: Fork(),Join(),Suspend(),Resume(),Block(), etc.

Fork() System Call :

- It is use to create new Process (Child Process).
- With the help of Fork() system call, sequence of instructions are divided into two concurrently executing sequence of instructions.
- After creation of the process both parent and child process starts execution from the next instruction.
- It returns 0 to child process and process-id of child process to the parent process.

NOTE: In a process, if we call the Fork() function n-times then, $2^{(n-1)}$ child process will be generated.

Deadlock

In a multiprogramming environment, several processes may compete for a finite number of resources. A process request resources: if the resources are not available at that time, the process enters a waiting state. Sometimes, a waiting process is never again able to change state, because the resources it has requested are held by other waiting processes. This situation is called deadlock.

System Model: A system consists of a finite number of resources to be distributed among a number of competing processes. The resources are partitioned into several types, each consisting of some number of identical instances. A process must request a resource before using it and must release the resource after using it.

Request: The process requests the resource. If the request cannot be granted immediately then the requesting process must wait until it can acquire the resource.

Use: The process can operate on the resource.

Release: The process releases the resource.

Necessary conditions for deadlock

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

Resource allocation Graph

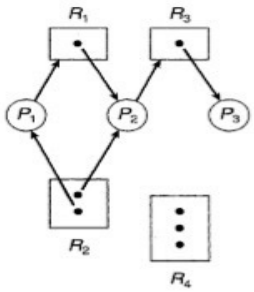
In Resource allocation graph we represent--

- Each process P_i as circle and each resource type R_j as a rectangle. Each instance of resource type R_j is represented by dot within the rectangle.
- A directed edge from process P_i to resource type R_j is denoted by $P_i \rightarrow R_j$ If process P_i has requested an instance of resource type R_j.

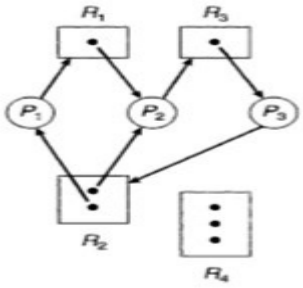
Manavendra Pratap Singh-[M.Tech-IIIT Allahabad]

- A directed edge from process R_j to resource type P_i is denoted by $R_j \rightarrow P_i$ If an instance of resource type R_j has been allocated to process P_i.
- 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.

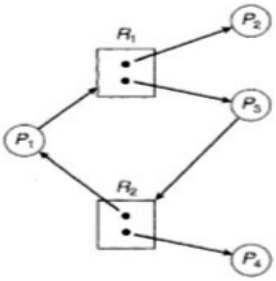
Example1: No cycle with No Deadlock



Example 2: Cycle with Deadlock



Example 3: Cycle with No Deadlock



Methods for handling Deadlocks

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

- We can use a protocol to prevent or avoid deadlocks, ensuring that the system will never enter a deadlocked state (Deadlock Prevention or Avoidance).
- We can allow the system to enter a deadlocked state, detect it, and recover (Detect and resolve).
- We can ignore the problem altogether and pretend that deadlocks never occur in the system (Ignorance).
    - ✓ To ensure that deadlocks never occur, deadlock prevention scheme, provides a set of methods for ensuring that at least one of the necessary conditions cannot hold.
    - ✓ To ensure that deadlocks never occur, deadlock avoidance scheme, requires that the operating system be given in advance additional information concerning which resources a process will request and use during its lifetime. With this additional knowledge, it can decide for each request whether or not the process should wait. To decide whether the current request can be satisfied or must be delayed, the system must consider the resources currently available, the resources currently allocated to each process, and the future requests and releases of each process.

Deadlock Prevention

To ensure that deadlocks never occur, deadlock prevention scheme, provides a set of methods for ensuring that at least one of the necessary conditions cannot hold.

Mutual Exclusion

The mutual-exclusion condition must hold for nonsharable resources. For example, a printer cannot be simultaneously shared by several processes. Sharable resources, in contrast, do not require mutually exclusive access and thus cannot be involved in a deadlock. Read-only files are a good example of a sharable resource. If several processes attempt to open a read-only file at the same time, they can be granted simultaneous access to the file. A process never needs to wait for a sharable resource. In general, however, we cannot prevent deadlocks by denying the mutual-exclusion condition, because some resources are intrinsically nonsharable.

Hold and Wait

To ensure that the hold-and-wait condition never occurs in the system, we must guarantee that, whenever a process requests a resource, it does not hold any other resources.

- ✓ One protocol that can be used requires each process to request and be allocated all its resources before it begins execution.

- ✓ An alternative protocol allows a process to request resources only when it has none. A process may request some resources and use them. Before it can request any additional resources, however, it must release all the resources that it is currently allocated.

No Preemption

To ensure that this condition does not hold, we can use the following protocol. If a process is holding some resources and requests another resource that cannot be immediately allocated to it then all resources the process is currently holding are preempted. The process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting.

Circular Wait

One way to ensure that this condition never holds is to impose a total ordering of all resource types and to require that each process requests resources in an increasing order of enumeration. let R = { R1, R2, ... , Rm} be the set of resource types. We assign to each resource type a unique integer number, which allows us to compare two resources and to determine whether one precedes another in our ordering.

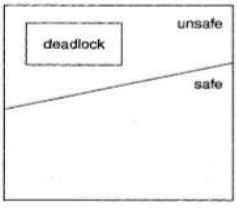
Deadlock Avoidance

Deadlock avoidance require additional information about how resources are to be requested. We can use one of the following two methods for deadlock avoidance.

1. Resource allocation graph algorithm
2. Banker's Algorithm.

Safe State

- ❖ A state is safe if the system can allocate resources to each process (up to its maximum) in some order and still avoid a deadlock. i.e. a system is in a safe state only if there exists a safe sequence.
- ❖ If safe sequence not exists, then the system state is said to be unsafe.
- ❖ A safe state is not a deadlocked state. Conversely, a deadlocked state is an unsafe state. Not all unsafe states are deadlocks.



Resource allocation graph algorithm

- In this algorithm we will make resource allocation graph for system on the basis of allocation and request of resources by the processes and algorithm checks cycle exists or not in the graph.
- If no cycle exists, then the allocation of the resource will leave the system in a safe state. If a cycle is found, then the allocation will put the system in an unsafe state. In that case, process P; will have to wait for its requests to be satisfied.
- The resource-allocation-graph algorithm is not applicable to a resourceallocation system with multiple instances of each resource type**.**

Banker's Algmorithm

- Banker's algorithm is also applicable to a resource allocation system with **multiple instances** of each resource type.
- It is is **less efficient** than the resource-allocation graph scheme.
- In this scheme when a new process enters the system, it must declare the maximum number of instances of each resource type that it may need. This nun1.ber may not exceed the total number of resources in the system. When a user requests a set of resources, the system must determine whether the allocation of these resources will leave the system in a safe state. If it will, the resources are allocated; otherwise, the process must wait until some other process releases enough resources.

Data Structure'smaintained for implementation of Banker's Algorithm: Let n is the number of the processes in the system. and m is the number of resource type.

Available. A vector of length m indicates the number of available resources of each type.

Max. An n x m matrix defines the maximum demand of each process.

Allocation. An 11 x m matrix defines the number of resources of each type currently allocated to each process

Need. An n x m matrix indicates the remaining resource need of each process. Need[i][j] = Max[i][j] - Allocation [i][j]

Safety Algorithm: It is used for finding out whether or not a systern is in a safe state.

1- Let Work and Finish be vectors of length m and n, respectively. Initialize Work= Available and Finish[i] =false for i = 0, 1, ... , n - 1.

2- Find an index i such that both

a. Finish[i] ==false

b. $Need_i \leq Work$

If no such i exists, go to step 4.

3- $Work = Work + Allocation_i$

Finish[i] = true Go to step 2.

Manavendra Pratap Singh-[M.Tech-IIIT Allahabad]

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

Resource-Request Algorithm: It is used for determining whether requests can be safely granted. Let Request; be the request vector for process P;. If Request; [j] == k, then process P; wants k instances of resource type Rj. When a request for resources is made by process P;, the following actions are taken-

1- If $Request_i \leq$ Need;, go to step 2. Otherwise, raise an error condition, since the process has exceeded its maximum claim.

2- If $Request_i \leq$ Available, go to step 3. Otherwise, P; must wait, since the resources are not available.

3- Have the system pretend to have allocated the requested resources to process P; by modifyil1.g the state as follows:

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; is allocated its resources. However, if the new state is unsafe, then P; must wait for Request;, and the old resource-allocation state is restored

Example: 1(a) Consider a system with five processes Po through P4 and three resource types A, B, and C. Resource type A has ten instances, resource type B has five instances, and resource type C has seven instances. Suppose that, at time T0, the following snapshot of the system has been taken:

	Allocation	**Max**
	ABC	ABC
P0	010	753
P1	200	322
P2	302	902
P3	211	222
P4	002	433

since available = Total – Total Allocation

= [10 5 7] – [725] = [325]

Need matrix is defined as Need = Max- allocation

	Allocation	Max	Need	Available
	ABC	ABC	ABC	ABC
P0	010	753	743	332
P1	200	322	122	
P2	302	902	600	
P3	211	222	011	
P4	002	433	431	
Total	725			

Select Process	Available/ Work=Available+ Allocation of selected process
P1	Available=[332]+[200]= [532]
P3	Available=[532]+[211]= [743]
P4	Available=[743]+[002]= [745]
P2	Available=[745]+[302]= [1047]
P0	Available=[1047]+[010]= [1057]

We can find a safe sequence **< Pl, P3, P4, P2, Po>** so system is in safe state.

1(b) process P1 requests one additional instance of resource type A and two instances of resource type C, so Request1 = (1,0,2). To decide whether this request can be immediately granted, we first check that $Request_1 \leq$ Available-that is, that (1,0,2) $\leq$ (3,3,2), which is true. We then pretend that this request has been fulfilled, and we arrive at the following new state.

	Allocation	Max	Need	Available
	ABC	ABC	ABC	ABC
P0	010	753	743	332-102=**230**
P1	200+102=**302**	322	122-102=**020**	
P2	302	902	600	
P3	211	222	011	
P4	002	433	431	
Total	725			

we execute our safety algorithm and find that the sequence **<P1, P3, P4, Po, P2>** satisfies the safety requirement. Hence, we can immediately grant the request of process P1.

Recovery From Deadlock:

a) **Process Termination:** To eliminate deadlocks by aborting a process, we use one of two methods. In both methods, the system reclaims all resources allocated to the terminated processes.
   I. **Abort all deadlocked processes**: This method clearly will break the deadlock cycle, but at great expense.
   II. **Abort one process at a time until the deadlock cycle is eliminated.** This method incurs considerable overhead, since after each process is aborted, a deadlock-detection algorithnc rnust be invoked to determine whether any processes are still deadlocked.

b) **Resource Preemption:** To eliminate deadlocks using resource preemption, we successively preempt some resources from processes and give these resources to other processes 1-m til the deadlock cycle is broken.

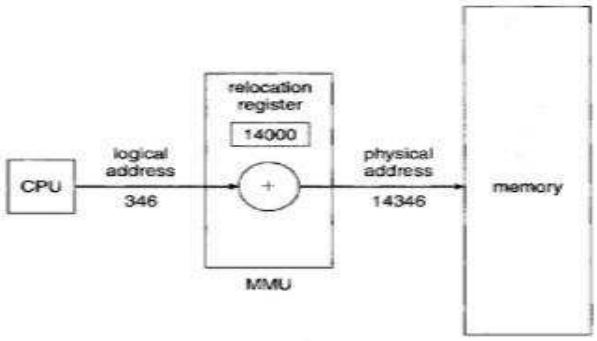
Unit-4

Memory Management Strategies

The memory management algorithms vary from a primitive bare-machine approach to paging and segmentation strategies. Each approach has its own advantages and disadvantages. Selection of a memory-management method for a specific system depends on many factors, especially on the hardware design of the system.

Logical versus Physical Address Space

- An address generated by the CPU is commonly referred to as a **Logical Address.**
- An address seen by the memory unit is commonly referred to as a **Physical Address.**
- The set of all logical addresses generated by a program is a logical address space.
- The set of all physical addresses corresponding to these logical addresses is a physical address space.
- The run-time mapping from virtual to physical addresses is done by a hardware device called Memory Management Unit (MMU).



Fragmentation

External Fragmentation: External fragmentation exists when there is enough total memory space to satisfy a request but the available spaces are not contiguous; storage is fragmented into a large number of small holes. This fragmentation problem can be severe. In the worst case, we could have a block of free (or wasted) memory between every two processes. If all these small pieces of memory were in one big free block instead, we might be able to run several more processes.

Internal Fragmentation: The memory allocated to a process may be slightly larger than the requested memory. The difference between these two numbers is internal memory that is internal to a partition.

Compaction: One solution to the problem of external fragmentation is Compaction. The goal is to shuffle the memory contents so as to place all free memory together in one large block. Compaction is not always possible. The simplest compaction algorithm is to move all processes toward one end of memory; all holes move in the other direction, producing one large hole of available memory but this scheme can be expensive. Another possible solution to the external-fragmentation problem is to permit the logical address space of the processes to be noncontiguous.

Contiguous memory allocation

- In contiguous memory allocation, each process is contained in a single contiguous section of memory.
- Accessing memory in contiguous memory allocation is faster.
- Managing contiguous memory allocation is easier.
- External fragmentation may occur in contiguous memory allocation.
- Contiguous memory allocation can be classified in following two categories:
    1. MFT (Multiprogramming with Fixed number of tasks).
    2. MVT (Multiprogramming with variable number of tasks).

MFT (Multiprogramming with Fixed number of tasks)

In this method memory is divided into fixed sized partitions. Each partition may contain exactly one process. Thus, the degree of multiprogramming is bound by the number of partitions. In this multiple partition method 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. MFT suffers from following problems—

- ✓ Internal fragmentation may occur in this method.
- ✓ External fragmentation may occur in this method.
- ✓ Deciding most appropriate value of Number of partitions and size of partition is difficult.

MVT (Multiprogramming with variable number of tasks)

In the scheme, the operating system keeps a table indicating which parts of memory are available and which are occupied. Initially, all memory is available for user processes and is considered one large block of available memory, a hole.

As processes enter the system, they are put into an input queue. The operating system takes into account the memory requirements of each process and the amount of available memory space in determining which processes are allocated memory. When a process is allocated space, it is loaded into memory, and it can then compete for CPU time. When a process terminates, it releases its memory which the operating system may then fill with another process from the input queue.

- ✓ This method solves the problem of internal fragmentation.
- ✓ External fragmentation may occur in this method.

Algorithms used in contiguous memory allocation techniques

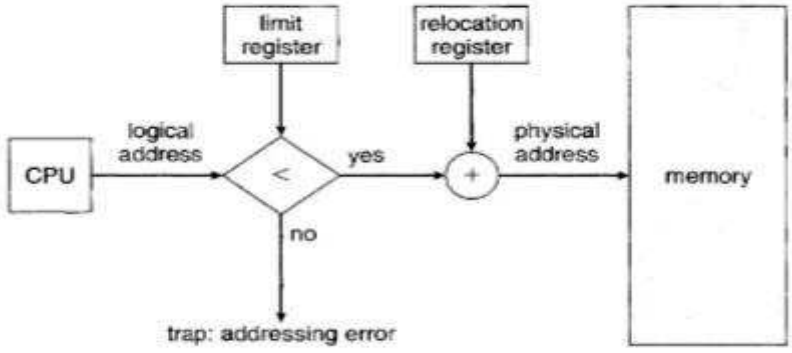
First fit: Allocate the first hole that is big enough. Searching can start either at the beginning of the set of holes or at the location where the previous first-fit search ended. We can stop searching as soon as we find a free hole that is large enough.

Best fit: Allocate the smallest hole that is big enough. We must search the entire list, unless the list is ordered by size. This strategy produces the smallest leftover hole.

Worst fit: Allocate the largest hole. Again, we must search the entire list, unless it is sorted by size. This strategy produces the largest leftover hole, which may be more useful than the smaller leftover hole from a best-fit approach.

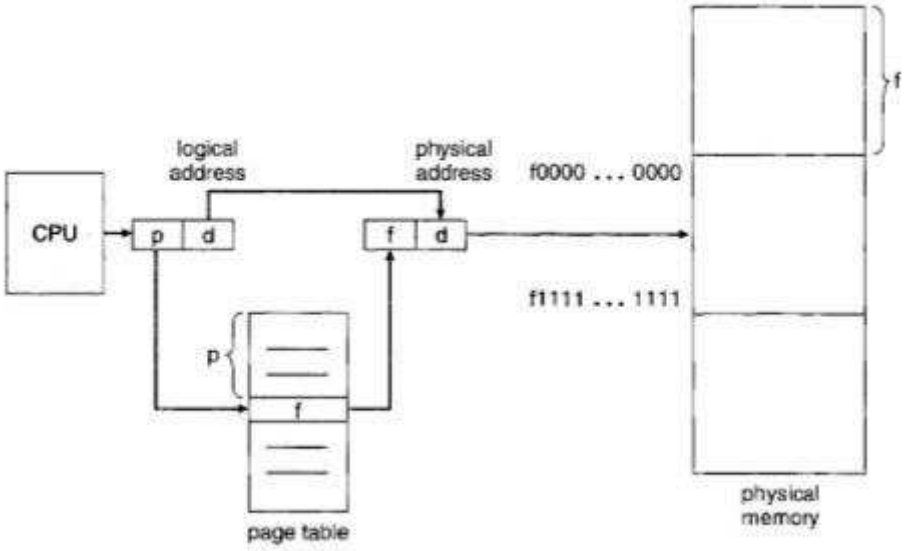
Memory Protection in contiguous memory allocation

- To make sure that each process has a separate memory space, we need the ability to determine the range of legal addresses that the process may access and to ensure that the process can access only these legal addresses. We can provide this protection by using two registers, usually a base and a limit or relocation.
- The base register holds the smallest legal physical memory address. The limit register specifies the size of the range. For example, if the base register holds 300 and the limit register is 100, then the program can legally access all addresses from 300 through 400.
- Protection of memory space is accomplished by having the CPU hardware compare every address generated in user mode with the registers. Any attempt by a program executing in user mode to access operating-system memory or other users' memory results in a trap to the operating system.



Paging

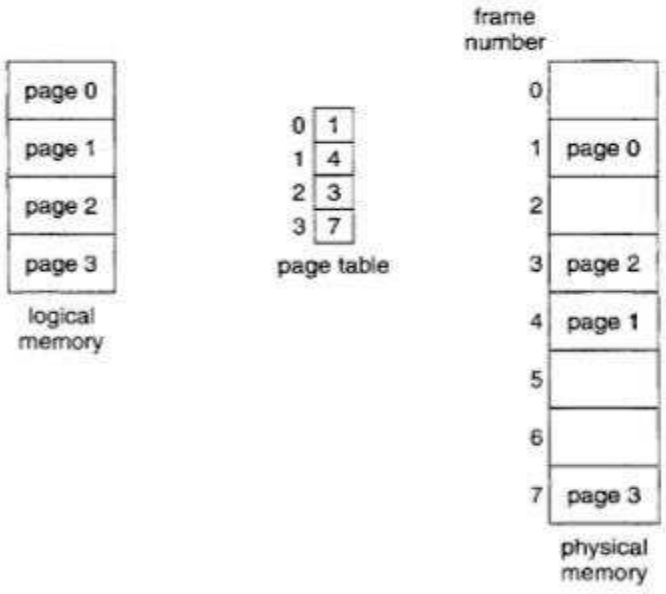
- Paging is a memory-management scheme that permits the physical address space a process to be non-contiguous.
- Paging avoids external fragmentation but not internal fragmentation.



Paging hardware.

- Paging involves breaking physical memory into fixed-sized blocks called frames and breaking logical memory into blocks of the same size called pages.
- When a process is to be executed, its pages are loaded into any available memory frames from their source
- Every address generated the CPU (Logical address) is divided into two parts: a page number p and a page offset d. The page number is used as an index into a page table. The page table contains the base address of each page in physical memory. This base address is combined with the page offset to define the physical memory address that is sent to the memory unit.
- The size of a page is typically a power of 2, varying between 512 bytes and 16 MB per page, depending on the computer architecture. The selection of a power of 2 as a page size makes the translation of a logical address into a page number and page offset particularly easy.

Example:



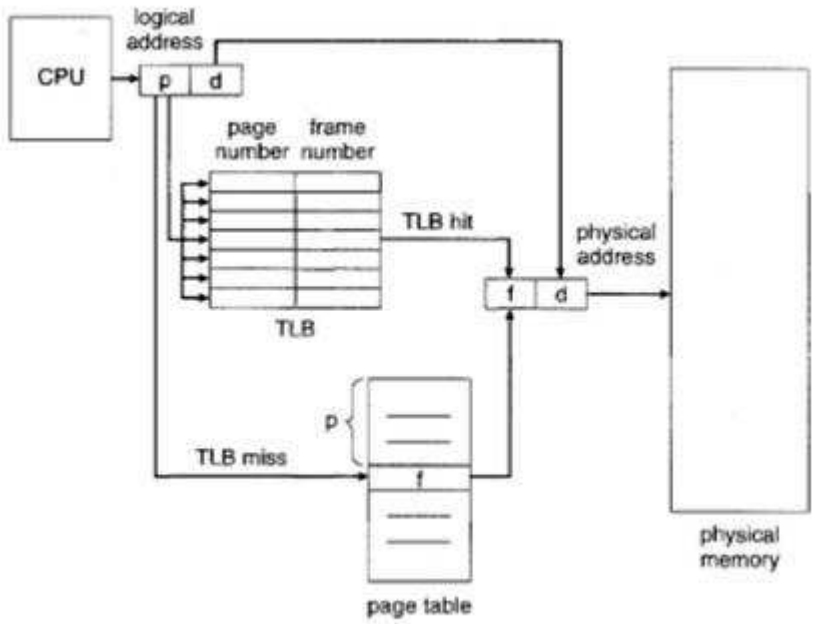
Paging model of logical and physical memory.

Page Table

- ✓ Page table is a data structure maintained by OS.
- ✓ In paging OS creates page table for every process separately.
- ✓ Page table contains frame number, corresponding to Page number.
- ✓ Page table resides in physical memory; it wastes large space in physical memory because generally its size is large.

Paging with TLB

- To increase the speed of memory access, TLB is used with page table.
- TLB is a special, small, fast, lookup hardware cache, called a translation look aside buffer.
- The TLB is used with page tables in the following way.
  - ✓ The TLB contains only a few of the page-table entries. When a logical address is generated by the CPU, its page number is presented to the TLB. If the page number is found (known as a TLB Hit), its frame number is immediately available and is used to access memory.
  - ✓ If the page number is not in the TLB (known as a TLB Miss), a memory reference to the page table must be made. When the frame number is obtained, we can use it to access memory.



Paging hardware with TLB.

- **Hit Ratio p = N_{Hit}/(N_{Hit} + N_{Miss})**
- **Effective Memory Access time = (N_{Hit} *(T_{TLB} + T_{PM}) + N_{Miss} *(T_{TLB} + 2*T_{PM}))/(N_{Hit} + N_{Miss})**

Where:-

N_{Miss} → Total number of Miss in TLB

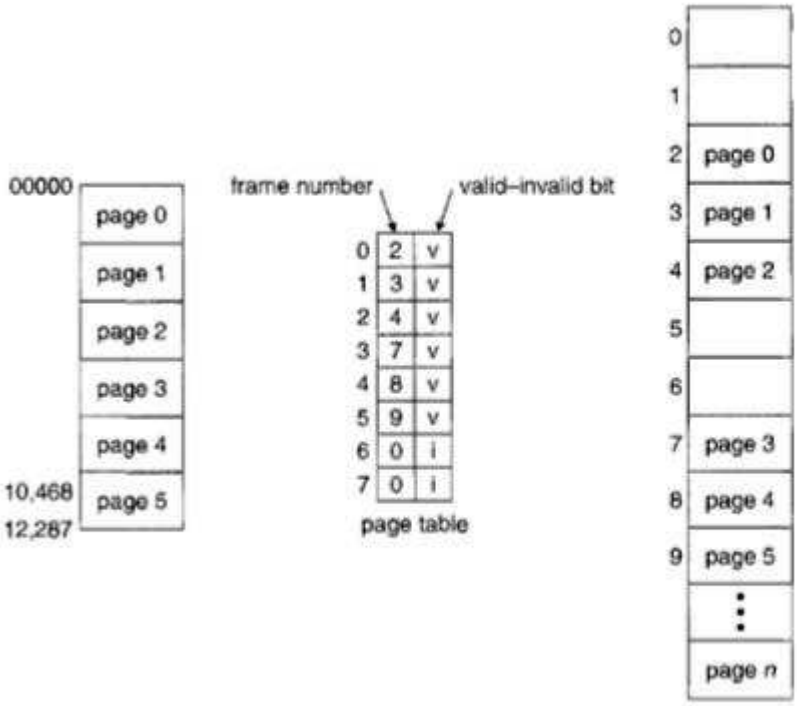
N_{Hit} → Number of Hit in TLB

T_{TLB} → access time of TLB

T_{PM} → access time of PM

Protection in Paging

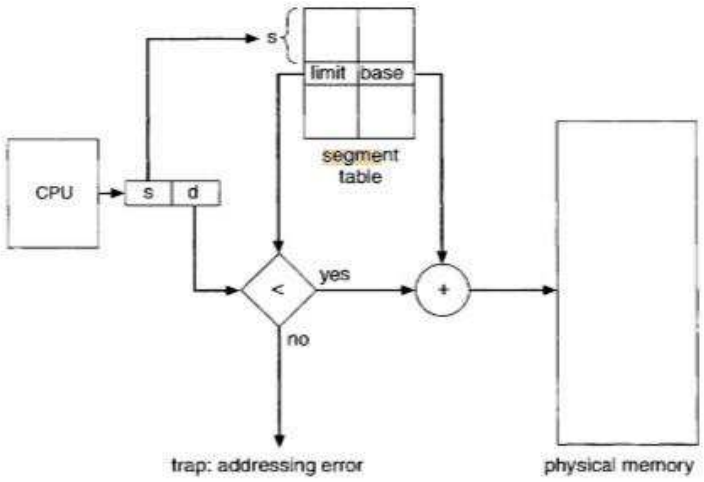
- Memory protection in a paged environment is accomplished by protection bits associated with each frame. Normally, these bits are kept in the page table.
- One bit can define a page to be read-write or read-only. Every reference to memory goes through the page table to find the correct frame number. At the same time that the physical address is being computed, the protection bits can be checked to verify that no writes are being made to a read-only page. An attempt to write to a read-only page causes a hardware trap to the operating system.
- One additional bit is generally attached to each entry in the page table called valid-invalid bit. When this bit is set to "valid," the associated page is in the process's logical address space and is thus a legal (or valid) page. When the bit is set to “invalid," the page is not in the process's logical address space. Illegal addresses are trapped by use of the valid -invalid bit. The operating system sets this bit for each page to allow or disallow access to the page.



Valid (v) or invalid (i) bit in a page table.

Segmentation

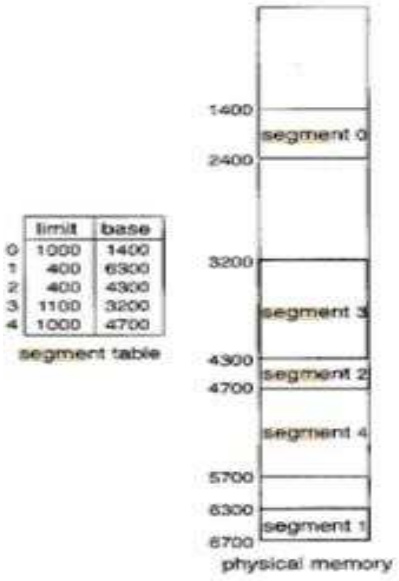
- Segmentation is a memory-management scheme that supports user view of memory.
- In segmentation logical address space is a collection of variable size segments. Each segment has a name and a length.
- Segmentation avoids internal fragmentation and external fragmentation both.
- Segment table takes lesser space in physical memory than page table.



- Each entry in the segment table has a segment base and a segment limit. The segment base contains the starting physical address where the segment resides in memory, and the segment limit specifies the length of the segment.

- In segmentation logical address consists of two parts: a segment number, s, and an offset into that segment, d. the segment number are used as an index to the segment table. The offset d of the logical address must be between 0 and the segment limit. If it is not, we trap to the operating system (logical addressing attempt beyond end of segment). When an offset is legal, it is added to the segment base to produce the address in physical memory of the desired byte. The segment table is thus essentially an array of base-limit register pairs.

  Example: segment 2 is 400 bytes long and begins at location 4300. Thus, a reference to byte 53 of segment 2 is mapped onto location 4300 +53= 4353. A reference to segment 3, byte 852, is mapped to 3200 (the base of segment 3) + 852 = 4052. A reference to byte 1222 of segment 0 would result in a trap to the operating system, as this segment is only 1,000 bytes long.

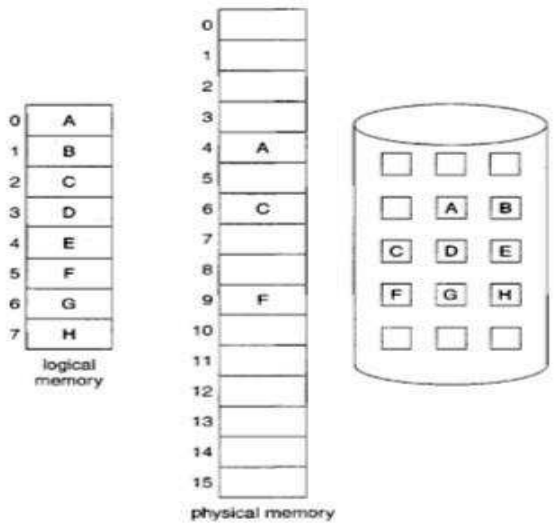


Virtual memory

- Virtual memory is a technique that allows the execution of processes that are not completely in memory. One major advantage of this scheme is that programs can be larger than physical memory.
- Virtual memory also allows processes to share files easily and to implement shared memory.
- Virtual memory is not easy to implement, however, and may substantially decrease performance if it is used carelessly.

Demand Paging

- In demand-paged virtual memory, pages are only loaded when they are demanded during program execution. Pages that are never accessed are thus never loaded into physical memory.
- This technique is commonly used in virtual memory systems.
- In demand paging lazy swapper is used. It never swaps a page into memory unless that page will be needed.
- Objective of Demand paging is to increase degree of multi-programming.



Performance of Demand Paging: Performance of demand paging is generally measured by effective access time. Let p be the probability of a page fault ($0 \leq p \leq 1$). We would expect p to be close to zero-that is, we would expect to have only a few page faults. The effective access time is then-

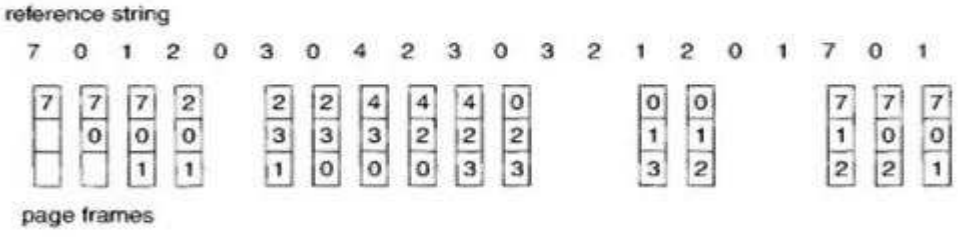
Effective access time= (1 - p) * ma + p * page fault time.

Page Replacement Policies

1. FIFO Page Replacement:

- The simplest page-replacement algorithm is a first-in, first-out (FIFO) algorithm. A FIFO replacement algorithm associates with each page the time when that page was brought into memory. When a page must be replaced, the oldest page is chosen.
- FIFO page-replacement algorithm is easy to understand and program.
- Belady's anomaly may occurs for some sequence like 1,2,3,4,1,2,5,1,2,3,4,5 number of faults for four frames (ten) is greater than the number of faults for three frames (nine).

Example:

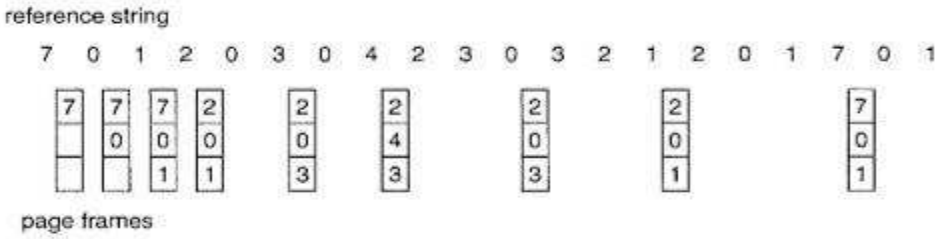


2. Optimal Page Replacement:

- It has the lowest page-fault rate of all algorithms and will never suffer from Belady's anomaly.
- It Replace the page that will not be used for the longest period of time.

- Use of this page-replacement algorithm guarantees the lowest possible page fault rate for a fixed number of frames.
- The optimal page-replacement algorithm is difficult to implement, because it requires future knowledge of the reference string.

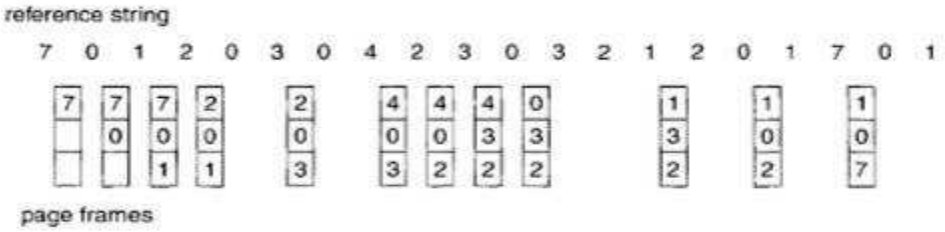
Example:



3. LRU Page Replacement:

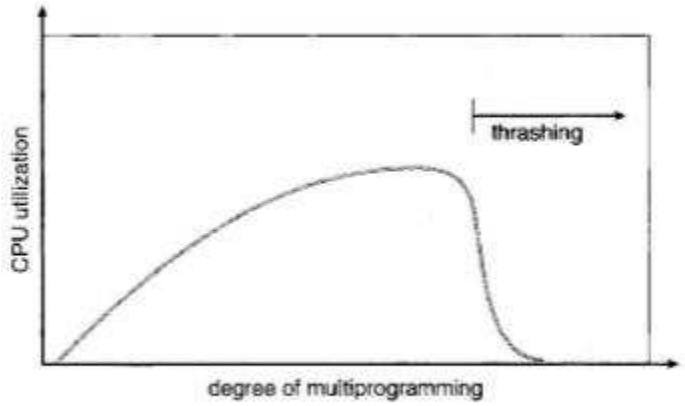
- In least recently algorithm we can replace the page that has not been used for the longest period of time.
- The major problem is how to implement LRU replacement. An LRU page-replacement algorithm may require substantial hardware assistance.

Example:



Thrashing

A process is thrashing if it spending more time in swap-in and swap-out than executing. Thrashing occurs when we increase degree of multiprogramming too much.



Thrashing.

As the degree of multiprogramming increases, CPU utilization also increases, although more slowly, until a maximum is reached. If the degree of multiprogramming is increased even further, thrashing sets in, and CPU utilization drops sharply. At this point, to increase CPU utilization and stop thrashing, we must decrease the degree of multiprogramming.

Locality of Reference

Locality of Reference refers to the tendency of the computer program to access instructions whose addresses are near one another. The property of locality of reference is mainly shown by loops and subroutine calls in a program.

- In case of loops in program control processing unit repeatedly refers to the set of instructions that constitute the loop.
- In case of subroutine calls, every time the set of instructions are fetched from memory.
- References to data items also get localized that means same data item is referenced again and again.

Temporal Locality – Temporal locality means current data or instruction that is being fetched may be needed soon. So we should store that data or instruction in the cache memory so that we can avoid again searching in main memory for the same data.

Spatial Locality – Spatial locality means instruction or data near to the current memory location that is being fetched, may be needed soon in the near future. This is slightly different from the temporal locality. Here we are talking about nearly located memory locations while in temporal locality we were talking about the actual memory location that was being fetched.

Bare Machine

In computer science, bare machine refers to a computer executing instructions directly on logic hardware without an intervening operating system. prior to the development of operating systems, sequential instructions were executed on the computer hardware directly using machine language without any

system software layer. This approach is termed the "bare machine" precursor to modern operating systems. Today it is mostly applicable to embedded systems and firmware Code runs faster on bare machine but creating software is more expensive for bare machines.

Resident Monitor

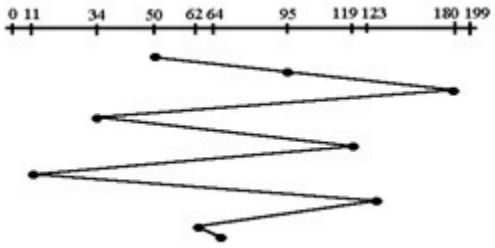
In computing, a resident monitor is a type of system software program that was used in many early computers from the 1950s to 1970s. It can be considered a precursor to the operating system. The name is derived from a program which is always present in the computer's memory thus being "resident". Because memory was very limited on these systems the resident monitor was often little more than a stub which would gain control at the end of a job and load a non-resident portion to perform required job cleanup and setup tasks.

On a general-use computer using punched card input, the resident monitor governed the machine before and after each job control card was executed, loaded and interpreted each control card, and acted as a job sequencer for batch processing operations. The functions that the resident monitor could perform were: clearing memory from the last used program (with the exception of itself), loading programs, searching for program data and maintaining standard IO routines in memory.

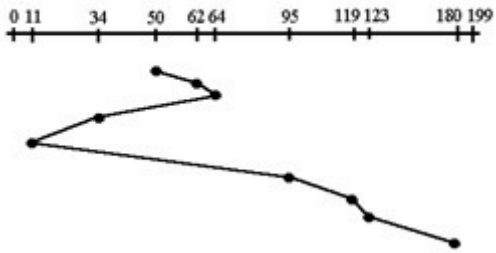
UNIT-5

Disk Scheduling Algorithms

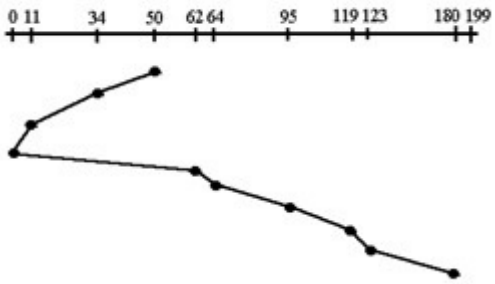
1. **FCFS:** FCFS is the simplest of all the Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue.
   Example: Given the following queue -- 95, 180, 34, 119, 11, 123, 62, 64 with the Read-write head initially at the track 50 and the tail track being at 199.



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

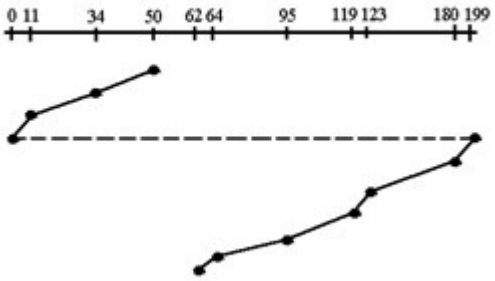


3. **SCAN:** In SCAN algorithm the disk arm moves into a particular direction and services the requests coming in its path and after reaching the end of disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works as an elevator and hence also known as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



4. **CSCAN**: In SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction. So, it may be possible that too many requests are waiting at the other end or there may be zero or few requests pending at the scanned area.These situations are avoided in *CSAN* algorithm in which the disk arm instead of reversing its direction goes to the other end of the disk and starts servicing the requests from there. So, the disk arm moves in a circular fashion and this algorithm is also similar to SCAN algorithm and hence it is known as C-SCAN (Circular SCAN).

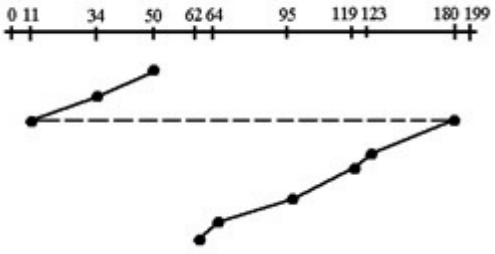
Example:



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

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

Example:



Kernel I/O Subsystem

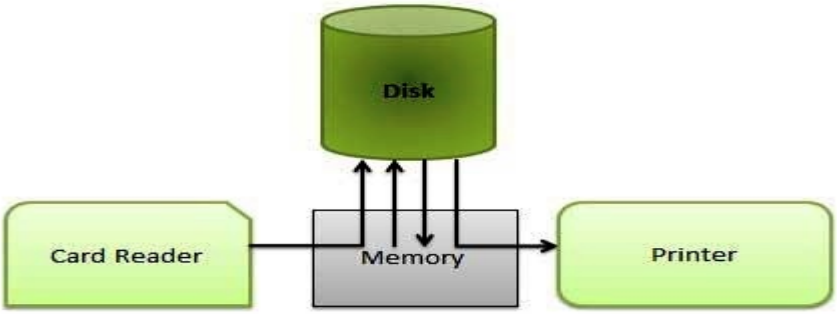
Kernel I/O Subsystem is responsible to provide many services related to I/O. Following are some of the services provided.

- **Scheduling** − Kernel schedules a set of I/O requests to determine a good order in which to execute them. When an application issues a blocking I/O system call, the request is placed on the queue for that device. The Kernel I/O scheduler rearranges the order of the queue to improve the overall system efficiency and the average response time experienced by the applications.
- **Buffering** − Kernel I/O Subsystem maintains a memory area known as **buffer** that stores data while they are transferred between two devices or between a device with an application operation. Buffering is done to cope with a speed mismatch between the producer and consumer of a data stream or to adapt between devices that have different data transfer sizes.
- **Caching** − Kernel maintains cache memory which is region of fast memory that holds copies of data. Access to the cached copy is more efficient than access to the original.
- **Spooling and Device Reservation** − A spool is a buffer that holds output for a device, such as a printer, that cannot accept interleaved data streams. The spooling system copies the queued spool files to the printer one at a time.
- **Error Handling** − an operating system that uses protected memory can guard against many kinds of hardware and application errors.

Spooling

- Spooling refers to simultaneous peripheral operations online.
- A spool is a buffer that holds output for a device, such as a printer, that cannot accept interleaved data streams.
- In spooling data is first onto the disk and then CPU interacts with disk via main memory.
- Spooling is capable of overlapping I/O operations for one job with CPU operations of other jobs.

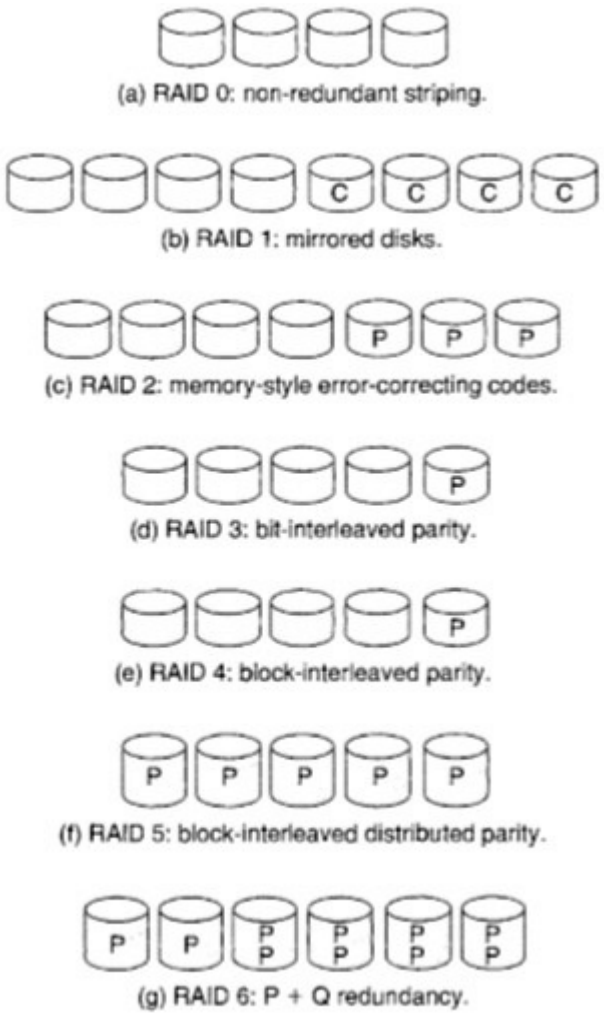
Example: using printer



RAID/ Redundant Array of Independent Disks

RAID is the way of combining several independent and relatively small disks into a single storage of a large size. The disks included into the array are called array members. RAID storage uses multiple disks in order to provide fault tolerance, to improve overall performance, and to increase storage capacity in a system. This is in contrast with older storage devices that used only a single disk drive to store data. The disks can be combined into the array in different ways which are known as **RAID levels**. Each of RAID levels has its own characteristics of:

Standard RAID Levels RAID devices use much different architecture, called levels, depending on the desired balance between performance and fault tolerance. RAID levels describe how data is distributed across the drives. Standard RAID levels include the following:



RAID levels.

Level 0: Striped disk array without fault tolerance

- Provides data striping (spreading out blocks of each file across multiple disk drives) but no redundancy. This improves performance but does not deliver fault tolerance. If one drive fails then all data in the array is lost.

Level 1: Mirroring and duplexing

- Provides disk mirroring. Level 1 provides twice the read transaction rate of single disks and the same write transaction rate as single disks.

Level 2: Error-correcting coding

- Not a typical implementation and rarely used, Level 2 stripes data at the bit level rather than the block level.

Level 3: Bit-interleaved parity

- Provides byte-level striping with a dedicated parity disk. Level 3, which cannot service simultaneous multiple requests, also is rarely used.

Level 4: Dedicated parity drive

- A commonly used implementation of RAID, Level 4 provides block-level striping (like Level 0) with a parity disk. If a data disk fails, the parity data is used to create a replacement disk. A disadvantage to Level 4 is that the parity disk can create write bottlenecks.

Level 5: Block interleaved distributed parity

- Provides data striping at the byte level and also stripe error correction information. This results in excellent performance and good fault tolerance. Level 5 is one of the most popular implementations of RAID.

Level 6: Independent data disks with double parity

- Provides block-level striping with parity data distributed across all disks.

Level 10: A stripe of mirrors

- Not one of the original RAID levels, multiple RAID 1 mirrors are created, and a RAID 0 stripe is created over these.

Problems with RAID: RAID does not always assure that data are available for the operating system and its users.

File Systems

File:A file is a collection of related information that is recorded on secondary storage. Or file is a collection of logically related entities. From user's perspective a file is the smallest allotment of logical secondary storage.

File Access Methods: The way that files are accessed and read into memory is determined by Access methods. Usually a single access method is supported by systems while there are OS's that support multiple access methods.

1. Sequential Access

- Data is accessed one record right after another is an order.
- Read command cause a pointer to be moved ahead by one.
- Write command allocate space for the record and move the pointer to the new End Of File.
- Such a method is reasonable for tape.

2. Direct Access

- This method is useful for disks.
- The file is viewed as a numbered sequence of blocks or records.
- There are no restrictions on which blocks are read/written; it can be done in any order.
- User now says "read n" rather than "read next".

3. **Indexed Sequential Access**

- It is built on top of Sequential access.
- It uses an Index to control the pointer while accessing files.

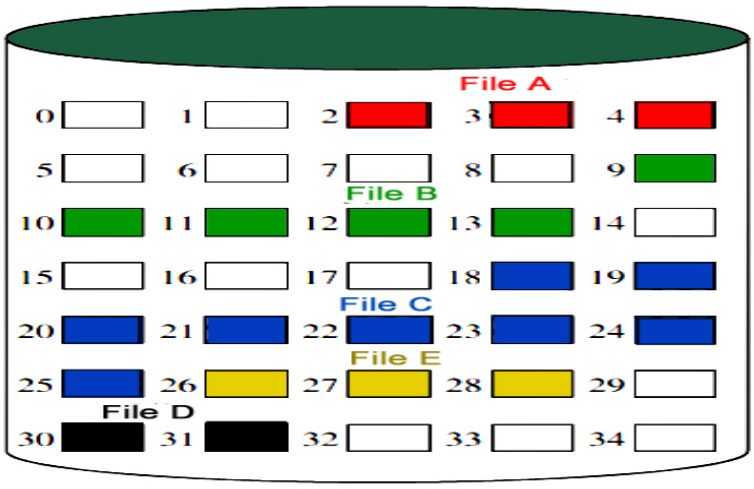
Directory: Information about files is maintained by Directories. A directory can contain multiple files. It can even have directories inside of them. In Windows we also call these directories as folders.

Following is the information maintained in a directory:

Name: The name visible to user.
Type: Type of the directory.
Location: Device and location on the device where the file header is located.
Size: Number of bytes/words/blocks in the file.
Position: Current next-read/next-write pointers.
Protection: Access control on read/write/execute/delete.
Usage: Time of creation, access, modification etc.
Mounting: When the root of one file system is "grafted" into the existing tree of another file system its called Mounting.

FILE Organization

1. Continuous Allocation: A single continuous set of blocks is allocated to a file at the time of file creation. Thus, this is a pre-allocation strategy, using variable size portions. The file allocation table needs just a single entry for each file, showing the starting block and the length of the file. This method is best from the point of view of the individual sequential file. Multiple blocks can be read in at a time to improve I/O performance for sequential processing. It is also easy to retrieve a single block. For example, if a file starts at block b, and the ith block of the file is wanted, its location on secondary storage is simply b+i-1.



File allocation table

File name	Start block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

Disadvantage

- External fragmentation may occur.
- Also, with pre-allocation, it is necessary to declare the size of the file at the time of creation.

2. Linked Allocation (Non-contiguous allocation): Allocation is on an individual block basis. Each block contains a pointer to the next block in the chain. Again the file table needs just a single entry for each file, showing the starting block and the length of the file. Although pre-allocation is possible, it is more common simply to allocate blocks as needed. Any free block can be added to the chain. The blocks need not be continuous. Increase in file size is always possible if free disk block is available. There is no external fragmentation because only one block at a time is needed but there can be internal fragmentation but it exists only in the last disk block of file.

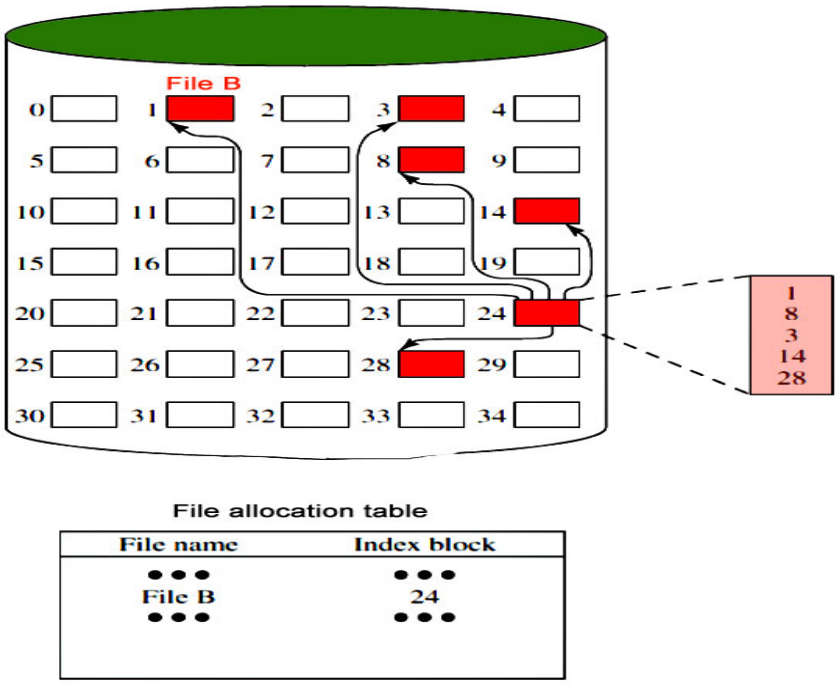
Disadvantage:

- Internal fragmentation exists in last disk block of file.
- There is an overhead of maintaining the pointer in every disk block.
- If the pointer of any disk block is lost, the file will be truncated.
- It supports only the sequential access of files.

3. Indexed Allocation:

It addresses many of the problems of contiguous and chained allocation. In this case, the file allocation table contains a separate one-level index for each file: The index has one entry for each block allocated to

the file. Allocation may be on the basis of fixed-size blocks or variable-sized blocks. Allocation by blocks eliminates external fragmentation, whereas allocation by variable-size blocks improves locality. This allocation technique supports both sequential and direct access to the file and thus is the most popular form of file allocation.



File sharing

File sharing is the public or private sharing of computer data or space in a network with various levels of access privilege. While files can easily be shared outside a network (for example, simply by handing or mailing someone your file on a diskette), the term *file sharing* almost always means sharing files in a network, even if in a small local area network. File sharing allows a number of people to use the same file or file by some combination of being able to read or view it, write to or modify it, copy it, or print it. Typically, a file sharing system has one or more administrators. Users may all have the same or may have different levels of access privilege. File sharing can also mean having an allocated amount of personal file storage in a common file system.

Protection and Security Methods

The different methods that may provide protect and securities for different computer systems are:

Authentication

This deals with identifying each user in the system and making sure they are who they claim to be. The operating system makes sure that all the users are authenticated before they access the system. The different ways to make sure that the users are authentic are:

- **Username/ Password**
  Each user has a distinct username and password combination and they need to enter it correctly before they can access the system.
- **User Key/ User Card**
  The users need to punch a card into the card slot or use the individual key on a keypad to access the system.
- **User Attribute Identification**

Different user attribute identifications that can be used are fingerprint, eye retina etc. These are unique for each user and are compared with the existing samples in the database. The user can only access the system if there is a match.

One Time Password

These passwords provide a lot of security for authentication purposes. A one time password can be generated exclusively for a login every time a user wants to enter the system. It cannot be used more than once. The various ways a one time password can be implemented are:

- **Random Numbers**
  The system can ask for numbers that correspond to alphabets that are pre arranged. This combination can be changed each time a login is required.
- **Secret Key**
  A hardware device can create a secret key related to the user id for login. This key can change each time.

File system Implementation

Numerous on-disk and in-memory configurations and structures are being used for implementing a file system. These structures differ based on the operating system and the file system but applying following general principles.

- A **boot control block** usually contains the information required by the system for booting an operating system from that volume. When the disks do not contain any operating system, this block can be treated as empty. This is typically the first chunk of a volume.
- A **volume control block** holds volume or the partition details, such as the number of blocks in the partition, size of the blocks or chunks, free-block count along with free-block pointers.
- A **directory structure per file system** is required for organizing the files.
- The **FCB** contains many details regarding any file which includes file permissions, ownership; the size of file and location of data blocks.