OPERATING SYSTEM

1. Introduction
2. Operating system is a system software
3. It acts as an intermediary between hardware and user.
4. Resouce manaer – manages resources in an unbiased fashion both hardware and software.
5. Provides s platform on which other application programs are installed.

User 2

User 1

Abstract view of system

Operating system

MS office

Assembler

VLC

NFS

Computer hardware

Compiler

1. Goals of operating system
2. Primary goal: convenience / user friendly
3. Secondary goal: Efficiency
4. Functions of operating system
5. Process management
6. Memory management
7. I/O device management
8. File management
9. Network management
10. Security and protection
11. Evolution of operating system
12. Mainframe computers

* Common i/o and operating devices were card readers and tape drivers.
* User prepare a job which consist of the program input data and control instructions
* Input jo is given in the form of punch cards and the result also appears in form of punch card after processing.
* So operating system was always present in memory, major task was to transfer the control from one job to the other.

1. Batch operating system

* Jobs with similar needs are batched together and executed through the processor as a group.
* Operator sort jobs as a deck of pumnch cards into batch with similar needs .
* Ex: FORTRAN batch, COBOL batch etc.
* Advantages : - In a batch job executes one after the other saving time for activitiers like loading compiler.
* During a batch execution no natural instruction is needed.
* Disadvantages: - Memory limitation
* Interaction of i/o and operating devices directly with the CPU.

User 3

User2

User 1

Job49

Job45

Job42

Job 5

Job4

User program

OS

CPU

Operator

1. Multiprogramming operating system

* Maximize CPU utilization
* Multiprogramming means more than one process in main memory which is ready to execute
* Process generally require CPU time and i/o time, so if running process perform i/o are some other event which do not require CPU then instead of sitting ideal CPU make a context switch and picks some other process and this idea will continue.
* CPU never gets ideal unless there is no process ready to execute or at the time of context switch.
* Advantages: - High CPU utilization
* Less waiting/response time ect
* May be extended to multiple users
* Disadvantages: - Difficult scheduling
* Main memory management is required
* Memory fragmentation
* Paging

1. Multitasking operating system/ Time sharing / fair share/ Multitasking with round robin

* Multitasking is multiprogramming with time sharing.
* There is only one CPU but switches between processes so quickly that it gives an illusion that all are executing at the same time.
* The task in multitasking is refereed as multiple thread of the same program.
* The main idea is to give better response time and executing multiple process together.

1. Processes in operating system

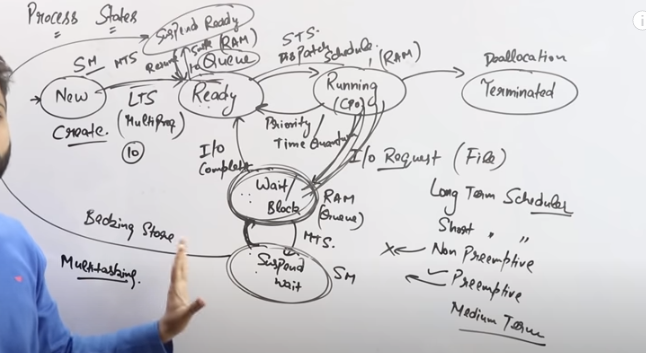
* A process is a program in execution which then forms the basis of all computation.

1. Process memory

* **Process memory** is divided into four sections for efficient working :
* The **Text section** is made up of the compiled program code, read in from non-volatile storage when the program is launched.
* The **Data section** is made up of the global and static variables, allocated and initialized prior to executing the main.
* 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. Space on the stack is reserved for local variables when they are declared.

1. Process states

* NEW- The process is being created.
* READY- The process is waiting to be assigned to a processor.
* RUNNING- Instructions are being executed.
* WAITING- The process is waiting for some event to occur(such as an I/O completion or reception of a signal).
* TERMINATED- The process has finished execution.



#kernal mode - When CPU is in kernel mode, the code being executed can access any memory address and any hardware resource.

Hence kernel mode is a very privileged and powerful mode.

If a program crashes in kernel mode, the entire system will be halted

# User mode - When CPU is in user mode, the programs don't have direct access to memory and hardware resources.

In user mode, if any program crashes, only that particular program is halted.

That means the system will be in a safe state even if a program in user mode crashes.

Hence, most programs in an OS run in user mode.

1. System calls

When a program in user mode requires access to RAM or a hardware resource, it must ask the kernel to provide access to that resource. This is done via something called a system call.

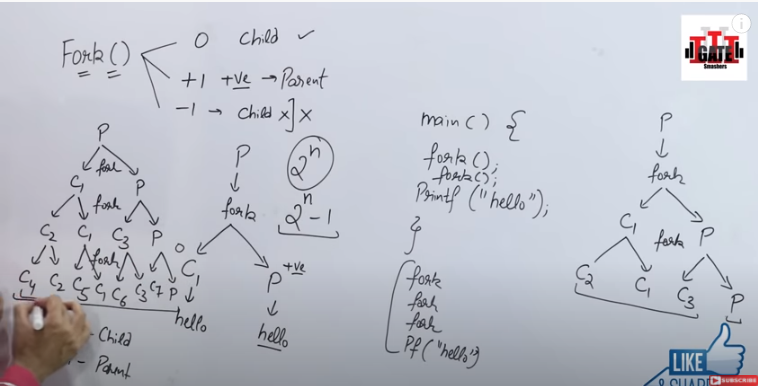
When a program makes a system call, the mode is switched from user mode to kernel mode. This is called a context switch.

Then the kernel provides the resource which the program requested. After that, another context switch happens which results in change of mode from kernel mode back to user mode.

* + File related – open(), read(),write(),close(), create file etc.
  + Device related – Read, Write , Reposition, ioctl, fcntl etc.
  + Information – get Pid, attributes, get System time and data
  + Process control – Load, Execute, abort, fork, wait, signal etc.
  + Communication – Pipe(), Create/delete connections, Shmget()
    - * Fork()

The fork() system call is used to create processes. When a process (a program in execution) makes a fork() call, an exact copy of the process is created. Now there are two processes, one being the parent process and the other being the child process.

The process which called the fork() call is the parent process and the process which is created newly is called the child process. The child process will be exactly the same as the parent. Note that the process state of the parent i.e., the address space, variables, open files etc. is copied into the child process. This means that the parent and child processes have identical but physically different address spaces. The change of values in parent process doesn't affect the child and vice versa is true too.



1. Process vs thread
2. Process
   * + - System calls are involved in the process.
       - OS treats different process differently
       - Different process has different copies of data, file , code .
       - Context switching is slower.
       - Blocking a process will not block another process
       - Independent
3. Thread
   * + - System calls are not involved in the process.
       - All user level threads treated as single task for OS.
       - Thread share same copies of data, file , code .
       - Context switching is faster.
       - Blocking a thread will block another
       - Interdependent
4. User level vs kernel level thread
5. User level thread
   * + - User level thread are managed by user level libraries
       - User level threads are typically fast.
       - Context switching is fatser.
       - If one user level threads perform blocking operation then the netire process gets blocked.
6. Kernel level thread
   * + - Kernel level thread are managed by IS.
       - Kernel level threads are solwer than the user level thread.
       - Context switching is slower.
       - If one kernel level thread is blocked there is no affect on the other.
7. CPU scheduling

* A process exection consist of a cycle of cpu execution and i/o execution.
* Normaly every [rpcess begins with cpu burst that may be followed by the i/o burst then another cpu burst then another i/o burst and eventually it qill end at cpu burst.
* CPU process – These are the process which needs most of the CPU time.
* I/O process – These are those process which needs most of the i/o devices

1. Types of cpu scheduling

* Non Pre-empitive approach
* When a process completes its execution its is known as preempitive cpu scheduling
* In this the process leaves the cpu voluntarily to perform some i/o operation on to wait for an event.
* Pre-empitive approach
* If a process enters in the ready state either form the new or waiting state and it is a high prority process.
* If a process switches from the running state to the ready state because of the quantum expies.

1. Terminologies

* Burst time/execution time/running time – It is a time a process requires for running on the CPU.
* Waiting time – Time spent by a process in ready state waiting for the CPU.
* Arrival time – Whena orocess enters ready state.
* Exit time – When a process completes its exection and exits from the CPU.
* Turn around time – Total time spent by the process in the system.

TAT = EXIT TIME – ARRIVAL TIME = BURST TIME + WAITING TIME

* Response time – Time between the process enters ready state and get scheduled on the CPU for the first time.

1. Criteria for CPU schedulimng algorithm

* Average waiting time
* Average response time
* CPU utilization
* Throughput – No. of process executed per unit time.

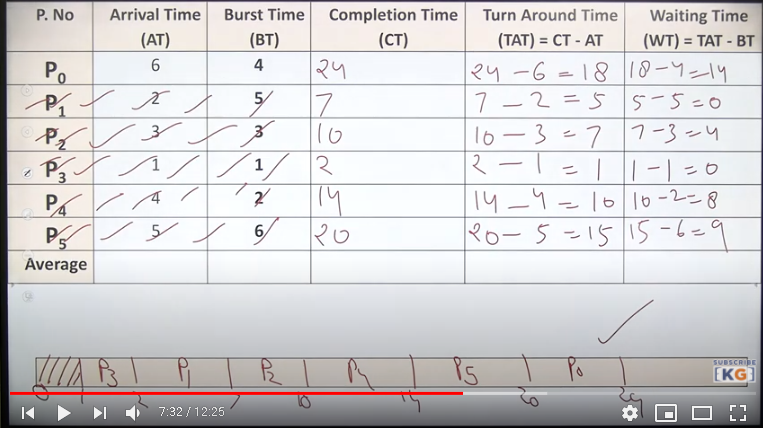
1. Scheduling algorithms

* First come first serve (FCFS)/ First in first out (FIFO)
* Simplest scheduling algorithm, it assigns CPU to the process which arises first.
* Easy to understand and can easilty be implemented using queue data structure.
* Always non- preempitie in nature.
* Convoy effect – Smaller process has to wait for long time for bigger process to release CPU.
* Advantages – Simple, easy to use, easy to understand, easy to implement and must be used for background process where execution is not urgent.
* Disadvantages – Suffer from convoy effect normally higher average waiting time, no consideration to the priority or burst time, should not be used for interactive system.

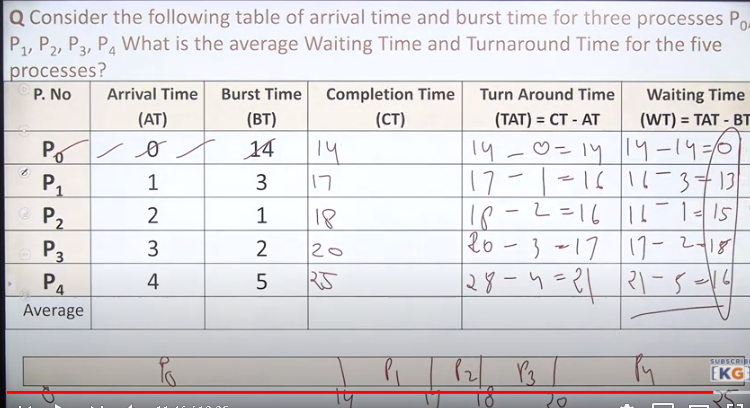
Q. Numerical on finding TAT and WT using AT and BT.



Q. Numerical on finding TAT and WT using AT and BT.



Q. Numerical on finding TAT and WT using AT and BT.



* Shortest job first schedulinig (SJF)
* Shortest Job First scheduling works on the process with the shortest burst time or duration first.
* This is the best approach to minimize waiting time.
* This is used in [Batch Systems](https://www.studytonight.com/operating-system/types-of-os).
* It is of two types:- Non Pre-emptive

- Pre-emptive

- To successfully implement it, the burst time/duration time of the processes should be known to the processor in advance, which is practically not feasible all the time.

- This scheduling algorithm is optimal if all the jobs/processes are available at the same time. (either Arrival time is 0 for all, or Arrival time is same for all)

- Advantages – SRTF (pre-empitive)guarantees minimum average waiting time.

- Better average response time as compared to FCFS

- Disadvantages – Algorithm cannot be implemented as there is no way to know the burst time of a process.

- Process with larger cpu burst time requirement will go into starvation.

- No idea of priority process withlarge burst time, have poor response time.

* Priority scheduling - In the [Shortest Job First](https://www.studytonight.com/operating-system/shortest-job-first) scheduling algorithm, the priority of a process is generally the inverse of the CPU burst time, i.e. the larger the burst time the lower is the priority of that process.

- In case of priority scheduling the priority is not always set as the inverse of the CPU burst time, rather it can be internally or externally set, but yes the scheduling is done on the basis of priority of the process where the process which is most urgent is processed first, followed by the ones with lesser priority in order.

- Processes with same priority are executed in FCFS manner.

- The priority of process, when internally defined, can be decided based on memory requirements, time limits ,number of open files, ratio of I/O burst to CPU burst etc.

- Types : \*\* Preemptive Priority Scheduling: If the new process arrived at the ready queue has a higher priority than the currently running process, the CPU is preempted, which means the processing of the current process is stoped and the incoming new process with higher priority gets the CPU for its execution.

\*\* Non-Preemptive Priority Scheduling: In case of non-preemptive priority scheduling algorithm if a new process arrives with a higher priority than the current running process, the incoming process is put at the head of the ready queue, which means after the execution of the current process it will be processed.

* Problem with priority scheduling: n priority scheduling algorithm, the chances of indefinite blocking or starvation.

A process is considered blocked when it is ready to run but has to wait for the CPU as some other process is running currently.

But in case of priority scheduling if new higher priority processes keeps coming in the ready queue then the processes waiting in the ready queue with lower priority may have to wait for long durations before getting the CPU for execution.

* Advantages: Provides a facility of priority specially for system process.

Allows to run important processes first even it is a user process.

* Disadvantages: Here process with higher priority may strave for the CPU.

No idea of response time or waiting time.

* + - * Round robin scheduling - Round Robin(RR) scheduling algorithm is mainly designed for time-sharing systems. This algorithm is similar to FCFS scheduling, but in Round Robin(RR) scheduling, preemption is added which enables the system to switch between processes.
  + A fixed time is allotted to each process, called a **quantum**, for execution.
  + Once a process is executed for the given time period that process is preempted and another process executes for the given time period.
  + Context switching is used to save states of preempted processes.
  + This algorithm is simple and easy to implement and the most important is thing is this algorithm is starvation-free as all processes get a fair share of CPU.
  + It is important to note here that the length of time quantum is generally from 10 to 100 milliseconds in length.
  + Advantage – perform best in the terms of average response time.

- works well on the case of time sharing system, client server architecture and interacive system.

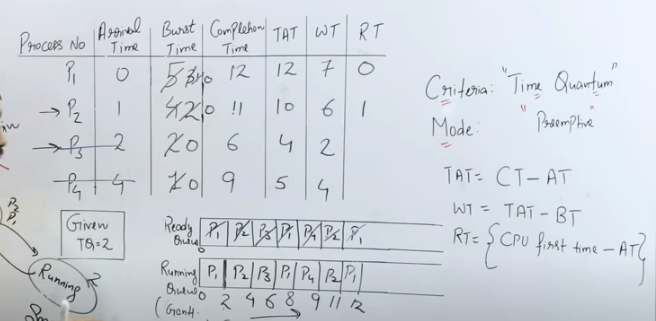
-kind of SJF implementation

\* Disadvantages – longer process may starve

- performance depends heavily in the quantum

- no idea of priority

Q. Numerical on finding TAT and WT using AT and BT.



1. Process synchronization

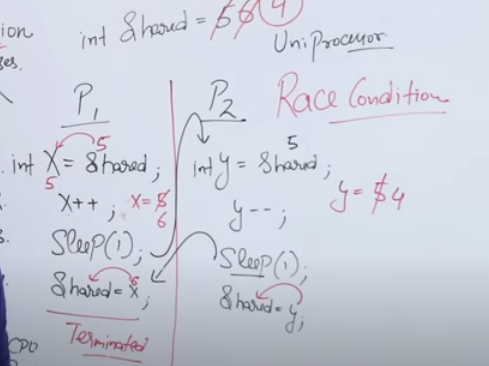
It is the task phenomenon of coordinating the execution of processes in such a way that no two processes can have access to the same shared data and resources.

* It is a procedure that is involved in order to preserve the appropriate order of execution of cooperative processes.
* In order to synchronize the processes, there are various synchronization mechanisms.
* Process Synchronization is mainly needed in a multi-process system when multiple processes are running together, and more than one processes try to gain access to the same shared resource or any data at the same time.

Process is categorized into two types on the basis of synchronization and these are given below:

* Independent Process - Two processes are said to be independent if the execution of one process does not affect the execution of another process.
* Cooperative Process - Two processes are said to be cooperative if the execution of one process affects the execution of another process. These processes need to be synchronized so that the order of execution can be guaranteed.

Ex:



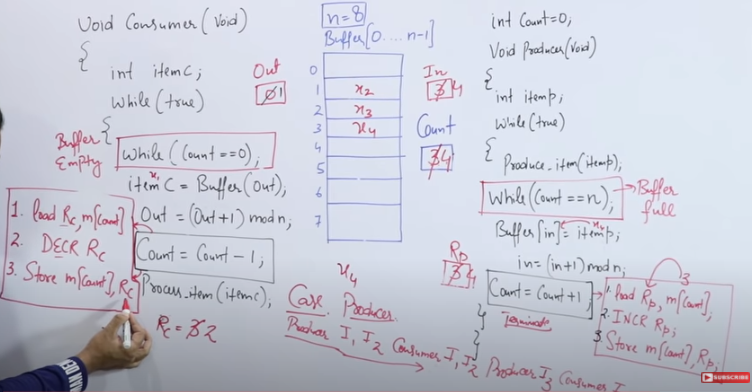
At the time when more than one process is either executing the same code or accessing the same memory or any shared variable; In that condition, there is a possibility that the output or the value of the shared variable is wrong so for that purpose all the processes are doing the race to say that my output is correct. This condition is commonly known as**a race condition.**

1. Classical problems on Synchronisation (cooperative)

(i) Bounded Buffer (Producer-Consumer) Problem

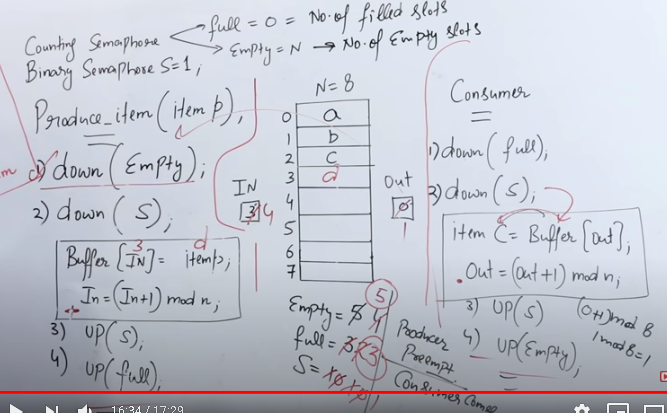
Because the buffer pool has a maximum size, this problem is often called the **Bounded buffer problem**.

* This problem is generalised in terms of the **Producer Consumer problem**, where a **finite** buffer pool is used to exchange messages between producer and consumer processes.
* Solution to this problem is, creating two counting semaphores "full" and "empty" to keep track of the current number of full and empty buffers respectively.
* In this Producers mainly produces a product and consumers consume the product, but both can use of one of the containers each time.
* The main complexity of this problem is that we must have to maintain the count for both empty and full containers that are available.

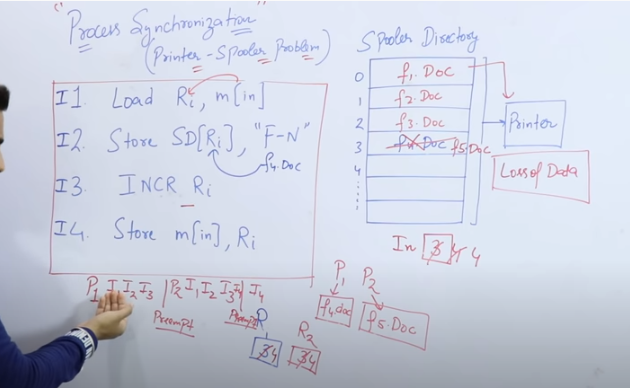


In the above example “In” value is changed from 4 to 2 . Which is incorrect and the cooperative process didn’t give the right answer.

Solution of producer consumer problem using binary semaphore-

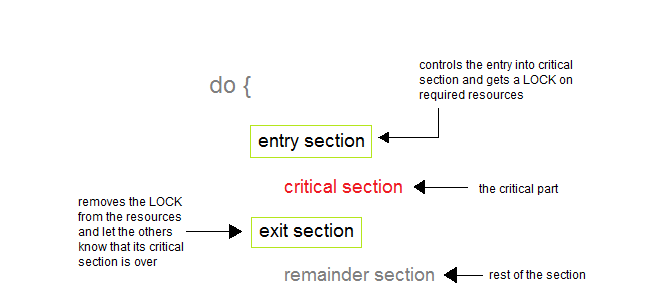


(ii) Printer – Spooler problem



1. Critical section problem

Critical section – It is a part of program where shared resources are accessed by various processes.

A Critical Section is a code segment that accesses shared variables and has to be executed as an atomic action. It means that in a group of cooperating processes, at a given point of time, only one process must be executing its critical section. If any other process also wants to execute its critical section, it must wait until the first one finishes. The entry to the critical section is mainly handled by wait() function while the exit from the critical section is controlled by the signal() function. 

Solution of Critical section problem

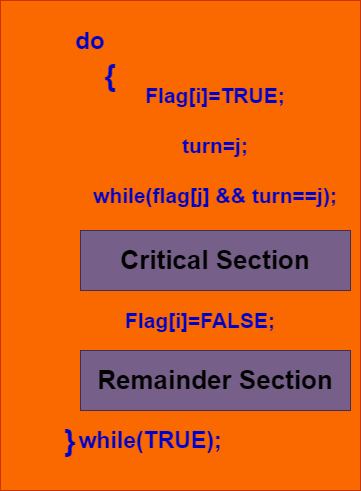
### Peterson's Solution

### This is widely used and software-based solution to critical section problems. Peterson's solution was developed by a computer scientist Peterson that's why it is named so.

With the help of this solution whenever a process is executing in any critical state, then the other process only executes the rest of the code, and vice-versa can happen. This method also helps to make sure of the thing that only a single process can run in the critical section at a specific time.

This solution preserves all three conditions:

* Mutual Exclusion is comforted as at any time only one process can access the critical section.
* Progress is also comforted, as a process that is outside the critical section is unable to block other processes from entering into the critical section.
* Bounded Waiting is assured as every process gets a fair chance to enter the Critical section.



The above shows the structure of process**Pi in Peterson's solution.**

* Suppose there are**N processes (P1, P2, ... PN)** and as at some point of time every process requires to enter in the **Critical Section**
* A **FLAG[]**array of size N is maintained here which is by default false. Whenever a process requires to enter in the critical section, it has to set its flag as true. Example: If Pi wants to enter it will set **FLAG[i]=TRUE.**
* Another variable is called**TURN** and is used to indicate the process number that is currently waiting to enter into the critical section.
* The process that enters into the critical section while exiting would change the**TURN** to another number from the list of processes that are ready.
* Example: If the turn is 3 then P3 enters the Critical section and while exiting turn=4 and therefore P4 breaks out of the wait loop.

### Synchronization Hardware

Many systems provide hardware support for critical section code. The critical section problem could be solved easily in a single-processor environment if we could disallow interrupts to occur while a shared variable or resource is being modified.

In this manner, we could be sure that the current sequence of instructions would be allowed to execute in order without pre-emption. Unfortunately, this solution is not feasible in a multiprocessor environment.

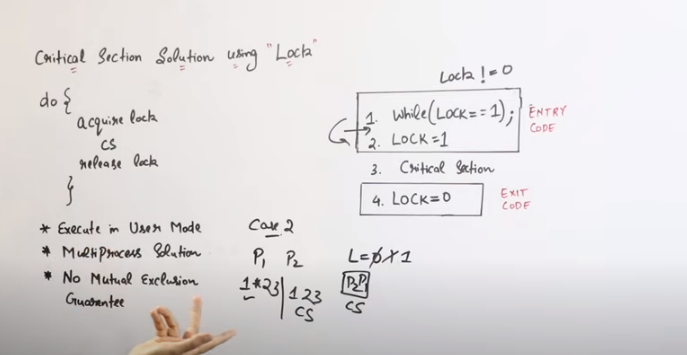
Disabling interrupt on a multiprocessor environment can be time-consuming as the message is passed to all the processors.

This message transmission lag delays the entry of threads into the critical section, and the system efficiency decreases.

### Mutex Locks

As the synchronization hardware solution is not easy to implement for everyone, a strict software approach called Mutex Locks was introduced. In this approach, in the entry section of code, a LOCK is acquired over the critical resources modified and used inside the critical section, and in the exit section that LOCK is released.

As the resource is locked while a process executes its critical section hence no other process can access it.



1. Reader writer problem

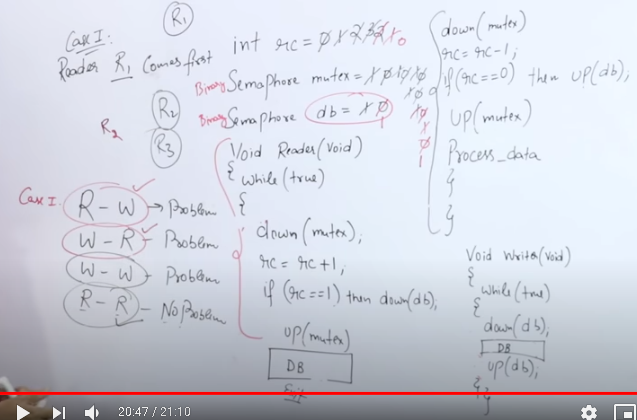
There is a shared resource which should be accessed by multiple processes. There are two types of processes in this context. They are reader and writer. Any number of readers can read from the shared resource simultaneously, but only one writer can write to the shared resource. When a writer is writing data to the resource, no other process can access the resource. A writer cannot write to the resource if there are non zero number of readers accessing the resource at that time.

Solution of Reader writer problem-

Readers have higher priority than writer. If a writer wants to write to the resource, it must wait until there are no readers currently accessing that resource.

Here, we use one mutex m and a semaphore w. An integer variable rc is used to maintain the number of readers currently accessing the resource. The variable rc is initialized to 0. A value of 1 is given initially to m and w.

Instead of having the process to acquire lock on the shared resource, we use the mutex m to make the process to acquire and release lock whenever it is updating the rc variable.



1. Dining philosophers Problem

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.

Solution of Dining philosopher problem-

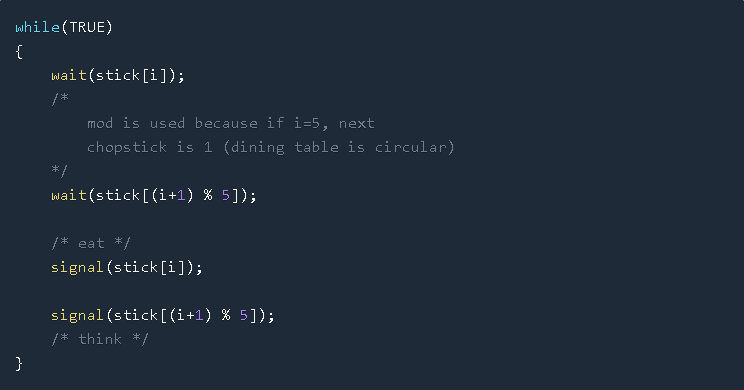
From the problem statement, it is clear that a philosopher can think for an indefinite amount of time. But when a philosopher starts eating, he has to stop at some point of time. The philosopher is in an endless cycle of thinking and eating.

An array of five semaphores, stick[5], for each of the five chopsticks.

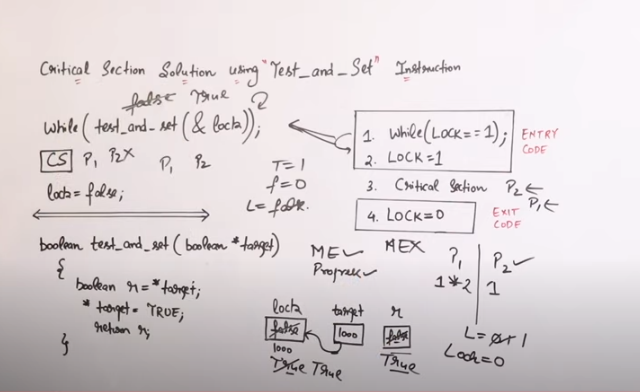
When a philosopher wants to eat the rice, he will wait for the chopstick at his left and picks up that chopstick. Then he waits for the right chopstick to be available, and then picks it too. After eating, he puts both the chopsticks down.

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 forever. The possible solutions for this are:

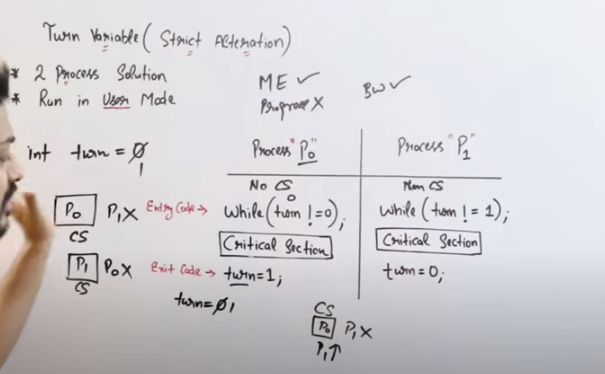
* A philosopher must be allowed to pick up the chopsticks only if both the left and right chopsticks are available.
* Allow only four philosophers to sit at the table. That way, if all the four philosophers pick up four chopsticks, there will be one chopstick left on the table. So, one philosopher can start eating and eventually, two chopsticks will be available. In this way, deadlocks can be avoided.



1. Test and Set Instruction



1. Turn Variable ( Strict Alteration Method)



1. Synchronization rules

### Mutual Exclusion

Out of a group of cooperating processes, only one process can be in its critical section at a given point of time.

### Progress

If no process is in its critical section, and if one or more threads want to execute their critical section then any one of these threads must be allowed to get into its critical section.

### Bounded Waiting

After a process makes a request for getting into its critical section, there is a limit for how many other processes can get into their critical section, before this process's request is granted. So after the limit is reached, the system must grant the process permission to get into its critical section.

15. Semaphores

It is a synchronizing tool and is accessed only through two low standard atomic operations, wait and signal designated by P(S) and V(S) respectively.

In very simple words, the semaphore is a variable that can hold only a non-negative Integer value, shared between all the threads, with operations wait and signal, which work as follow:

P P(S): if S >= 1 then S := S - 1

else <block and enqueue the process>;

V(S): if <some process is blocked on the queue>

then <unblock a process>

else S := S + 1;

The classical definitions of wait and signal are:

* Wait: This operation decrements the value of its argument S, as soon as it would become non-negative(greater than or equal to 1). This Operation mainly helps you to control the entry of a task into the critical section. In the case of the negative or zero value, no operation is executed. wait() operation was originally termed as P; so it is also known as P(S) operation. The definition of wait operation is as follows:

wait(S)

{

while (S<=0);//no operation

S--;

}

Note:

When one process modifies the value of a semaphore then, no other process can simultaneously modify that same semaphore's value. In the above case the integer value of S(S<=0) as well as the possible modification that is S-- must be executed without any interruption.

* Signal: Increments the value of its argument S, as there is no more process blocked on the queue. This Operation is mainly used to control the exit of a task from the critical section.signal() operation was originally termed as V; so it is also known as V(S) operation. The definition of signal operation is as follows:

signal(S)

{

S++;

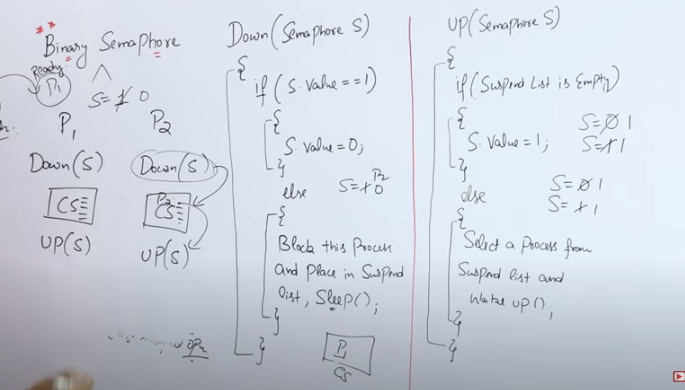
}

* Also, note that all the modifications to the integer value of semaphore in the wait() and signal() operations must be executed indivisibly.
* Properties
* It's simple and always have a non-negative integer value.
* Works with many processes.
* Can have many different critical sections with different semaphores.
* Each critical section has unique access semaphores.
* Can permit multiple processes into the critical section at once, if desirable.

(ii) Types

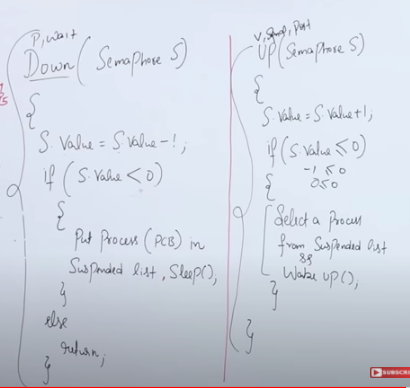
* **Binary Semaphore:**

It is a special form of semaphore used for implementing mutual exclusion, hence it is often called a **Mutex**. A binary semaphore is initialized to 1 and only takes the values 0 and 1 during the execution of a program. In Binary Semaphore, the wait operation works only if the value of semaphore = 1, and the signal operation succeeds when the semaphore= 0. Binary Semaphores are easier to implement than counting semaphores.



* **Counting Semaphores:**

These are used to implement **bounded concurrency**. The Counting semaphores can range over an **unrestricted domain**. These can be used to control access to a given resource that consists of a finite number of Instances. Here the semaphore count is used to indicate the number of available resources. If the resources are added then the semaphore count automatically gets incremented and if the resources are removed, the count is decremented. Counting Semaphore has no mutual exclusion.



## (iii)Advantages of Semaphores

* With the help of semaphores, there is a flexible management of resources.
* Semaphores are machine-independent and they should be run in the machine-independent code of the microkernel.
* Semaphores do not allow multiple processes to enter in the critical section.
* They allow more than one thread to access the critical section.
* As semaphores follow the mutual exclusion principle strictly and these are much more efficient than some other methods of synchronization.

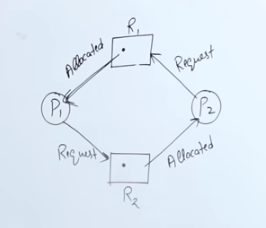
## Disadvantages of Semaphores

* One of the biggest limitations is that semaphores may lead to priority inversion; where low priority processes may access the critical section first and high priority processes may access the critical section later.
* To avoid deadlocks in the semaphore, the Wait and Signal operations are required to be executed in the correct order.
* Using semaphores at a large scale is impractical; as their use leads to loss of modularity and this happens because the wait() and signal() operations prevent the creation of the structured layout for the system.
* Their use is not enforced but is by convention only.

16. Deadlock

\* In a multiprogramming system a number of process compete for limited number of resources and if a resource is not available at that instance then process enters in the wainting state.

\* If a process is unabke to change its waiting state indefinitely because the resources requested by it are held by aniother waiting process then the system is said to be in deadlock.



(i) Conditions for deadlock

The deadlock situation can only arise if all the following four conditions hold simultaneously:

* Mutual Exclusion

According to this condition, atleast one resource should be non-shareable (non-shareable resources are those that can be used by one process at a time.)

* Hold and wait

According to this condition, A process is holding atleast one resource and is waiting for additional resources.

* NO preemption

Resources cannot be taken from the process because resources can be released only voluntarily by the process holding them.

* Circular wait

In this condition, the set of processes are waiting for each other in the circular form.

The above four conditions are not completely independent as the circular wait condition implies the hold and wait condition. We emphasize on the fact that all four conditions must hold for a deadlock.

(ii) Handling deadlock

\* Prevention – It means to design such a system which violate atleast one of the four necessary conditions of deadlock and ensure independence from deadlock.

\* Avoidance – System maintains a set of data using which it takes a decision whether to entertain the request or not, to be in a safe condition.

\* Detection and recovery – Here we wait until the deadlock occur and once we detect it we recorver from it.

\* Ignorance / ostrich algo - We ignore the problem as it doesn’t exist

1. Deadlock prevention

* Mutual exclusion

This condition must hold for non-sharable resources. For example, a printer cannot be simultaneously shared by several processes. In contrast, Sharable resources do not require mutually exclusive access and thus cannot be involved in a deadlock. A good example of a sharable resource is Read-only files because if several processes attempt to open a read-only file at the same time, then they can be granted simultaneous access to the file.

A process need not to wait for the sharable resource. Generally, deadlocks cannot be prevented by denying the mutual exclusion condition because there are some resources that are intrinsically non-sharable.

* Hold and wait
  1. conservative approach – A process is allowed to start execution if and only if it has accquiered all the resources(it is less efficient, non implementable,easy,deadlock independent).
  2. Do not hold – Process will acquire only desired resources but before making any fresh request it must release all the resources that t is currently holding ( it is efficient and implementable)
  3. Wait timeouts – We place a maximum time upto which a process can wait after which the process must release all the holding resources
* No preemption

1. According to the First Protocol: "If a process that is already holding some resources requests another resource and if the requested resources cannot be allocated to it, then it must release all the resources currently allocated to it."
2. According to the Second Protocol: "When a process requests some resources, if they are available, then allocate them. If in case the requested resource is not available then we will check whether it is being used or is allocated to some other process waiting for other resources. If that resource is not being used, then the operating system preempts it from the waiting process and allocate it to the requesting process. And if that resource is being used, then the requesting process must wait".

The second protocol can be applied to those resources whose state can be easily saved and restored later for example CPU registers and memory space, and cannot be applied to resources like printers and tape drivers.

* Circular wait

Assign a priority number to each resource. There will be a condition that any process cannot request for a lesser priority resource. This method ensures that not a single process can request a resource that is being utilized by any other process and due to which no cycle will be formed.

1. Deadlock avoidance

Banker's algorithm is a deadlock avoidance algorithm. It is named so because this algorithm is used in banking systems to determine whether a loan can be granted or not.

Consider there are n account holders in a bank and the sum of the money in all of their accounts is S. Every time a loan has to be granted by the bank, it subtracts the loan amount from the total money the bank has. Then it checks if that difference is greater than S. It is done because, only then, the bank would have enough money even if all the n account holders draw all their money at once.

Banker's algorithm works in a similar way in computers.

Whenever a new process is created, it must specify the maximum instances of each resource type that it needs, exactly

* .Data Structures used to implement the Banker’s Algorithm

Some data structures that are used to implement the banker's algorithm are:

* Available

It is an array of length m. It represents the number of available resources of each type. If Available[j] = k, then there are k instances available, of resource type Rj.

* Max

It is an n x m matrix which represents the maximum number of instances of each resource that a process can request. If Max[i][j] = k, then the process Pi can request atmost k instances of resource type Rj.

* Allocation

It is an n x m matrix which represents the number of resources of each type currently allocated to each process. If Allocation[i][j] = k, then process Pi is currently allocated k instances of resource type Rj.

* Need

It is a two-dimensional array. It is an n x m matrix which indicates the remaining resource needs of each process. If Need[i][j] = k, then process Pi may need k more instances of resource type Rj to complete its task.

Need[i][j] = Max[i][j] - Allocation [i][j]

* Banker’s algorithm comprises of two algorithms:
* Safety algorithm
* Resource request algorithm

Safety Algorithm

A safety algorithm is an algorithm used to find whether or not a system is in its safe state. The algorithm is as follows:

Let Work and Finish be vectors of length m and n, respectively. Initially,

Work = Available

Finish[i] =false for i = 0, 1, ... , n - 1.

This means, initially, no process has finished and the number of available resources is represented by the Available array.

Find an index i such that both

Finish[i] ==false

Needi <= Work

If there is no such i present, then proceed to step 4.

It means, we need to find an unfinished process whose needs can be satisfied by the available resources. If no such process exists, just go to step 4.

Perform the following:

Work = Work + Allocationi

Finish[i] = true

Go to step 2.

When an unfinished process is found, then the resources are allocated and the process is marked finished. And then, the loop is repeated to check the same for all other processes.

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

That means if all processes are finished, then the system is in safe state.

This algorithm may require an order of mxn² operations in order to determine whether a state is safe or not.

Resource Request Algorithm

Now the next algorithm is a resource-request algorithm and it is mainly used to determine whether requests can be safely granted or not.

Let Requesti be the request vector for the process Pi. If Requesti[j]==k, then process Pi wants k instance of Resource type Rj.When a request for resources is made by the process Pi, the following are the actions that will be taken.

1. If Requesti <= Needi, then go to step 2;else raise an error condition, since the process has exceeded its maximum claim.

2.If Requesti <= Availablei then go to step 3; else Pi must have to wait as resources are not available.

3.Now we will assume that resources are assigned to process Pi and thus perform the following steps:

Available= Available-Requesti;

Allocationi=Allocationi +Requesti;

Needi =Needi - Requesti;

If the resulting resource allocation state comes out to be safe, then the transaction is completed and, process Pi is allocated its resources. But in this case, if the new state is unsafe, then Pi waits for Requesti, and the old resource-allocation state is restored.

* Disadvantages of Banker's Algorithm

Some disadvantages of this algorithm are as follows:

* During the time of Processing, this algorithm does not permit a process to change its maximum need.
* Another disadvantage of this algorithm is that all the processes must know in advance about the maximum resource needs.
* This algorithm permits the requests to be provided in constrained time, but for one year which is a fixed period.

17. Starvation vs Deadlock

18. System model

\* Every process will request for the resource

\* If entertained, then the process will use the resource.

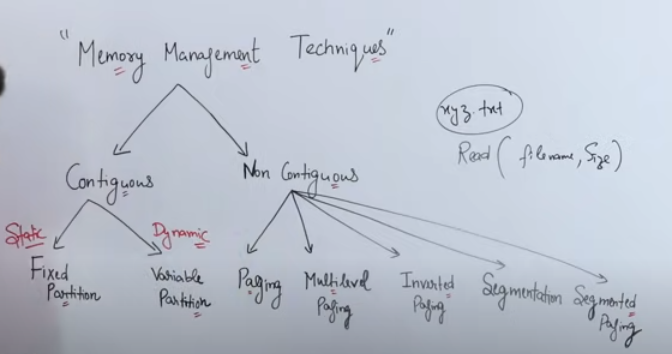
\* Process must release the resource after the use.

19. Memory Management

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

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

20. Memory management techniques



.

* 1. Contiguous Memory Allocation

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

## Fixed-size Partition Scheme

This technique is also known as Static partitioning. In this scheme, the system divides the memory into fixed-size partitions. The partitions may or may not be the same size. The size of each partition is fixed as indicated by the name of the technique and it cannot be changed.

In this partition scheme, each partition may contain exactly one process. There is a problem that this technique will limit the degree of multiprogramming because the number of partitions will basically decide the number of processes.

Whenever any process terminates then the partition becomes available for another process.

### Advantages of Fixed-size Partition Scheme

* This scheme is simple and is easy to implement
* It supports multiprogramming as multiple processes can be stored inside the main memory.
* Management is easy using this scheme

### Disadvantages of Fixed-size Partition Scheme

Some disadvantages of using this scheme are as follows:

* **Internal Fragmentation**

Suppose the size of the process is lesser than the size of the partition in that case some size of the partition gets wasted and remains unused. This wastage inside the memory is generally termed as Internal fragmentation

As we have shown in the above diagram the 70 KB partition is used to load a process of 50 KB so the remaining 20 KB got wasted.

* **Limitation on the size of the process**

If in a case size of a process is more than that of a maximum-sized partition then that process cannot be loaded into the memory. Due to this, a condition is imposed on the size of the process and it is: the size of the process cannot be larger than the size of the largest partition.

* **External Fragmentation**

It is another drawback of the fixed-size partition scheme as total unused space by various partitions cannot be used in order to load the processes even though there is the availability of space but it is not in the contiguous fashion.

* **Degree of multiprogramming is less**

In this partition scheme, as the size of the partition cannot change according to the size of the process. Thus the degree of multiprogramming is very less and is fixed.

## Variable-size Partition Scheme

This scheme is also known as **Dynamic partitioning** and is came into existence to overcome the drawback i.e internal fragmentation that is caused by **Static partitioning**. In this partitioning, scheme allocation is done dynamically.

The size of the partition is not declared initially. Whenever any process arrives, a partition of size equal to the size of the process is created and then allocated to the process. Thus the size of each partition is equal to the size of the process.

### Advantages of Variable-size Partition Scheme

Some Advantages of using this partition scheme are as follows:

* **No Internal Fragmentation** As in this partition scheme space in the main memory is allocated strictly according to the requirement of the process thus there is no chance of internal fragmentation. Also, there will be no unused space left in the partition.
* **Degree of Multiprogramming is Dynamic** As there is no internal fragmentation in this partition scheme due to which there is no unused space in the memory. Thus more processes can be loaded into the memory at the same time.
* **No Limitation on the Size of Process** In this partition scheme as the partition is allocated to the process dynamically thus the size of the process cannot be restricted because the partition size is decided according to the process size.

### Disadvantages of Variable-size Partition Scheme

Some Disadvantages of using this partition scheme are as follows:

* **External Fragmentation** As there is no internal fragmentation which is an advantage of using this partition scheme does not mean there will no external fragmentation. Let us understand this with the help of an example: In the above diagram- process P1(3MB) and process P3(8MB) completed their execution. Hence there are two spaces left i.e. 3MB and 8MB. Let’s there is a Process P4 of size 15 MB comes. But the empty space in memory cannot be allocated as no spanning is allowed in contiguous allocation. Because the rule says that process must be continuously present in the main memory in order to get executed. Thus it results in External Fragmentation.
* **Difficult Implementation** The implementation of this partition scheme is difficult as compared to the Fixed Partitioning scheme as it involves the allocation of memory at run-time rather than during the system configuration. As we know that OS keeps the track of all the partitions but here allocation and deallocation are done very frequently and partition size will be changed at each time so it will be difficult for the operating system to manage everything.
  1. Non - contiguous Memory Allocation
* Swapping

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

* Memory Protection

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

* Memory Allocation

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

First Fit:The first hole that is big enough is allocated to program.

Best Fit:The smallest hole that is big enough is allocated to program.

Worst Fit:The largest hole that is big enough is allocated to program.

* Fragmentation

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

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

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

* Paging

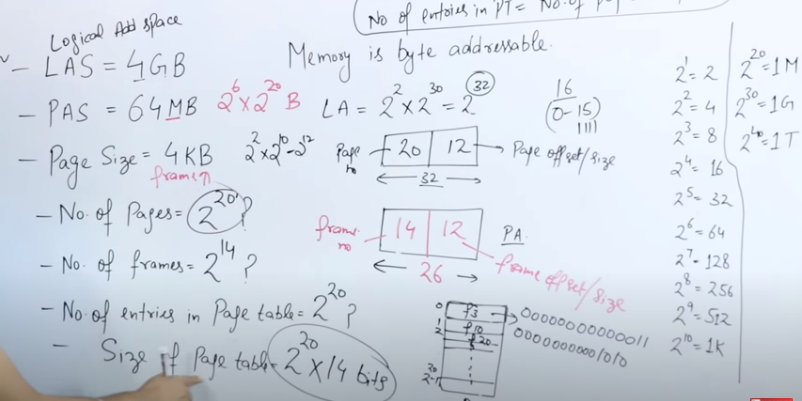
A solution to fragmentation problem is Paging. Paging is a memory management mechanism that allows the physical address space of a process to be non-contagious. Here physical memory is divided into blocks of equal size called Pages. The pages belonging to a certain process are loaded into available memory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

* Page Table

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

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

Question on paging and page table



* Segmentation

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

* Segmentation with Paging

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

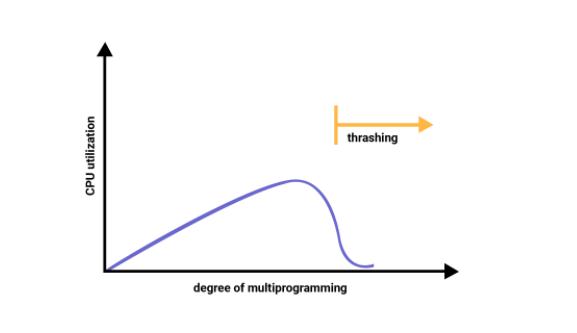
Segment numbers(S)

Page number (P)

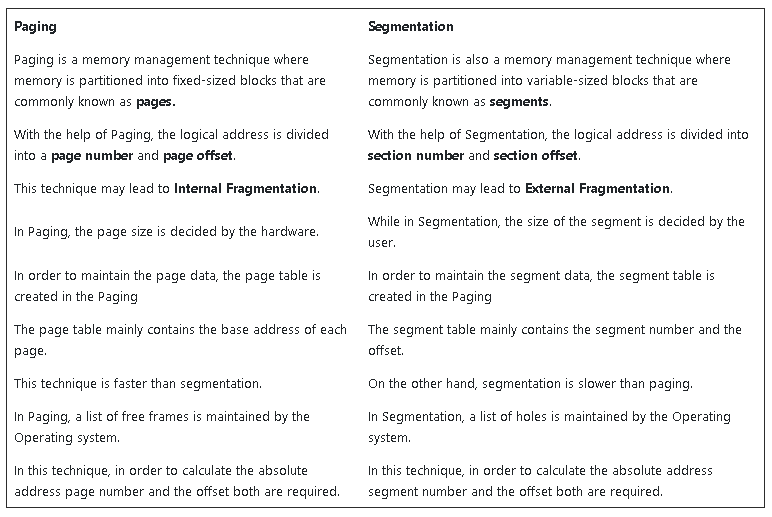
The displacement or offset number (D)

* Thrashing

In case, if the page fault and swapping happens very frequently at a higher rate, then the operating system has to spend more time swapping these pages. This state in the operating system is termed thrashing. Because of thrashing the CPU utilization is going to be reduced.



21. Differences between Paging and Segmentation



22. Virtual Memory

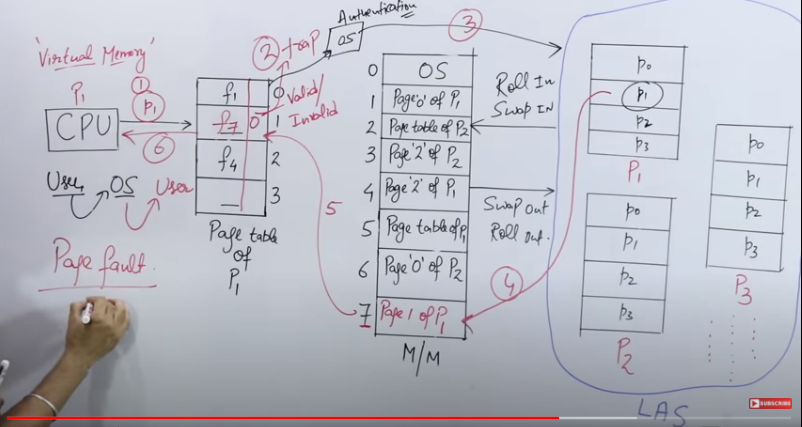
Virtual Memory is a space where large programs can store themselves in form of pages while their execution and only the required pages or portions of processes are loaded into the main memory. This technique is useful as a large virtual memory is provided for user programs when a very small physical memory is there. Thus Virtual memory is a technique that allows the execution of processes that are not in the physical memory completely.

Virtual Memory mainly gives the illusion of more physical memory than there really is with the help of Demand Paging.

In real scenarios, most processes never need all their pages at once, for the following reasons :

* Error handling code is not needed unless that specific error occurs, some of which are quite rare.
* Arrays are often over-sized for worst-case scenarios, and only a small fraction of the arrays are actually used in practice.
* Certain features of certain programs are rarely used.

In an Operating system, the memory is usually stored in the form of units that are known as pages. Basically, these are atomic units used to store large programs.



* 1. Need of Virtual Memory

Following are the reasons due to which there is a need for Virtual Memory:

* In case, if a computer running the Windows operating system needs more memory or RAM than the memory installed in the system then it uses a small portion of the hard drive for this purpose.
* Suppose there is a situation when your computer does not have space in the physical memory, then it writes things that it needs to remember into the hard disk in a swap file and that as virtual memory.

(ii) Benefits of having Virtual Memory

* Large programs can be written, as the virtual space available is huge compared to physical memory.
* Less I/O required leads to faster and easy swapping of processes.
* More physical memory available, as programs are stored on virtual memory, so they occupy very less space on actual physical memory.
* Therefore, the Logical address space can be much more larger than that of physical address space.
* Virtual memory allows address spaces to be shared by several processes.
* During the process creation, virtual memory allows: copy-on-write and Memory-mapped files

1. Advantages of Virtual Memory

Given below are the advantages of using Virtual Memory:

* Virtual Memory allows you to run more applications at a time.
* With the help of virtual memory, you can easily fit many large programs into smaller programs.
* With the help of Virtual memory, a multiprogramming environment can be easily implemented.
* As more processes should be maintained in the main memory which leads to the effective utilization of the CPU.
* Data should be read from disk at the time when required.

() Disadvantages of Virtual Memory

Given below are the drawbacks of using Virtual Memory:

* Virtual memory reduces the stability of the system.
* The performance of Virtual memory is not as good as that of RAM.
* If a system is using virtual memory then applications may run slower.
* Virtual memory negatively affects the overall performance of a system.
* Virtual memory occupies the storage space, which might be otherwise used for long term data storage.

23. Demand Paging

The demand paging system is somehow similar to the paging system with swapping where processes mainly reside in the main memory(usually in the hard disk). Thus demand paging is the process that solves the above problem only by swapping the pages on Demand. This is also known as lazy swapper( It never swaps the page into the memory unless it is needed).

Swapper that deals with the individual pages of a process are referred to as Pager.

Demand Paging is a technique in which a page is usually brought into the main memory only when it is needed or demanded by the CPU. Initially, only those pages are loaded that are required by the process immediately. Those pages that are never accessed are thus never loaded into the physical memory.

(i) Advantages of Demand Paging

The benefits of using the Demand Paging technique are as follows:

* With the help of Demand Paging, memory is utilized efficiently.
* Demand paging avoids External Fragmentation.
* Less Input/Output is needed for Demand Paging.
* This process is not constrained by the size of physical memory.

(ii) Disadvantages of Demand paging

Drawbacks of Demand Paging are as follows:

* There is an increase in overheads due to interrupts and page tables.
* Memory access time in demand paging is longer.

24. Page Replacement / Page Replacement Algorithm

In Virtual Memory Management, Page Replacement Algorithms play an important role. The main objective of all the Page replacement policies is to decrease the maximum number of page faults.

Page Fault – It is basically a memory error, and it occurs when the current programs attempt to access the memory page for mapping into virtual address space, but it is unable to load into the physical memory then this is referred to as Page fault.

* 1. FIFO Page Replacement Algorithm

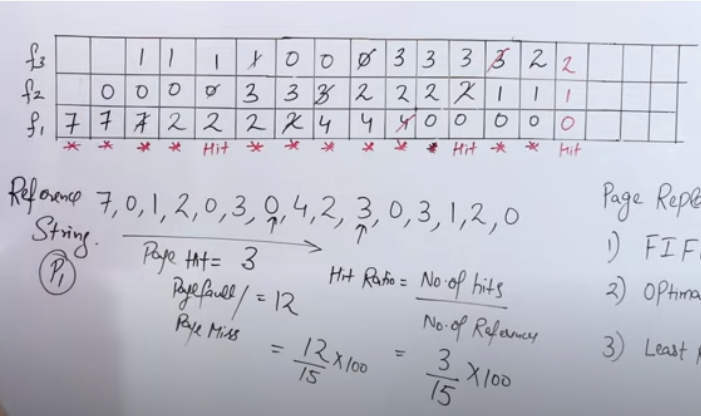
It is a very simple way of Page replacement and is referred to as First in First Out. This algorithm mainly replaces the oldest page that has been present in the main memory for the longest time.

This algorithm is implemented by keeping the track of all the pages in the queue.

As new pages are requested and are swapped in, they are added to the tail of a queue and the page which is at the head becomes the victim.

This is not an effective way of page replacement but it can be used for small systems.

* Advantages
* This algorithm is simple and easy to use.
* FIFO does not cause more overhead.
  + - * Disadvantages
* This algorithm does not make the use of the frequency of last used time rather it just replaces the Oldest Page.
* There is an increase in page faults as page frames increases.
* The performance of this algorithm is the worst.



(ii) LIFO Page Replacement Algorithm

This Page Replacement algorithm stands for "Last In First Out".This algorithm works in a similar way to the LIFO principle.

In this, the newest page is replaced which is arrived at last in the primary memory

This algorithm makes use of the stack for monitoring all the pages.

(iii) LRU Page Replacement Algorithm in OS

This algorithm stands for "Least recent used" and this algorithm helps the Operating system to search those pages that are used over a short duration of time frame.

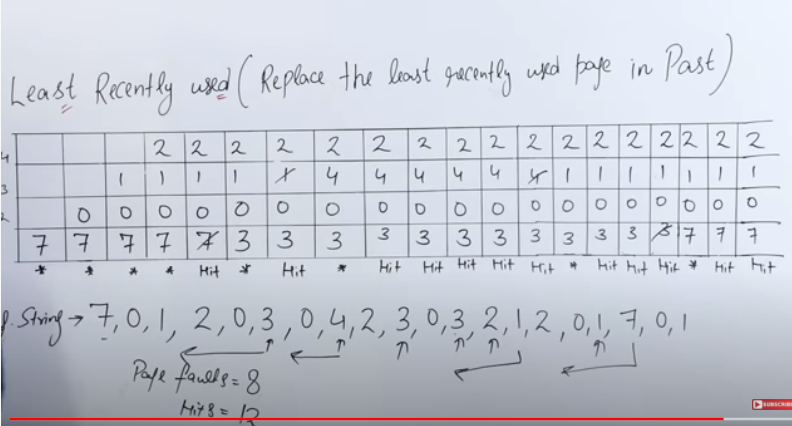
The page that has not been used for the longest time in the main memory will be selected for replacement.

This algorithm is easy to implement.

This algorithm makes use of the counter along with the even-page.

* Advantages of LRU
* It is an efficient technique.
* With this algorithm, it becomes easy to identify the faulty pages that are not needed for a long time.
* It helps in Full analysis.
  + - * Disadvantages of LRU
* It is expensive and has more complexity.

- There is a need for an additional data structure.



(iv) Optimal Page Replacement Algorithm

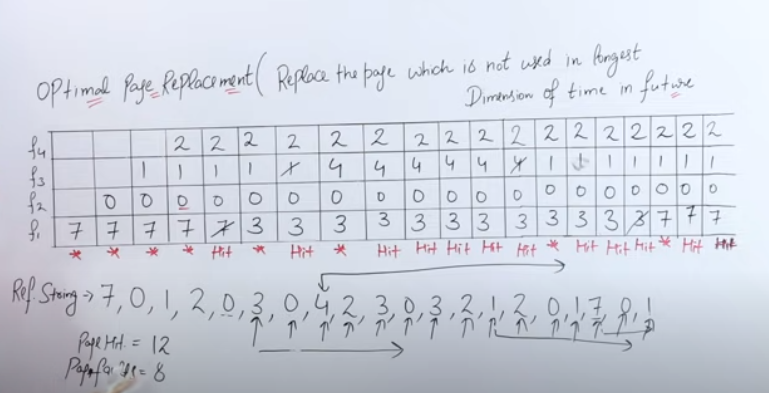
This algorithm mainly replaces the page that will not be used for the longest time in the future. The practical implementation of this algorithm is not possible.

Practical implementation is not possible because we cannot predict in advance those pages that will not be used for the longest time in the future.

This algorithm leads to less number of page faults and thus is the best-known algorithm

Also, this algorithm can be used to measure the performance of other algorithms.

* Advantages of OPR
* This algorithm is easy to use.
* This algorithm provides excellent efficiency and is less complex.
* For the best result, the implementation of data structures is very easy
  + - * Disadvantages of OPR
* In this algorithm future awareness of the program is needed.
* Practical Implementation is not possible because the operating system is unable to track the future request



(v) Random Page Replacement Algorithm

As indicated from the name this algorithm replaces the page randomly. This Algorithm can work like any other page replacement algorithm that is LIFO, FIFO, Optimal, and LRU.

25. Secondary Storage

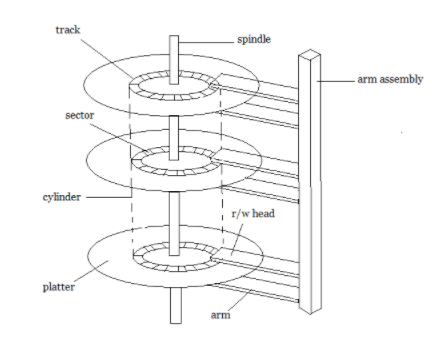
Secondary storage devices are those devices whose memory is non volatile, meaning, the stored data will be intact even if the system is turned off. Here are a few things worth noting about secondary storage.

Secondary storage is also called auxiliary storage.

Secondary storage is less expensive when compared to primary memory like RAMs.

The speed of the secondary storage is also lesser than that of primary storage.

* 1. Disk Structure



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

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

The speed of the disk is measured as two parts:

Transfer rate: This is the rate at which the data moves from disk to the computer.

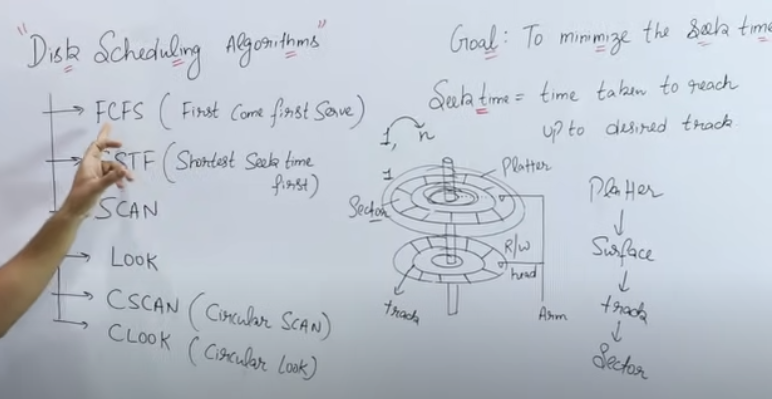
Random access time: It is the sum of the seek time and rotational latency.

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

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

* 1. Disk Scheduling Algorithms

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

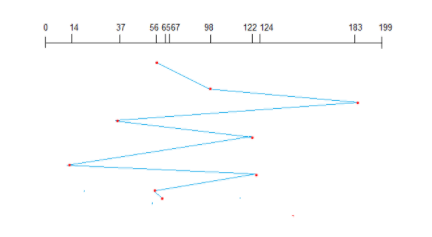


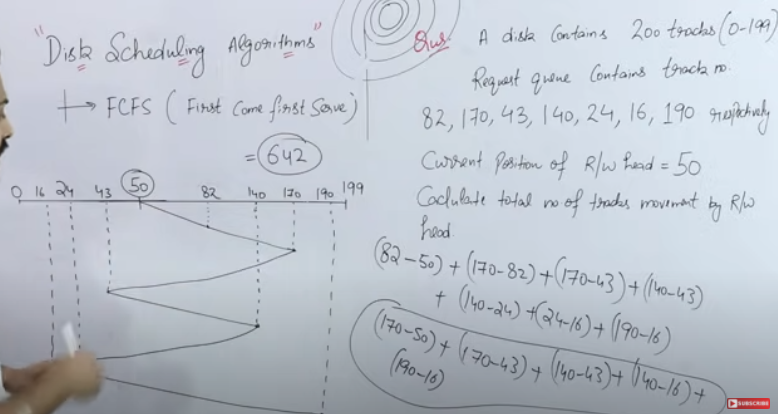
* First Come First Serve

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

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

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

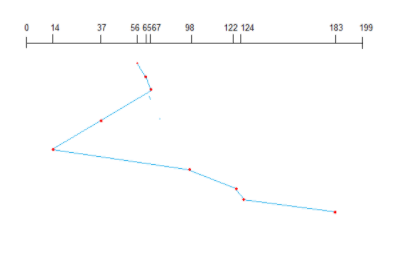


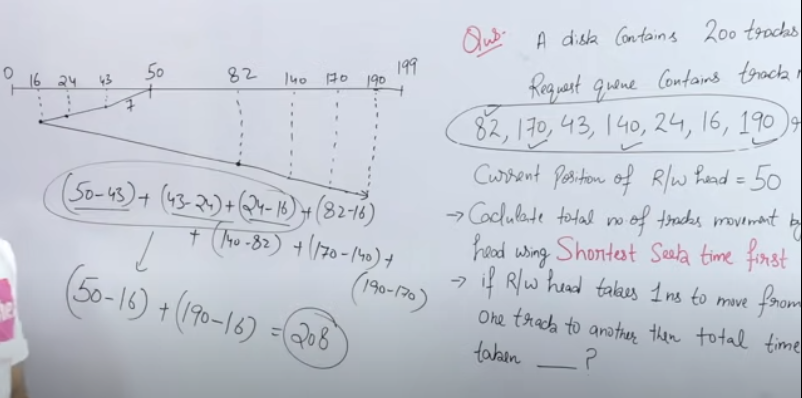


* Shortest Seek Time First (SSTF)

Here the position which is closest to the current head position is chosen first. Consider the previous example where disk queue looks like, 98, 183, 37, 122, 14, 124, 65, 67

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

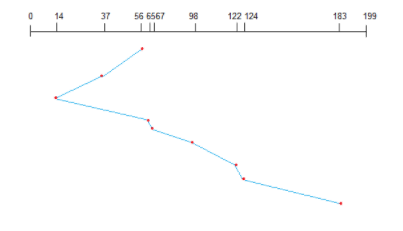




* SCAN algorithm

This algorithm is also called the elevator algorithm because of it's behavior. Here, first the head moves in a direction (say backward) and covers all the requests in the path. Then it moves in the opposite direction and covers the remaining requests in the path. This behavior is similar to that of an elevator. Let's take the previous example, 98, 183, 37, 122, 14, 124, 65, 67

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



26. File System

A file can be "free formed", indexed or structured collection of related bytes having meaning only to the one who created it. Or in other words an entry in a directory is the file. The file may have attributes like name, creator, date, type, permissions etc.

* 1. File Structure

A file has various kinds of structure. Some of them can be :

* Simple Record Structure with lines of fixed or variable lengths.
* Complex Structures like formatted document or reloadable load files.
* No Definite Structure like sequence of words and bytes etc.

(ii)Attributes of a File

Following are some of the attributes of a file :

* Name . It is the only information which is in human-readable form.
* Identifier. The file is identified by a unique tag(number) within file system.
* Type. It is needed for systems that support different types of files.
* Location. Pointer to file location on device.
* Size. The current size of the file.
* Protection. This controls and assigns the power of reading, writing, executing.
* Time, date, and user identification. This is the data for protection, security, and usage monitoring.

(iii) 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.

* 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.

\* 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 dobe in any order.

- User now says "read n" rather than "read next".

- "n" is a number relative to the beginning of file, not relative to an absolute physical disk location.

\* Indexed Sequential Access

- It is built on top of Sequential access.

- It uses an Index to control the pointer while accessing files.

(i) What is a 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.