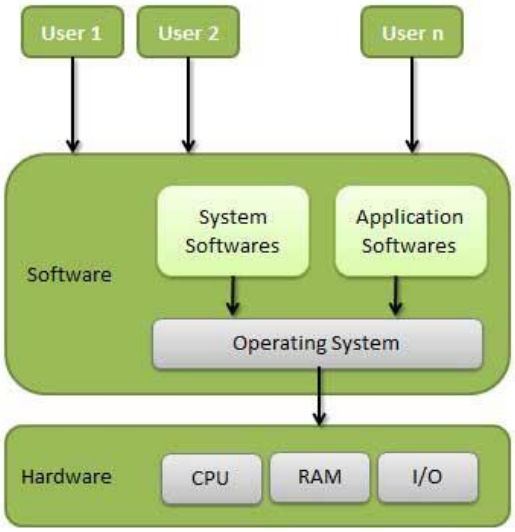
Operating Systems

OS- an operating system is the interface between the user and the Hardware. When computer boots up Operating System is the first program that loads. It also provides protection and security.

Kernel-. Its interface between application and Hardware. It is the first program to load when operating system loads. It convert user command into machine language. Its main purpose is memory management, process management.

CPU-CPU means central processing unit. its consist several transister.it is brain of computer. CPU responsible for receiving data input, processing data, and providing data Output.



Types of Operating Systems

Batch operating system

Each user prepares his job on device like punch cards and submits it to the computer operator. Similar jobs are batched together. OS assign job to the CPU, only after the execution of previous job completes.

The problems with Batch Systems are as follows –

- Lack of interaction between the user and the job.
- CPU is often idle, because the speed of the I/O devices is slower than the CPU. ex: Payroll System, Bank Statements

Multiprogramming Batch Systems

To overcome the problem of underutilization of CPU and main memory, the multiprogramming was introduced. The OS selects one of the processes and assign to the CPU. Whenever the executing process needs to wait for any other operation (like I/O), the OS selects another process from the job queue and assign to the CPU. This way, the CPU is never kept idle.

Multitasking OS

Multitasking OS combination of Multiprogramming OS and CPU scheduling.

Suppose you are writing a paragraph, meanwhile you wanted to play a song. You can simply do it by selecting the song with a music player software and do both. Now think about OS level operations. It was previously allocating memory and processing power for your typing and visualization. Now OS have to allocate separate memory and processing power for media player and OS have to do both task simultaneously. Here 2 application sharing common resource (CPU). This is called multi-tasking in OS.

Ex. Windows, android, MAC

Time Sharing OS

It supports 2 simple concepts: multi-tasking and the allows more than one user logged in and using its computing resources at the same time. ex: Multics, Unix.

Real Time OS

Real time operating system is an operating system which is specifically used in real time scenario like missile, aerospace, automotive etc. The main responsibility of RTOS is to complete the task at specified time it may be in Nano second, millions second.
Ex.FreeRTOS
Real time application: airbag in car
Suppose a person is driving a car on a highway at a speed of 70 miles per hour. Now, somehow the car meets with an accident. Fortunately the airbag deployed at the right time and saved the life of the driver. What would have happened if the airbag would have deployed a few seconds late? Yes, we would have lost a life. Here, RTOS makes that airbag deploy at the right time

Distributed operating System

Distributed operating system combines group of computers and gives the illusion as a single computer to the end user.
Whenever server traffic grow one can upgrade the hardware and software configuration of the server to handle it which is known as the vertical scaling. The vertical scaling is good but one cannot scale it after some point of time. Even best hardware and software cannot provide the better service to big traffic.
The following are the different application of the distributed system.

- Global positioning System
- World Wide Web

In the World Wide Web the data or application distributed on the several number of the heterogeneous computer system. But it appear as a single system to the end user. These are referred as loosely coupled systems or distributed systems. ex: LOCUS is a discontinued OS.

Multiprogramming, Multitasking, Multiprocessing and Multithreading

- **Multiprogramming** – Multiprogramming is nothing but a situation where a computer running more than one program at a time (like running Excel and Firefox simultaneously).Multiprogramming need not be multi-processing or multi-tasking.
- **Multiprocessing** – A computer using more than one CPU at a time.
- **Multitasking** – Multiple Task can share a common resource (like 1 CPU).ex. Running browser and music player in single CPU system.
- **Multithreading** is an extension of multitasking. Multi-threading allows a single process to have multiple code segments (i.e., threads) running concurrently .e.g. VLC media player, where one thread is used for opening the VLC media player, one thread for playing a particular song and another thread for adding new songs to the playlist.

Process Management

Program vs Process vs Thread

A program is set of instruction stored in Disk.

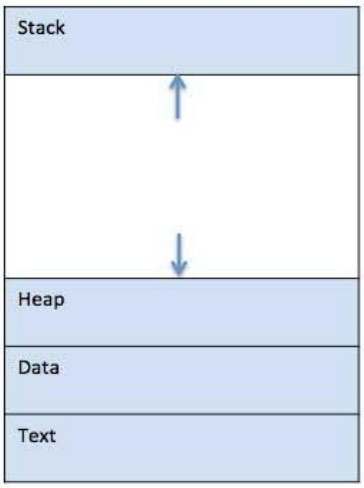
A process is a program in execution and heavy weight operation. A single program can create many processes when run multiple times, for example when we open a .exe or binary file multiple times, many processes are created.

Thread is a lightweight operation. Because process has own memory space but thread uses the memory of process and Threads have same properties as of the process. There are many threads are possible in a single process. Processes don't share memory with other processes. Threads share memory with other threads of the same process

Process	Thread
Processes are heavyweight operations	Threads are lighter weight operations
Each process has its own memory space	Threads use the memory of the process they belong to
Processes don't share memory with other processes	Threads share memory with other threads of the same process
Inter-process communication is slow because processes have different memory addresses	Inter-thread communication can be faster than inter-process communication because threads of the same process have the same memory address
Context switching between processes is slow.	Context switching between threads of the same process is fast.

What does a process look like in Memory?

- **Text Section:** contain compiled program code
- **Data Section**: Contains the global variable and static variable.
- **The Heap** is used for the dynamic memory allocation, and is managed via calls to new, delete, malloc, free, etc.
- **The Stack** is used for local variables and function call.



Process Control Block (PCB):

It is a data structure .Process Control Block available for each process, containing all the information about the process.

Attributes of PCB:

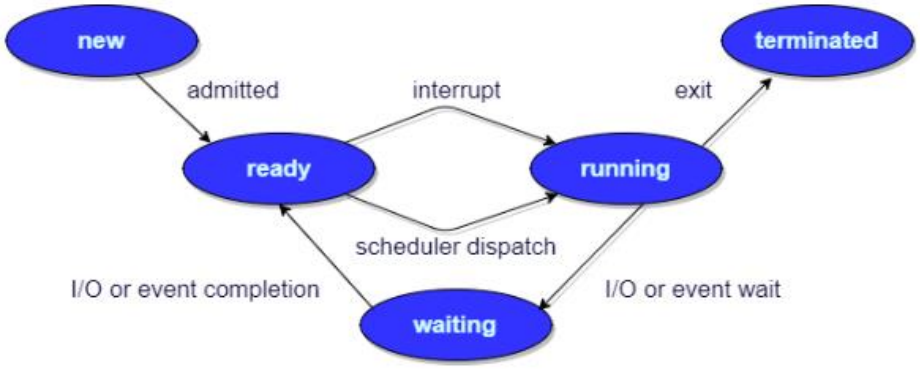
1. *Process Id:* A unique identifier assigned by operating system
2. *Process State:* Can be ready, running, .etc
3. Program Counter: holds the address of the next instruction to be executed for that process.
4. *CPU scheduling information*: For example Priority scheduling ,FIFO scheduling.

5. *Memory Management information*: For example, page tables or segment tables.
6. *Priority:* priority of the process, if it have.

Different Process States

Processes in the operating system can be in one of the following states:

- NEW- when The process is created it resides in NEW state
- READY- once the process ready to get the CPU it resides in READY state.
- RUNNING- Process is in executed state .
- WAITING- The process is waiting for (I/O) resources
- TERMINATED- The process has finished execution.



Threads:

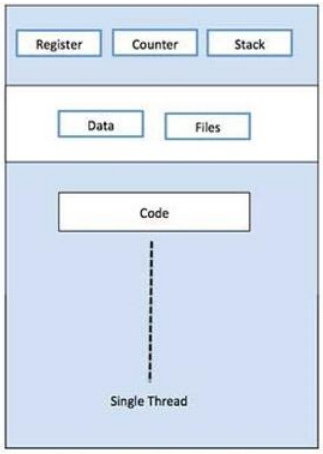
Thread is a lightweight operation. Because process has own memory space but threads uses the memory of process where they belong to. Processes don't share memory with other processes. Threads share memory with other threads of the same process.
Threads have same properties of the process. There are many threads are possible in a single process. Tell example using VLC Media player.
A new thread, or a child process of a given process, can be introduced by using the fork() system call
Each thread have their own register, counter, stack.
There are two types of threads:

- User threads

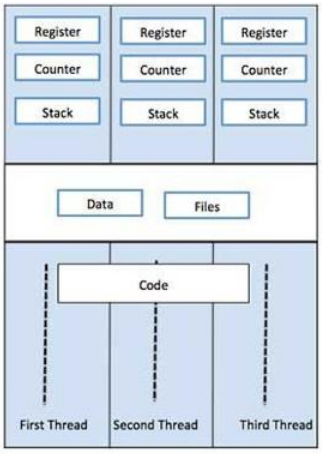
Example: Java thread, POSIX threads

- Kernel threads

Example : Window Solaris.



Single Process P with single thread



Single Process P with three threads

USER LEVEL THREAD	KERNEL LEVEL THREAD
User threads are implemented by users.	kernel threads are implemented by OS.
OS doesn't recognize user level threads.	Kernel threads are recognized by OS.
Implementation of User threads is easy.	Implementation of Kernel thread is complicated.
If one user level thread performs blocking operation then entire process will be blocked.	If one kernel thread performs blocking operation then another thread can continue execution.

Advantages of Thread

- Threads minimize the context switching time.
- Use of threads provides concurrency (parallelism) within a process.

Why Multithreading?

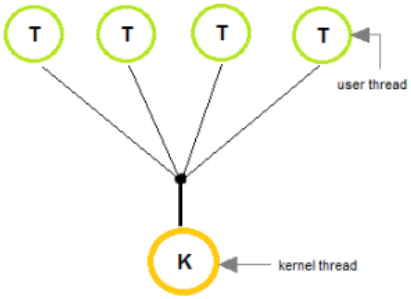
- Thread is lightweight operation. **Multithreading** is an extension of multitasking. Multi-threading allows a single process to have multiple code segments (i.e., threads) running

concurrently . So, we can achieve this parallelism by dividing a process into multiple code segments (i.e.: threads).

- For example, in a browser, multiple tabs can be different threads. MS word uses multiple threads, one thread to format the text, other thread to keyboard inputs etc.
- VLC media player, where one thread is used for opening the VLC media player, one thread for playing a particular song and another thread for adding new songs to the playlist.

Types of multithreading:

- Many to many relationship- many user-level threads are all mapped to a many kernel thread
- Many to one relationship-many user-level threads are all mapped to a single kernel thread



- One to one relationship-one user-level thread mapped to a single kernel thread

Benefits of Multithreading:-

Responsiveness
For example, if you're typing a document in Word, one thread responding to your keyboard, one thread checking your spelling and one thread checking your grammar.

Process Scheduling

The act of determining which process is in the ready state, and which process should be moved to the running state is known as Process Scheduling. The main goal is to keep the CPU busy all the time (not ideal) and to deliver minimum response time for all process.

Scheduling fell into one of the two general categories:

- Non Pre-emptive Scheduling: The executing process is not interleaved before completing its execution.
- Pre-emptive Scheduling: The executing process can interleaved before completing its execution due to other high priority process.

Objectives of Process Scheduling Algorithm:

- Max CPU utilization (Keep CPU as busy as possible)
- Max throughput (Number of completed processes at particular time limit)
- Min waiting time (Time for which a process waits in ready queue)

Process Scheduling: Below are different times with respect to a process.

1. **Arrival Time** – Time at which the process arrives in the ready queue.
2. **Completion Time** – Time at which process completes its execution.
3. **Burst Time** – Time required by a process for CPU execution.
4. **Turn Around Time** – Time Difference between completion time and arrival time.
   ```
   Turn Around Time = Completion Time - Arrival Time
   ```
5. **Waiting Time (WT)** – Time Difference between turn around time and burst time.
   ```
   Waiting Time = Turn Around Time - Burst Time
   ```

Process Scheduling Queues

The Operating System maintains the following queues for process scheduling

- **Job queue** – This queue keeps all the processes in the system.
- **Ready queue** – This queue keeps all the processes which are ready to get the CPU. A new process is always put in this queue.
- **Device queues** – This queue keeps all the processes which are blocked due to unavailability of an I/O device put into this queue.

Long-Term	Short-Term	Medium-Term
Long term is also known as a job scheduler	Short term is also known as CPU scheduler	Medium-term is also called swapping scheduler.

There are three types of process scheduler.

1. **Long Term or job scheduler:** It brings the new process to the 'Ready State'. It controls *Degree of Multi-programming*, i.e., number of process present in ready state at any point of time.
2. **Short term or CPU scheduler:** It brings the process from ready state to running state. Note: Short-term scheduler only brings the process to running state. It doesn't responsible for loading the process.

Dispatcher is responsible for loading the process . Context switching is done by dispatcher only.

3. **Medium-term scheduler:** It is responsible for suspending and resuming the process. It mainly does swapping (moving processes from main memory to disk and vice versa).

Degree of multiprogramming

The maximum number of process that can reside in the ready state.

e.g., if degree of programming = 100 means 100 processes can reside in the ready state at maximum.

Context Switching

The process of saving the context of one process and loading the context of other process is known as Context Switching. In simple term, it is like moving the process from ready state to running state and moving the process from running state to waiting state.

When does Context switching happen?

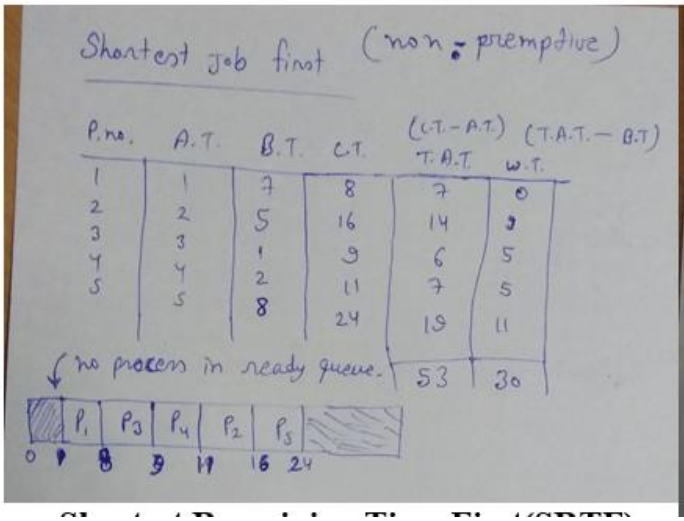
- When a high priority process comes to ready state ,and running process has low priority then context switch will occur.
- When Interrupt Occurs
- When Preemptive CPU scheduling used

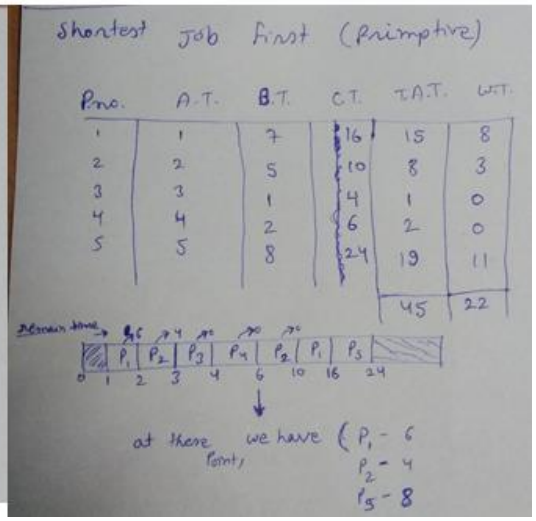
Why do we need scheduling?

In a Uniprogramming system like MS-DOS, most of time wasted for getting I/O resources and CPU is free during this time. In multi programming systems, If one process wait for I/O resources then another process can use CPU .

Different Scheduling Algorithms

1. First Come First Serve (FCFS): scheduling happen according to arrival times of processes.
2. Shortest Job First(SJF): scheduling happen according to shortest burst time of processes.

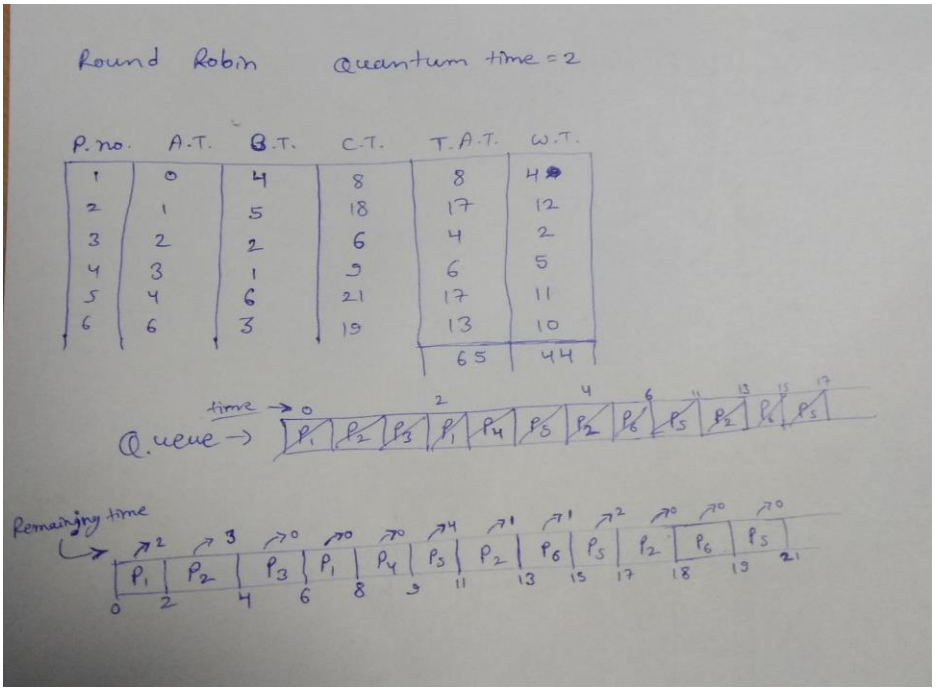




3.shortest remaining time first(srtf):

preemptive mode of SJF algorithm. Here, the process are schedule according to shortest remaining time.

4.Round Robin Scheduling: Each process is assigned a fixed time in cyclic way. The fixed time is called quantum time.

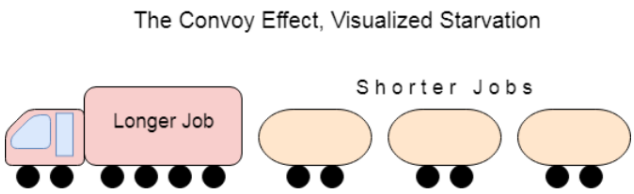


5.Priority Based scheduling (Non Preemptive): scheduling happened according to priorities of the processes. i.e., Highest priority process is schedule first. If priorities of two processes match, then scheduling is according to the arrival time.

Some useful facts about Scheduling Algorithms:

- FCFS can cause long waiting times, especially when the first job takes too much CPU time.it leads to convey effect.
- Both SJF and Shortest Remaining time first algorithms may cause starvation. Consider a situation when long process is there in ready queue and shorter processes keep coming.
- If time quantum for Round Robin scheduling is very large, then it behaves same as FCFS scheduling.
- Priority Based scheduling (Non Preemptive): In this scheduling, processes are scheduled according to their priorities, i.e., highest priority process is schedule first. If priorities of two processes match, then schedule according to arrival time. Priority scheduling may cause to starvation.

Convoy Effect in Operating Systems



Convoy Effect is a state associated with the First Come First Serve (FCFS) algorithm. Here, Operating System slows down due to few slow processes.

FCFS algorithm is non-preemptive in nature. So, once CPU has been allocated to a process, other processes can get CPU only after the current process has finished. This property of FCFS scheduling leads to the situation called Convoy Effect.

Ex: Just imagine the truck convey stuck under the bridge. Due to that other vehicles are waiting indefinitely.

Starvation

Starvation can occur Priority scheduling and shortest job first scheduling and shortest remaining time first.

In priority scheduling,

Starvation occurs when a low priority program is requesting for a system resource, but are not able to execute because a higher priority program is utilizing that resource for an extended period.

We can think of a scenario in which only one process P0 is having very low-priority (for example 10) and other process and upcoming process having high priority (more than 100).Here, P0 may wait indefinitely to get CPU.This is leads to starvation.

In SJF and SRTF,

Consider a situation when long process is there in ready queue and shorter processes keep coming.

In starvation resources are continuously utilized by high priority processes. Problem of starvation can be resolved using Aging. In Aging priority of long waiting processes is gradually increased.

Aging

Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time. For example, if priority range from 0(low) to 127(high), we could increase the priority of a waiting process by 1 Every 5 minutes. In aging the process with priority of 10 would not take more than 32 hours to achieve high priority and getting the resource.

Process Synchronization

When a shared resources used by more than 1 process at a time leads to inconsistency (not accurate) of shared resources. Process synchronization is a mechanism to avoid the clashes. Getting and releasing resources are two important things in process synchronization.

Let's consider, railway track which is a shared resource and trains as a different process. If all the trains using same track at a same point of time there is a possibility of clashes which leads to inconsistency of shared resource. Here signal used to maintain process synchronization (i.e: avoid clashes).

If process synchronization not done properly leads to deadlock, inconsistency(not correct) of shared data.

There are 2 software approach available for process synchronization

1. Semaphore
2. Mutex

- Semaphore is a signaling approach.
- Mutex is a lock based approach.

Semaphore

Mutex

Semaphore vs Mutex:

- Semaphore is signalling approach. Mutex is lock based approach
- In semaphore ,wait() and signal() function used to update the semaphore variable when the process is acquiring a resources and releasing resources. There is no special function available for update the mutex object.
- Semaphore can be released by any process. in mutex, if one process acquire resource ,that process only have rights to lock /unlock the mutex object
- In semaphore, multiple process can access finite number of resources at a time.in mutex, only one process allowed to access single resource at a time.
- two types of semaphore available binary(0,1) and counting(more than 0).no types in mutex

When to use mutex and semaphore:

- semaphore-when many resource are available then semaphore preferred
- **mutex- if there is only one resource are available then mutex preferred**

Binary semaphore vs mutex:

- **A mutex can be released only by the process which had acquired it.**
- A binary semaphore can be released by any process.

Note: mutual exclusion-only one at time (A resource can shared to only one process at a time)

Real life example:

Mutex:

- A mutex can be released only by the process which had acquired it.
- mutex- if there is only one resource are available then mutex preferred

Let consider the situation we have only one dressing room. Here, the mutex value is the count of keys. It is set to 1 at beginning and the count value is decremented if a person get into dressing room. If one person occupied the room. i.e.: There is no free key left, the mutex value is 0. When that particular person leaves from the dressing room the mutex value increased by 1 and Given key to next person in the queue. Here, that particular person only update the mutex value and only one dressing room we have.

Semaphore:

- A semaphore can be released by any process.
- semaphore-when many resource are available then semaphore preferred

Let consider the situation we have four dressing room with identical locks and keys. The semaphore value is the count of keys. Initially it is set to 4 at beginning (all four rooms are free), then the count value is decremented if people get into the room. If all rooms are full, ie. There are no free keys left, the semaphore count is 0. Now, when one person leaves the room, semaphore is increased to 1 , and given key to the next person in the queue. Here any person can update the semaphore value and many dressing room we have.

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

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

Race condition:

A *Race condition* is a special condition that may occur inside a critical section. A *critical section* is a section of code which is executed by multiple threads .If the final result of critical section depending on the sequence of multiple threads then the critical section contain a race condition.

```
if (x == 5) // The "Check"
{
   y = x * 2; // The "Act"

   // If another thread changed x in between "if (x == 5)" and "y = x
   // y will not be equal to 10.
}
```

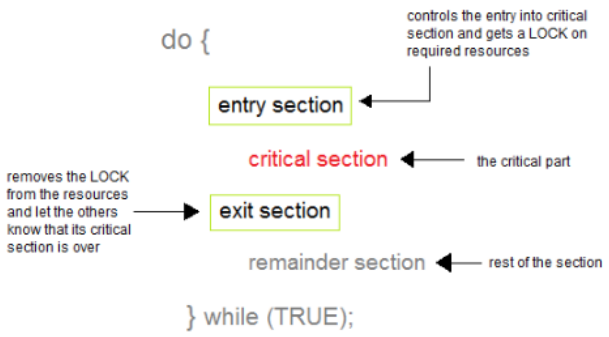
In order to prevent race conditions, we put a lock around the shared data to ensure only one thread can access the data at a time.

```
// Obtain lock for x
if (x == 5)
{
   y = x * 2; // Now, nothing can change x until the lock is released.
              // Therefore y = 10
}
// release lock for x
```

Critical Sections

Running more than one thread inside the same application does not cause problems. The problems arise when multiple threads access the same resources (database, memory of variable) at a same time.

A *Critical section* is a section of code which is executed by multiple threads .The final result of critical section dependent on sequence of threads if threads not synchronized properly. Because, Critical section contains the shared variable. If shared variable is accessed by more than 1 thread at a time the problem raised. To overcome this problem only one thread need to execute critical section at a time and all the remaining thread need to wait to complete the execution of the previous thread inside the critical section.



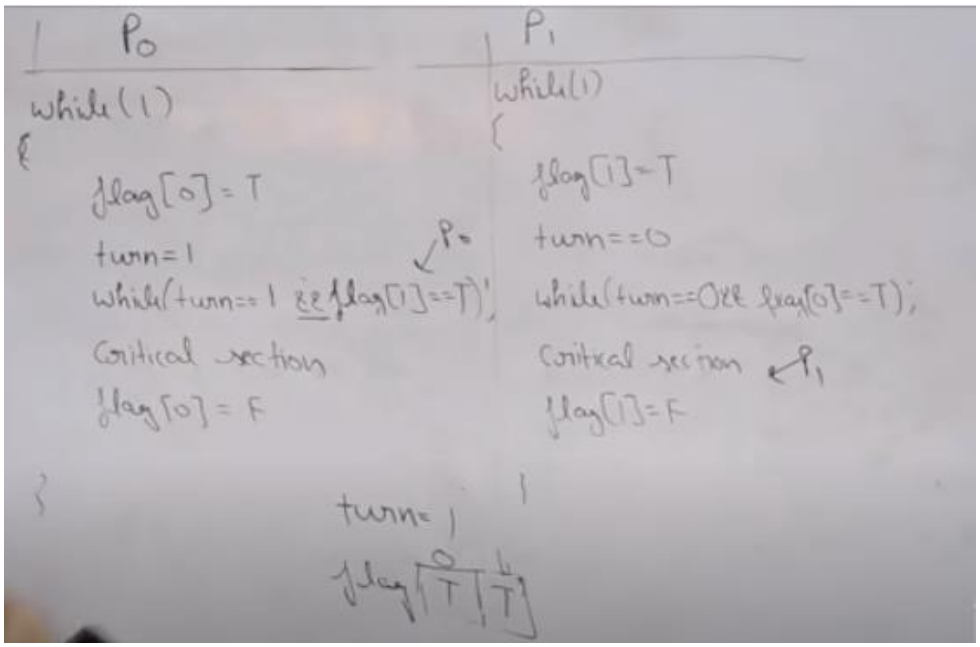
Solution to Critical Section Problem

A solution to the critical section problem must satisfy the following three conditions:

1. Mutual Exclusion: Mutual Exclusion means that if one thread is executing inside critical section then the other thread must not enter in the critical section.
2. Progress: Progress means that if one thread doesn't need to execute critical section then it should not stop other thread to get into the critical section.
3. Bounded Waiting: We should be able to predict the waiting time for every thread to get into the critical section.

Peterson's Solution

Peterson's Solution is a software based solution to the critical section problem.
Peterson's Solution preserves all three conditions.
In Peterson's solution, we have two shared variables:



1. boolean flag[i] :Initialized to FALSE, initially no one is interested in entering the critical section.
2. int turn : The process whose turn is to enter the critical section.

In above code flag[i] represent state of i^{th} process. initially , no process try to enter critical section. So, flag[0]=false, flag[1]=false. Now p0 ,try to enter critical section it can enter.its still in critical section. During that p1 trying to enter into critical section. But it can not enter. Because, while loop execute indefinitely until flag[0]=false;

Link: https://www.youtube.com/watch?v=XAsAAJSotA4

Disadvantages of Peterson's Solution

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

Semaphore Solution

Semaphore is signalling approach for process synchronization. It is another solution to the critical section problem.

A semaphore uses two function, wait and signal for process synchronization.

The wait function decrements the value of its argument S, if it is positive. If S is negative or zero, then no operation is performed and waiting for s >0.

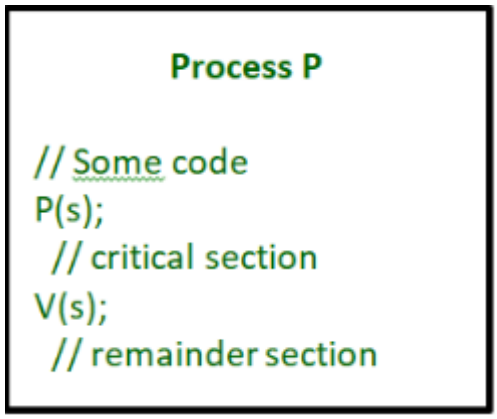
```
wait(S){
   while (S<=0);
   S--;
}
```

The signal function increments the value of its argument S.

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

Semaphore is always initialized to one.

A critical section is surrounded by both function to implement process synchronization. See below image. Critical section of a Process is in between P(wait function) and V (signal function).



Now, let us see how it implements mutual exclusion. Let there be two processes P1 and P2 and a semaphore s is initialized as 1. Now if suppose P1 enters in its critical section then the value of semaphore s becomes 0. Now if P2 wants to enter its critical section then it will wait until s > 0, this can only happen when P1 finishes its critical section and calls V operation on semaphore s. This way mutual exclusion is achieved.

The description above is for binary semaphore which can take only two values 0 and 1 and ensure the mutual exclusion. There is one other type of semaphore called counting semaphore which can take values greater than one.

Now suppose there is a resource whose number of instance is 4. Now we initialize S = 4 and rest is same as for binary semaphore. Whenever process wants that resource it calls P or wait function and when it is done it calls V or signal function. If the value of S becomes zero then a process has to wait until S becomes positive. For example, Suppose there are 4 process P1, P2, P3, P4 and they all call wait operation on S(initialized with 4). If another process P5 wants the resource then it should wait until one of the four processes calls signal function and value of semaphore becomes positive.

Semaphores are of two types:

1. Binary Semaphore – This is also known as mutex lock but not actual mutex. It can have only two values – 0 and 1. Its value is initialized to 1. It is used to implement the solution of critical section problem.
2. Counting Semaphore – It can take any value .

Limitations

1. One of the biggest limitation of semaphore is priority inversion.
2. With improper use of resource may block the process indefinitely. Such situation is called deadlock.

Classical Problems of Synchronization

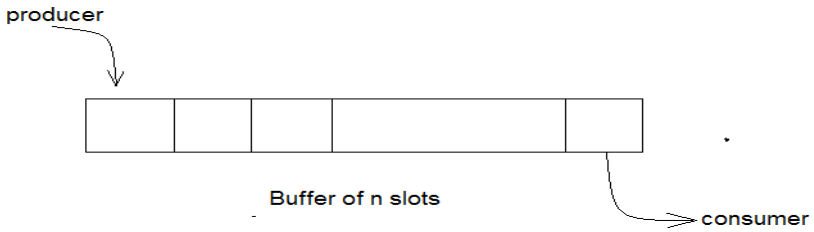
We will discuss the following three problems:

1. Bounded Buffer (Producer-Consumer) Problem
2. The Readers Writers Problem
3. Dining Philosophers Problem

Bounded Buffer Problem (producer consumer problem)

What is the Problem Statement?

We have a buffer of fixed size. The job of the Producer is to generate the data, put it into the buffer, and again start generating data. While the job of the Consumer is to consume the data from the buffer.



What's the problem here?

- If the buffer is full, then the producer shouldn't be allowed to put any data into the buffer.
- If the buffer is empty, then the consumer shouldn't be allowed to take any data from the buffer.
- The producer and consumer should not access the buffer at the same time.

If either producer or consumer not obey the above rule the problem will occur.

What's the solution?

The above three problems can be solved with the help of semaphores

- *Mutex*- a binary semaphore which is used to acquire and release the lock.
- *Empty*-a counting semaphore : This semaphore variable is used to define the empty space in the buffer. Initially, it is set to the whole space of the buffer i.e. "n" because the buffer is initially empty.
- *Full*-a counting semaphore : This semaphore variable is used to define the space that is filled by the producer. Initially, it is set to "0" because there is no space filled by the producer initially.

At any instant, the current value of empty represents the number of empty slots in the buffer and full represents the number of occupied slots in the buffer.

The Producer Operation

```
do
{
    wait (empty);
    wait (mutex) ; // aquire lock

    // perform the insert operation in a slot

    signal(mutex); // release lock
    signal(full);
} while (TRUE);
```

The wait operation decrements the value of its argument S, if it is positive. If S is negative or zero, then no operation is performed.

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

The signal operation increments the value of its argument S.

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

The Consumer Operation

```
do
{
   wait (full);
   wait (mutex) ; // aquire lock

   // perform the remove operation in a slot

   signal(mutex); // release lock
   signal(empty);
} while (TRUE);
```

Readers Writer Problem

The Problem Statement

There is a shared resource and accessed by two types of processes namely reader and writer. If two readers access the shared resource at the same time there is no problem. However if two writers or a reader and writer access the shared resource at the same time, there may be problems.

Solution:

The solution for above problem is if a writer is writing data to the resource then no other process can access the resource. *and If a writer wants to write data into the resource, it must wait until there are no readers or writers currently accessing that resource*.

– wait() : decrements the semaphore value.

– signal() : increments the semaphore value.

In the Below code, mutex and wrt are semaphores that are initialized to 1. Also, rc is a variable that is initialized to 0. wrt is common to the reader and writer process code.

The variable rc denotes the number of readers accessing the object.

Reader Process code:

```
do {

wait (mutex);
rc ++;
if (rc == 1)
wait (wrt);
signal(mutex);
.
. READ THE OBJECT
.
wait(mutex);
rc --;
if (rc == 0)
signal (wrt);
signal(mutex);
} while(true);
```

As soon as rc becomes 1, writer cannot access the object anymore. After the read operation is done, rc is decremented. When rc becomes 0, writer can access the object now.

Writer process code:

```
do {

wait(wrt);
.
. WRITE INTO THE OBJECT
.
signal(wrt);
} while(true);
```

If a writer wants to access the object, wait operation is performed on wrt. After that no other writer can access the object. When a writer is done writing into the object, signal operation is performed on wrt.

The wait operation decrements the value of its argument S, if it is positive. If S is negative or zero, then no operation is performed.

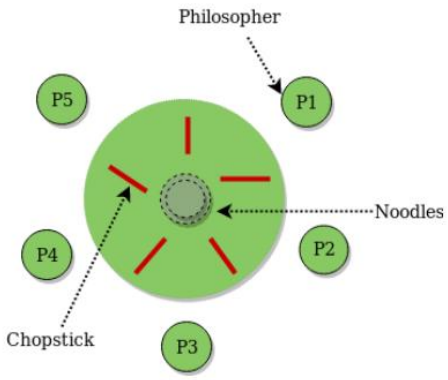
```
wait(S){
   while (S<=0);
   S--;
}
```

The signal operation increments the value of its argument S.

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

Dining Philosophers Problem

In dining philosophes problem multiple resources need to allocate to multiple processes.



What is the Problem Statement?
Consider there are five philosophers sitting around a circular dining table. The dining table has five chopsticks and a bowl of rice in the middle as shown in the below figure.

At any instant, a philosopher is either eating or thinking. When a philosopher wants to eat, he uses two chopsticks - one from their left and one from their right. When a philosopher wants to think, he keeps down both chopsticks at their original place. When a philosopher wants to eat the rice, he need to get both chopstick from his right

and left hand side. After eating, he puts both the chopsticks down.

But if all five philosophers are hungry simultaneously, and each of them pickup one chopstick, then a deadlock situation occurs because they will be waiting for another chopstick infinitely.
The possible solutions for this are:

- A philosopher must be allowed to pick up the chopsticks only if both the left and right chopsticks are available.
- Allow only four philosophers to sit at the table. That way, if all the four philosophers pick up four chopsticks, there will be one chopstick left on the table. So, one philosopher can start eating by using that remaining chopstick and once finished eating he can pass that remaining chopstick to another philosopher. In this way, deadlocks can be avoided.

The structure of the chopstick is shown below −

```
semaphore chopstick [5];
```

Initially the elements of the chopstick are initialized to 1 as the chopsticks are on the table and not picked up by a philosopher.

The structure of a random philosopher i is given as follows −

```
do {
   wait( chopstick[i] );
   wait( chopstick[ (i+1) % 5] );
   . .
   . EATING THE RICE
   .
   signal( chopstick[i] );
   signal( chopstick[ (i+1) % 5] );
   .
   . THINKING
   .
} while(1);
```

In the above structure, first wait operation is performed on chopstick[i] and chopstick[(i+1) % 5]. This means that the philosopher i has picked up the chopsticks on his sides. Then the eating function is performed.

After that, signal operation is performed on chopstick[i] and chopstick[(i+1) % 5]. This means that the philosopher i has eaten and put down the chopsticks on his sides. Then the philosopher goes back to thinking.

Under the normal mode of operation, a process may utilize a resource in only the following sequence:

1. Request- The process requests the resource. If the request cannot be granted immediately (for example, if the resource is being used by another process), then the requesting process must wait until it can acquire the resource.

2. Use- The process can operate on the resource (for example, if the resource is a printer, the process can print on the printer).

3. Release- The process releases the resource.

IPC

Inter process communication is a mechanism which allows processes to communicate each other. A process can be two types:

- Independent process
- Co-operating process

An independent process not affected by execution of other process while co-operating process affected by execution of other process. IPC coming to the picture among the co-operating process. 2 types of IPC possible

- shared memory
- message passing

Shared memory:

In shared memory ,process A share the resource to common shared space.Process B get that resource from shared space.

Ex: Producer consumer problem (bounded buffered):Here the buffer area is common shared space to both producer and consumer

Message passing:

In message passing, processes communicate with each other without using shared memory.

If two processes p1 and p2 want to communicate with each other, they proceed as follows:

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

Start exchanging messages using basic primitives.

We need at least two primitives:

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

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

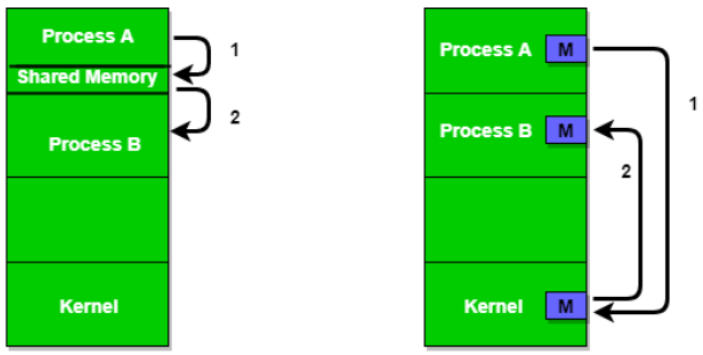


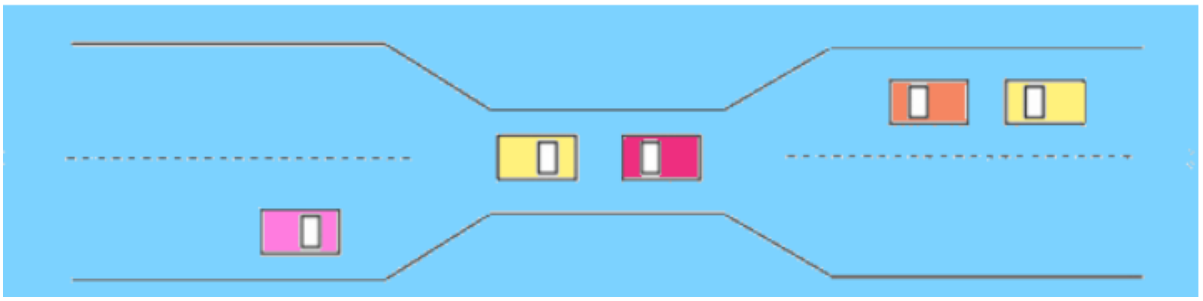
Figure 1 - Shared Memory and Message Passing

Deadlock:

A situation where a set of processes are blocked because each process is holding a resource and waiting for another resource which is acquired by some other process from the set. For example, Process A is allocated Resource B as it is requesting Resource A. In the same way, Process B is allocated Resource A, and it is requesting Resource B. This creates a circular wait loop.

Ex:

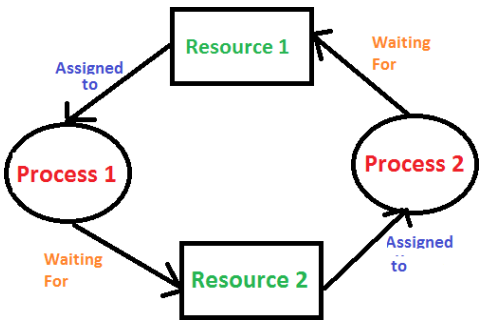
- You can't get the job without having the (professional) experience and you can't get the experience without having a job
- One way Bridge traffic. Consider bridge as resource. If two vehicle comes in opposite direction at a same time dead lock will occur. it can be resolved if one car back up (resource preemption).using starvation(priority) ,we will back up one vehicle from that bridge.
- If two person call each other at a same time will cause the dead lock. Both will get busy state as response from network provider.
- Exam pen and paper example. Person A has pen and person B has paper. Here, person A require paper to write the exam and person B require pen to write the exam .this situation met the Deadlock.



Dead lock condition:

Deadlock can arise if following four conditions hold simultaneously (Necessary Conditions):

1. *Mutual Exclusion:* Resource is used by Only one process at a time
2. Hold and Wait – A process is holding at least one resource and waiting for resources.
3. No Preemption – A process cannot interleaved before completing its execution.
4. Circular Wait – A set of processes are waiting for each other in circular form.



Methods for handling deadlock

There are three ways to handle deadlock

1. Deadlock prevention: The idea is to not let the system into deadlock state.
2. Deadlock detection and recovery : Let deadlock occur, then do preemption to handle it once occurred.
3. Ignore the Dead lock: If deadlock is very rare, then let it happen and reboot the system. This is the approach that both Windows and UNIX take.

Deadlock prevention

if we break one of the condition:-

1. Mutual Exclusion:

Mutual section from the resource point of view is the fact that a resource can never be used by more than one process simultaneously which is fair enough but that is the main reason behind the deadlock. If a resource could have been used by more than one process at the same time then the process would have never been waiting for any resource.

2. Hold and Wait

- Allocate all required resources to the process before start of its execution, this way hold and wait condition is eliminated but it will lead to low device utilization.
- For example, if a process requires printer at a later time and we have allocated printer before the start of its execution printer will remained blocked till it has completed its execution.

3. No Preemption

Preempt resources from process when resources required by other high priority process.

4. Circular Wait

To violate circular wait, we can assign a priority number to each of the resource. A process can request for the resources only in increasing order of numbering.
This ensures that not a single process can request a resource which is being utilized by some other process and no cycle will be formed.

For Example, if P1 process is allocated R5 resources, now next time if P1 ask for R4, R3 lesser than R5 such request will not be granted, only request for resources more than R5 will be granted.

Banker's Algorithm in Operating System

The banker's algorithm is a resource allocation and deadlock avoidance algorithm

Why Banker's algorithm is named so?
Banker's algorithm is named because it is used in banking system to check whether loan can be sanctioned to a person or not. Suppose there are n number of account holders in a bank and the total sum of their money is S. If a person applies for a loan then the bank first subtracts the loan amount from the total money(X) that bank has and if the

remaining amount is greater than S then only the loan is sanctioned. because if all the account holders comes to withdraw their money, the bank will be in problem state. The bank would try to be in safe state always.

Deadlock vs Starvation

S.NO	DEADLOCK	STARVATION
1.	All processes keep waiting for each other to complete and none get executed	High priority processes keep executing and low priority processes are blocked
2.	Resources are blocked by the processes	Resources are continuously utilized by high priority processes
3.	Necessary conditions Mutual Exclusion, Hold and Wait, No preemption, Circular Wait	Priorities are assigned to the processes
4.	Also known as Circular wait	Also know as lived lock
5.	It can be prevented by avoiding the necessary conditions for deadlock	It can be prevented by Aging

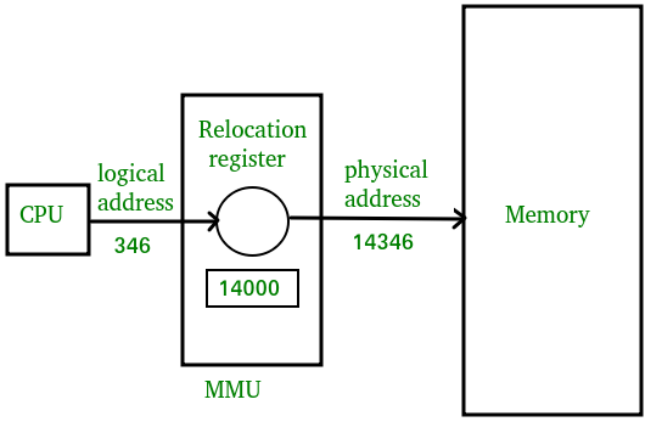
Memory Management

Memory Management:

Memory management is the mechanism of an operating system which handles swapping the processes between main memory and disk . It decides which process will get memory at what time. These techniques allow the memory to be shared among multiple processes.

Logical vs physical address space:

Logical Address is generated by CPU while a program is running. The logical address does not exist physically therefore it is also known as Virtual Address. CPU use this address to access the physical address.P hysical Address points to a physical location of required data in a memory. MMU used to map the logical address into physical address.



- Logical Address or Virtual Address : An address generated by the CPU
- Physical Address: An address actually available on memory unit

MMU (Memory Management Unit):

The hardware device called Memory-Management Unit is used for mapping logical address to its corresponding physical address.

Main memory usually has two partitions −

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

Swapping:

Swapping is a method used to swap lower-priority process from main memory to secondary storage (disk) and make that memory available to other higher-priority processes. In later, the swapping method swaps back the lower-priority process from the secondary storage to main memory.

Swapping the process can affect the performance of the process but it enables the multiple process can running in parallel.

Linking and Loading

Linking:

Object codes generated by the assembler. Linking combines object code to generate the executable module

Loading:

Moving the executable module from secondary memory to the main memory is called loading.

Dynamic linking:

a) When one program is dependent on some other program. In such a case, rather than loading all the dependent programs, CPU loads dependent program when its required. This mechanism is known as Dynamic Linking.
b) Dynamic linking refers to the linking that is done during run-time.

Loading:

STATIC	DYNAMIC
Loading the entire program into the main memory before start of the program execution is called as static loading.	Loading the program into the main memory on demand is called as dynamic loading.
Inefficient utilization of memory because whether it is required or not required, entire program is brought into the main memory.	Efficient utilization of memory.
Program execution will be faster.	Program execution will be slower.
If the static loading is used then accordingly static linking is applied.	If the dynamic loading is used then accordingly dynamic linking is applied.

Memory Management Techniques:

- *Single Partition Allocation Schemes*
  The memory is divided into two parts. One part is kept to be used by the OS and the other is kept to be used by the users.
- *Multiple Partition Schemes*
  Fixed Partition – The memory is divided into fixed size partitions.
  Variable Partition – The memory is divided into variable sized partitions.

Variable partition allocation schemes:

- **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**. 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

- Both the first-fit and best-fit strategies for memory allocation suffer from external fragmentation.
- As processes are loaded and removed from memory, the free memory space is broken into little pieces.

Note:

- Best fit does not necessarily give the best results for memory allocation.

Fragmentation

During processes are loaded and removed from memory, the free memory space is broken into little pieces. It happens after sometimes when no other process allocated to that memory space. So, that memory space remains unused. This problem is known as Fragmentation(vetridam).

Fragmentation is of two types –

External fragmentation

Total memory space is enough to hold a process in it, but it is not contiguous, so it cannot be used. This situation is called as external fragmentation.

Internal fragmentation

If bigger Memory block assigned to smaller process then remaining portion of that bigger memory block is left. If it cannot be utilized by some other process then it is called as internal fragmentation.

Suppose we have 2 process and we use FCFS scheduling and first fit memory allocation technique.

10	40	50

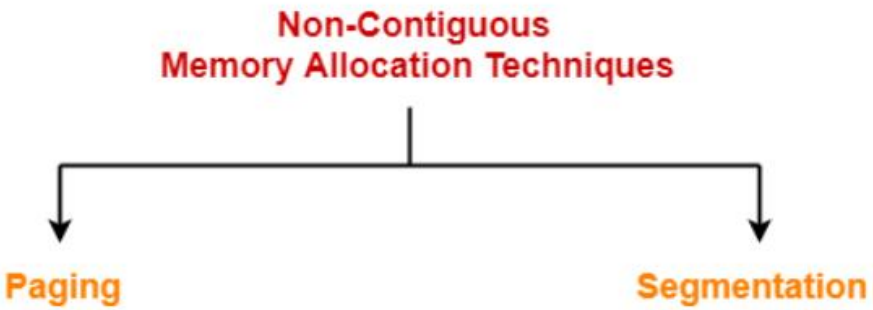
Let us consider the situation,According to First fit technique p1(35kb size) stored in second block 40kb but process need only 35kb. There is 5kb of memory lost is situation called internal fragmentation.

Let us consider the another situation , according to first fit technique p1(40kb size) stored in second block 40kb.if another process p2(60kb) arrived, we not able to store it in the main memory because the available memory is not contiguous.

Non-contiguous memory allocation technique:

- Non-contiguous memory allocation is a memory allocation technique.
- It stores a single process in a non-contiguous fashion.

There are two popular techniques used for non-contiguous memory allocation-



Paging with Example

Paging is a Noncontiguous memory allocation scheme. In paging, main memory and process in secondary memory are divided into equal fixed size partitions. The partitions of main memory are called as frames. The partitions of process in secondary memory are called as pages. Here, Page size is equal to the frame size. One page of the process is to be stored in one of the frames of the memory.

Advantage:

- No external Fragmentation. Because, paging provides the flexibility of storing the process at the different places.
- Swapping is easy between equal-sized pages and page frames.

Disadvantages of Paging

- May cause Internal fragmentation
- Page tables consume additional memory.

Example

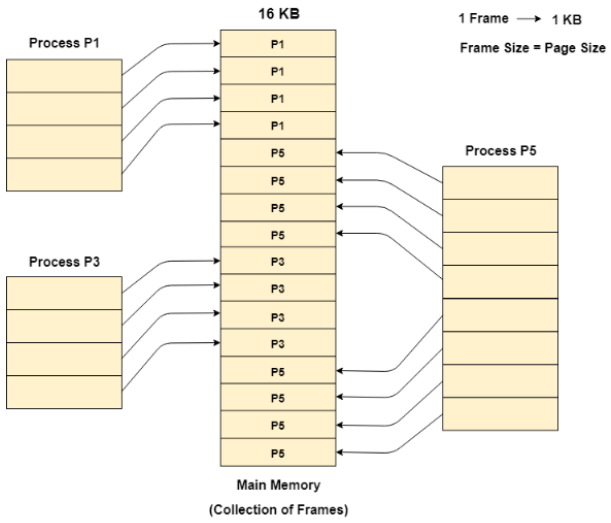
Let us consider the main memory size 16 Kb and Frame size is 1 KB therefore the main memory will be divided into the collection of 16 frames of 1 KB each.

There are 4 processes in the system that is P1, P2, P3 and P4 of 4 KB each. Each process is divided into pages of 1 KB each so that one page can be stored in one frame.

Initially, all the frames are empty therefore pages of the processes will get stored in the contiguous way.

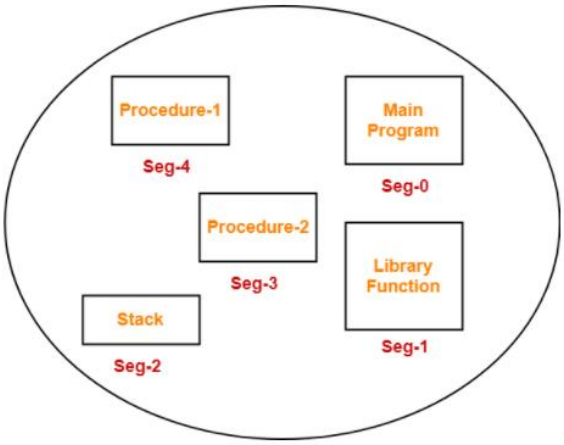
Let us consider that, P2 and P4 are moved to waiting state after some time. Now, 8 frames become empty and therefore other pages can be loaded in that empty place. Now The process P5 of size 8 KB (8 pages) is waiting inside the ready queue.

Given the fact that, we have 8 non-contiguous frames available in the memory and paging provides the flexibility of storing the process at the different places. Therefore, we can load the pages of process P5 in the place of P2 and P4.



Segmentation

Like Paging, Segmentation is another non-contiguous memory allocation technique. In segmentation, process is not divided blindly into fixed size pages. The process divided into several modules with variable size. Each module is known as segment .The main memory is divided into the several parts and size is dependent upon module. Segmentation gives user's view of the process which paging does not give.



Advantages of Segmentation –

- No Internal fragmentation.
- Segment Table consumes less space in comparison to Page table in paging.

Disadvantage of Segmentation

- As processes are loaded and removed from the memory, the free memory space is broken into little pieces, causing External fragmentation.

Note:

- CPU always generates a logical address. A physical address is needed to access the main memory.

In paging,

CPU generates a logical address consisting of two parts: Page Number, Page Offset

- Page Number specifies the particular page which CPU wants to read.
- Page Offset specifies the particular word on that page which CPU wants to read.

By using page table logical address convert to physical address.

In segmentation,

CPU generates a logical address consisting of two parts: segment Number, segment Offset

- Segment Number specifies the particular segment which CPU wants to read.
- Segment Offset specifies the specific word on that segment which CPU wants to read.

By using segment table logical address convert to physical address.

Paging	Segmentation
Main memory is divided into fixed size which is called frame	Main memory is divided into variable size which is called module
Paging divides program into fixed size which is called as pages.	Segmentation divides program into variable size which is called as segment.
It may lead to internal fragmentation.	It may lead to external fragmentation.
There is no external fragmentation	No internal fragmentation
Non-Contiguous memory allocation	Non-Contiguous memory allocation.
The paging technique is faster then segmentation for memory access.	Segmentation is slower than paging method.
Paging is closer to Operating System	Segmentation is closer to User
Logical address is divided into page number and page offset	Logical address is divided into segment number and segment offset
Page table is used to maintain the page information.	Segment Table maintains the segment information

Segmentation with Paging:-

Pure segmentation is not very popular and not being used in many of the operating systems. However, Segmentation can be combined with Paging to get the best features out of both the techniques.

In Segmented Paging, the main memory is divided into variable size segments which are further divided into fixed size pages.

- Pages are smaller than segments.
- Each Segment has a page table which means every program has multiple page tables.

Virtual Memory

We come across the situation like one may play a game of size 8 GB in a computer with 4 GB of RAM only. One can run multiple programs whose combined size more than RAM size. This is because of virtual memory.

What is Virtual Memory?

Virtual Memory is a storage mechanism which gives illusion of having a very large main memory to the end user. It is possible by treating a part of secondary memory as the main memory. In Virtual Memory large programs divided into pages and stored into secondary memory. Only the required pages are loaded from the secondary memory into the main memory during the execution.

Example:

- Before virtual memory existed, a word processor, e-mail program, & browser couldn't be run at the same time unless there was enough memory to hold all three programs at once.
- This would mean that one would have to close one program in order to run the other, but with virtual memory, multitasking is possible.

Physical Memory Or Main Memory	RAM
Secondary Memory	HDD or SSD
Virtual Memory	Illusion created by OS for optimization(You can say this definition too)
VMM	Virtual Memory Manager

How VM works:

- Let's say that an OS needs 120 MB of memory in order to hold all the running programs.
- But there's currently only 50 MB of available physical memory .
- So, The VMM will create a file on the hard disk which is 70 MB (120 – 50) in size.
- This file called as paging file (also known as swap-file) .Here, the original program broken into number of pages with same size.
- Whenever the OS needs a 'block' of memory that's not in the main(RAM) memory, the VMM swap out a block(old block) from the main memory which is not used recently.

- The VMM takes a block(which needed currently) from the paging file & moves it into the main memory – in place of the old block.
- This process is called swapping (also known as paging) .There are several algorithms for this process, called Page Replacement Algorithms.

Advantages of Virtual Memory

- The degree of Multiprogramming will be increased.
- User can run large application with less real RAM.
- There is no need to buy more memory RAMs.

Disadvantages of Virtual Memory

- The system becomes slower since swapping takes time.
- It takes more time in switching between applications.

Virtual Memory Can Slow Down Performance of the program (Thrashing)!

- We know that Accessing the hard disk take long time compare to the main memory (RAM).
- If the size of virtual memory is quite large in comparison to the main memory, then more swapping from the hard disk to main memory and main memory to hard disk will occur. This is called thrashing which can really slow down a system's performance.

Page fault:

- In virtual memory mechanism some block of memory is loaded in RAM & rest is stored as a paging file in secondary memory.
- A page fault occurs when a process tries to access a page which is not in RAM but available in secondary memory as paging file.
- There are many algorithms to decrease page fault rate, like LRU, FIFO etc.,

In some situation, no pages are loaded into the main memory initially, pages are only loaded when demanded by the process by generating page faults. This is called **Demand Paging.**

Page replacement algorithm:

Page replacement is a technique which is used to decide which page need to swap out and which page need to swap in. It decrease the page fault.

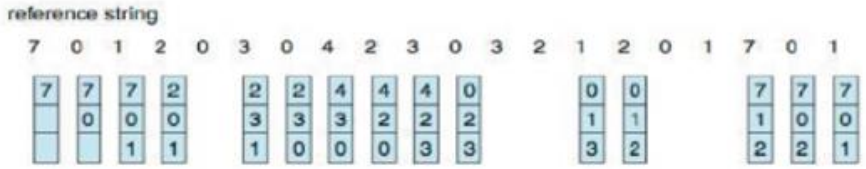
First In First Out (FIFO) algorithm

Oldest page in main memory will be selected for replacement.

Results depend on number of frames available

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.
- Example:
  - Reference string: **7,0,1,2,0,3,0,4,2,3,0,3,0,3,2,1,2,0,1,7,0,1**
  - 3 frames (3 pages can be in memory at a time per process)



What is Belady's Anomaly?

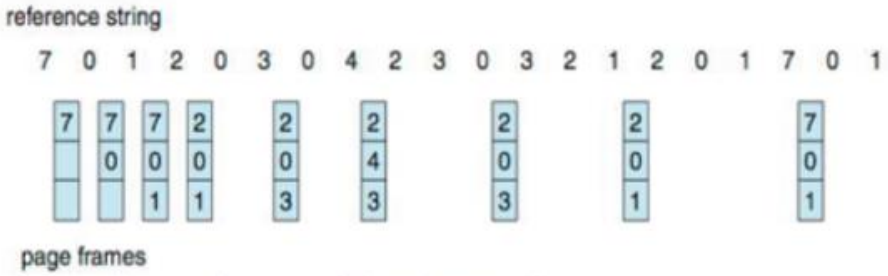
Belady's anomaly is an anomaly with some page replacement algorithm .This algorithm increasing the number of page frames. It causes the increasing of number of page faults. It occurs with First in First Out page replacement algorithm.

- Example:For the reference string 1,2,3,4,1,2,5,1,2,3,4,5

  page fault is 9 with frames 3 and page fault is 10 with frames 4.

Optimal Page Replacement

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



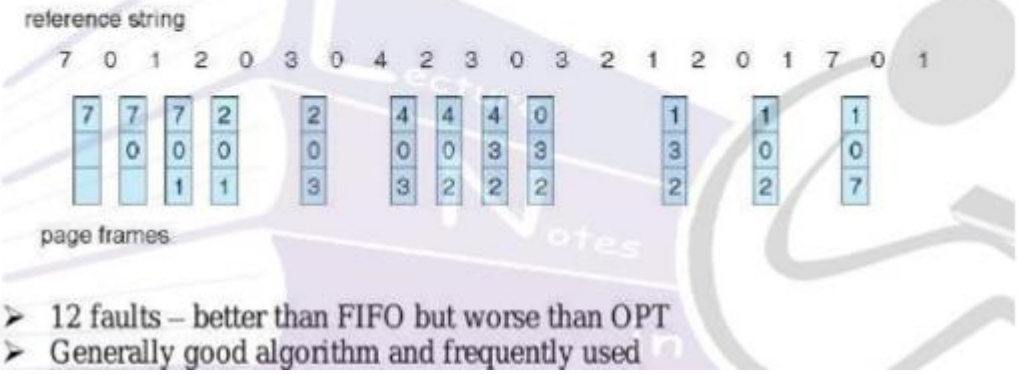
- It is difficult to implement, because it requires future knowledge of the reference string.

Least Recently Used (LRU) algorithm

Page which has not been used for the longest period of time is selected for replacement.

LRU Page Replacement

- Use past knowledge rather than future
- Replace page that has not been used in the most amount of time
- Associate time of last use with each page



- 12 faults – better than FIFO but worse than OPT
- Generally good algorithm and frequently used

File

A file is a collection of related information which is stored on secondary storage.

FILE DIRECTORIES:
Collection of files is a file directory. The directory contains information about the files, including attributes, location and ownership.

File Allocation Methods:

1. **Continuous Allocation**: A single continuous set of memory blocks is allocated to a file at the time of file creation.
2. **Non-contiguous allocation**: Allocation is on an individual block basis. Each block contains a pointer to the next block in the chain (like linked list).
3. **Indexed Allocation**: It addresses many of the problems of contiguous and linked allocation. In this case, the file allocation table contains a separate index for each file

Disk Scheduling

Disk scheduling used for schedule I/O request. Disk scheduling is also known as I/O scheduling.

1.Seek Time: Seek time is the time taken to locate the disk arm where the data is to be read or write.

2. **Rotational Latency**: Rotational Latency is the time taken by the desired sector of disk to rotate into a position so that it can access the read/write heads.
3. **Transfer Time:** Transfer time is the time to transfer the data. It depends on the rotating speed of the disk and number of bytes to be transferred.
4. **Disk Access Time**: Seek Time + Rotational Latency + Transfer Time
5. **Disk Response Time:** Response Time is the average time spent by a request waiting to perform its I/O operation. Average Response time is the response time of the all requests.

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.
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 a 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.
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 the disk, it reverses its direction and again services the request arriving in its path. So, this algorithm works like an elevator and hence also known as elevator algorithm.
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.
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 a 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.

All the best...

With