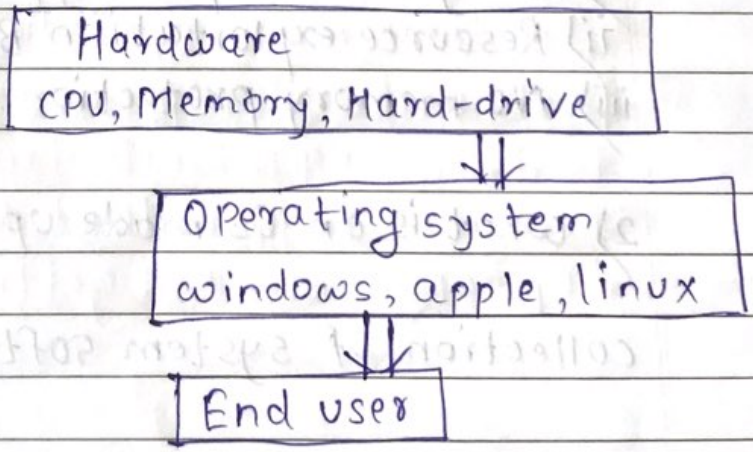
classmate
Date 17/8/22
Page

→ OPERATING SYSTEM

An operating system is a software that acts as an interface between computer hardware components and the user. Every computer system must have atleast one operating system to run other programs. Applications like Browser, Ms Office, Notepad, Games, etc, need some environment to run and perform its task.

The OS helps you to communicate with the computer without knowing how to speak the computer's language. It is not possible for the user to use any computer or mobile device without having an operating system



Another defination:-

An operating system is a piece of software that manages all the resource of a computer system, both hardware and software, and provides an environment in which the user can execute his/her programs in a convenient and efficient manner by hiding underlying complexity of hardware and acting as resource manager.

→ Why OS ?

1) what if there is no OS ?

i) Bulky and complex App.
ii) Resource exploitation By 1 app.
iii) No memory protection.

2) what is an OS made up of ?

collection of system software

→ functions of OS:-

1) Convenience: An OS makes a computer more convenient to use.
2) Efficiency: An OS allows the computer system resources to be used efficiently.
3) Ability to Evolve: An OS should be constructed in such a way as to permit the effective development, testing and introduction of new system functions at the same time without interfering with service
4) Throughput: An OS should be constructed so that it can give maximum throughput (No. of tasks per unit time).

→ Types of OS:

* Goals of OS:-

1) Maximum CPU utilization.
ex: If there are 3 processes to be performed, so at first the first operation is performed and remaining 2 are waiting for accomplishment of first operation, and if process 1 has to perform some i/o tasks, so at that time process 2 will come in action, means the CPU should not remain idle at any time.

2) Process starvation.

If there are 3 operations to be performed and suppose operation 1 is taking infinite time, so it should not happen that the other 2 operations are not getting time.

3) High priority Execution :-

suppose if the cpu is performing some of it's task and at that if we plug some removable device into our system, so at that time our OS stops the process on which it was working and scans the device for antivirus because it is a high priority task.

→ Types :-

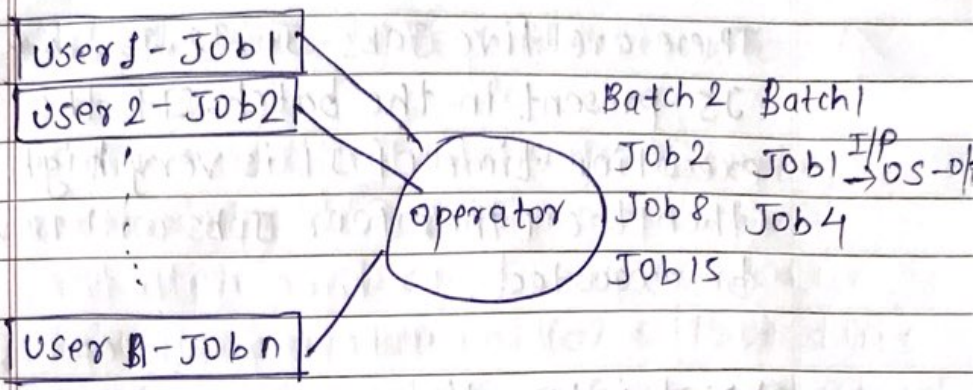
1) Single Process OS :-

In single process OS only one process is carried at a time. So in these type of OS, there is no maximum utilization of CPU, process starvation is there, and high priority operations are not performed untill a continuing process is finished.

ex: MS-DOS

2) Batch OS:

In Batch OS, the similar Jobs are grouped together into batches with the help of some operator and these batches are executed one by one. for example, let us assume that we have 10 programs that need to be executed. Some programs are written in C++, some in C and rest in Java. Now, every time when we run these programs individually then we will have to load the compiler of that particular language and then execute the code. But what if we make a batch of these 10 programs. The benefit with this approach is that, for the C++ batch you need to load the compiler only once and similarly for Java and C, the compiler needs to be loaded only once and the whole batch gets executed.



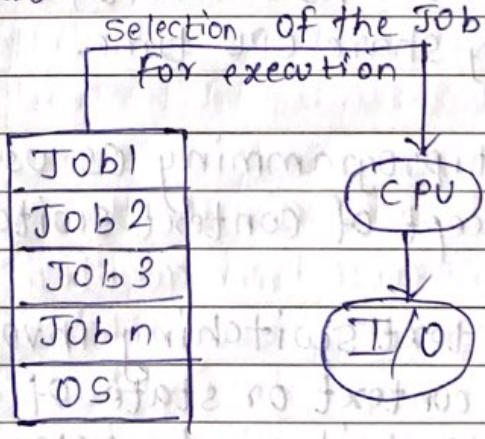
→ Advantages:-

1) The Overall time taken by the system to execute all the programmes will be reduced.

2) The Batch OS can be shared between multiple users.

→ Disadvantages:-

1) Starvation



There are five Jobs J_1, J_2, J_3, J_4 and J_5 present in the batch. If the execution time of J_1 is very high, then the other four Jobs will never be executed.

2) Not interactive.

3) Multiprogramming Os :-

A multiprogramming os may run many programs on a single processor computer. If one program must wait for an input/output transfer in a multiprogramming Os, the other programs are ready to use the CPU. As a result, various Jobs may share CPU time.

→ Multiprogramming Os uses the concept of context switching:

context switching involves storing the context or state of process so that it can be reloaded when required and execution can be

resumed from the same point as earlier.

This allows a single CPU to be shared by multiple processess.

Ex: Assume that initially Process1 is running. Process1 is switched out and Process2 is switched in because of an interrupt or a system call. context switching involves saving the state of process1 into PCB1 and loading the state of Process2 from PCB2. After sometime again a context switch occurs and Process2 is switched out and Process1 is switched in again. This involves saving the state of Process2 and loading the state of Process1 from PCB1.

PCB ⟹ Process Control Block.

4) Multitasking OS :-

In Multitasking OS, more than one processes are being executed at a particular time with the help of time-sharing concept. So in time sharing environment, we decide a time that is called time quantum and when the process starts its execution then the execution continous for only that amount of time and after that other processes will be given chance for that amount of time only. In the next cycle, the first process will again come for its execution and it will be executed for that time quantum only.

→ Advantages:

1) since equal time quantum is given to each process, so each process gets equal opportunity to execute.
2) The CPU will not be idle at any time.

Disadvantages:

Process having higher priority will not get the chance to be executed first because the equal opportunity is given to each process.

5) Multiprocessing OS:

In operating system, to improve the performance of more than one CPU can be used within one computer system called multiprocessing OS.

Multiple CPUs are interconnected so that a job can be divided among them for faster execution. when a job finishes results from all CPU's are collected and compiled to give the final o/p. Jobs need to share main memory and they may also share other system resources among themselves.

Ex: UNIX OS.

In Multiprocessing, Parallel computing is achieved. More than one processor is present in the system can execute more than one process simultaneously, which will increase throughput of the system.

→ Advantages:-

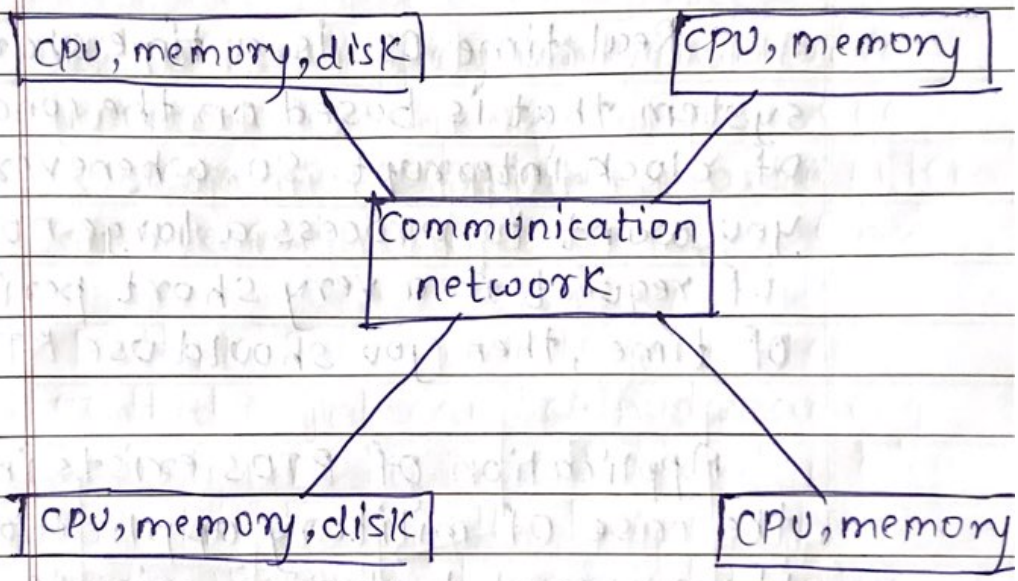
1) Increased reliability: Due to multiprocessing system, processing tasks can be distributed among several processors. If one processor fails, the task can be given to another

2) Increased throughput.

6) Distributed OS:

In distributed OS, we have various systems and all these system have their own CPU, main memory secondary memory and resources. These systems are connected to

each other using a shared communicatio network. Here, each system can perform it task individually. The best part about these distributed OS is remote access i.e one user can access the data of other system and can work accordingly. so remote access is possible in these system.



Advantages:

1) failure of one system can't stop the execution
2) Resources are shared between each other
3) load on host computer gets distributed

7) Real Time OS [RTOS]:-

RTOS are used in the situation where we are dealing with some real time data. So, as soon as the data comes, the execution of the process should be done and there should be no delay i.e no buffer delays should be there.

Real time OS is a time sharing system that is based on the concept of clock interrupt. So, whenever you want to process a large no. of request in a very short period of time, then you should use RTOS.

Application of RTOS exists in the case of military applications if you want to drop a missile, the missile should be drop with certain precision.

→ Process in OS :-

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

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

When a program is loaded into the memory and it becomes a process.

→ Thread : light weight Process

A process is a complete program to be execute. So, a thread is a sub-part or say a sub-process within that process.

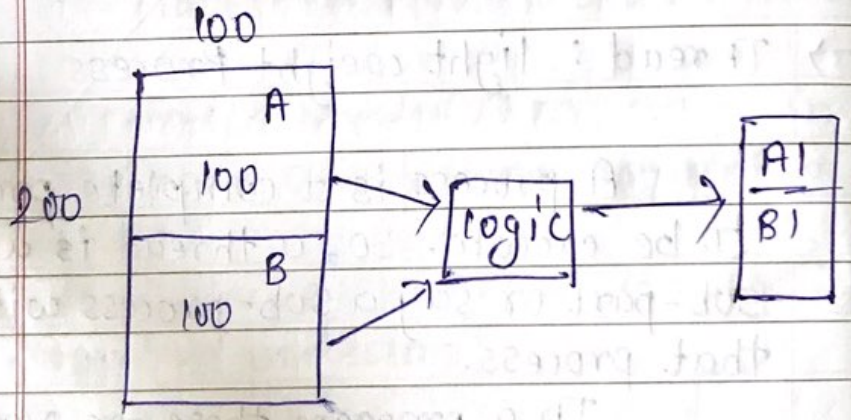
In a program there are various things to be done. for ex, if there is a software and it takes user input, and displays it on the screen, and also uploads the user input to

cloud, so here uploading the user input to cloud can be done using a thread, because it is an independent process.

So thread is a sub-process which can be run independently

for ex:

If we have to convert an JPG img to PNG img of 100x100 pixels but the height of image is 200 so here we break down the image into 2 parts.



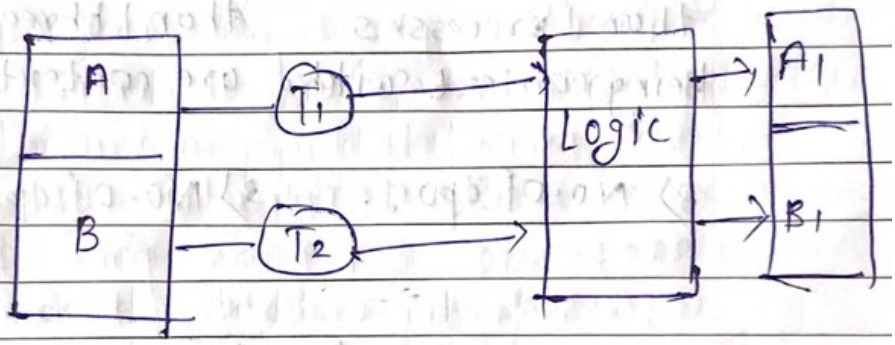
JPG

So the A part of image is given first to program logic and then B part and the both the O/p are concatenated.

so here we are doing sequential processing.

Another way of doing the same process is by using the concept of Multithreading.

So here, the A part of img is given to T_1 thread and B part to T_2 thread.



So here both parts of img are processed simultaneously. So it takes less amount of time as compared to ~~simultaneous~~ sequentially processing of img.

Multitasking	Multithreading
1) The execution of more than one task simultaneously is called as multitasking	1) A process is divided into several different sub tasks called as threads which has its own execution. This concept is called as multithreading
2) concept of more than 1 processes being context switched	2) concept of more than 1 thread. Threads are context switched.
3) No. of cpu 1.	3) No. of cpu >= 1.
4) Isolation and memory protection	4) No isolation and memory protection.

→ Thread Scheduling :

Threads are scheduled for execution based on their priority. Even though threads are executing within the runtime, all threads are assigned

processor time slices by the operating system.

Thread context - switching	Process - context switching
1) OS saves current state of thread and switches to another thread of same process	1) OS saves current state of process and switches to another process by restoring its state
2) Doesn't include switching of memory address space.	2) Includes switching of memory address space.
3) fast switching.	3) slow switching.
4) CPU's cache state is preserved.	4) CPU's cache state is flushed.

CLASSMATE
Date
Page

→ Components of OS :-

1) User-space
2) Kernel

1) User-space :-

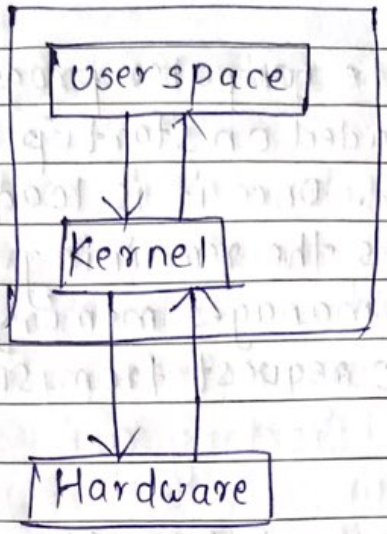
User space is a part of computer operating system virtual memory. All user applications, work and programs are stored in computer's user space.

where application software runs, apps don't have privileged access to the underlying hardware. It interacts with kernel.

a) GUI
b) CLI

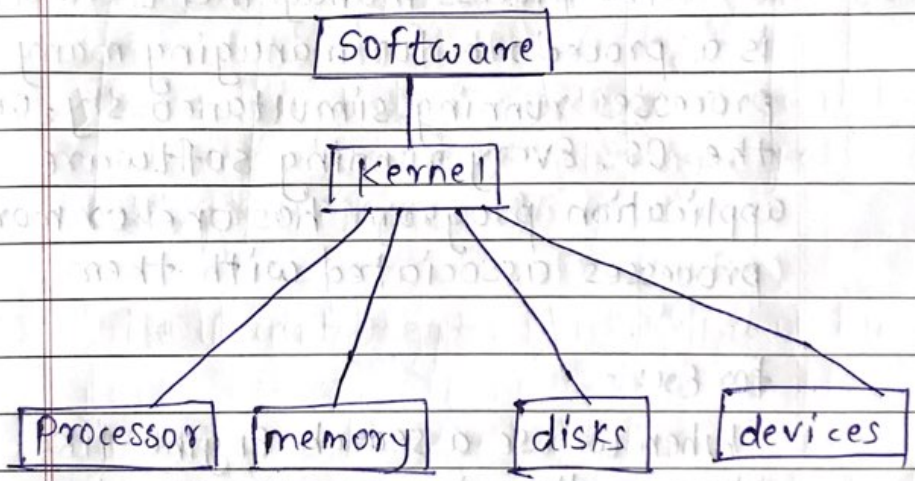
2) Kernel space :-

It is also called as heart of OS. The commands which are given by user space is accessed by kernel space and then it directs the hardware to do a specific job. It directly interacts with hardware, ~~space~~.



OS

Userspace interacts with Kernel and further Kernel interacts with hardware.



i) Kernel is the core part of OS, hence it has full control over everything in system. Each operation of hardware and software is managed and administered by Kernel.

2) It is the computer program that first loaded on startup of the system. Once it is loaded, it manages the remaining start-ups. It also manages memory, peripheral, and I/O request from software.

→ function of kernel :-

1) Process Management:

The process management component is a procedure for managing many processes running simultaneously, on the OS. Every running software application program has one or more processes associated with them.

for Ex :-
When we use a search Engine like Chrome, there is a process running for that browser program.

Process management keeps processes running effectively. It also uses memory allocated to them and shutting them down when needed.

functions of Process management

1) Process creation and deletion.
2) suspension and resumption.
3) synchronization process.
4) communication process.
5) scheduling processes and threads on the CPU.

2) Memory Management

1) Allocating and deallocating memory space as per need.
2) Keeping track of which part of memory are currently being used and by which process.

3) File Management

1) creating and deleting files.
2) creating and deleting directories to organize files.
3) Mapping files into secondary storage
4) Backup support onto a stable storage media.

4) I/O management :

To manage and control I/O operations and I/O devices

a) Buffering (data copy between two devices), caching and spooling.

i) Spooling :

1) within differing speed two Jobs.

⇒

ii) Buffering :

1) within one Job

iii) caching :

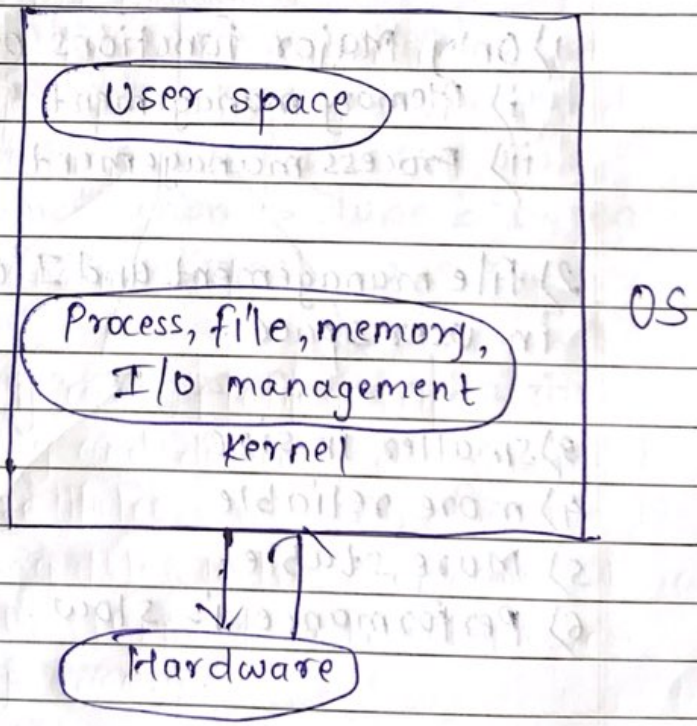
1) Memory caching, Web caching, etc.

classmate
Date
Page

Types of Kernel:-

1) Monolithic Kernel:-

All the function of kernel are within itself. Both user services and the kernel services are implemented in the same memory space. By doing this the size of kernel increases but at the same time it increases the speed size of OS. The execution of process will be faster as there is no seperate user space and kernel space.



→ Advantage:

1) Fast communication

→ Disadvantage

1) less reliable.

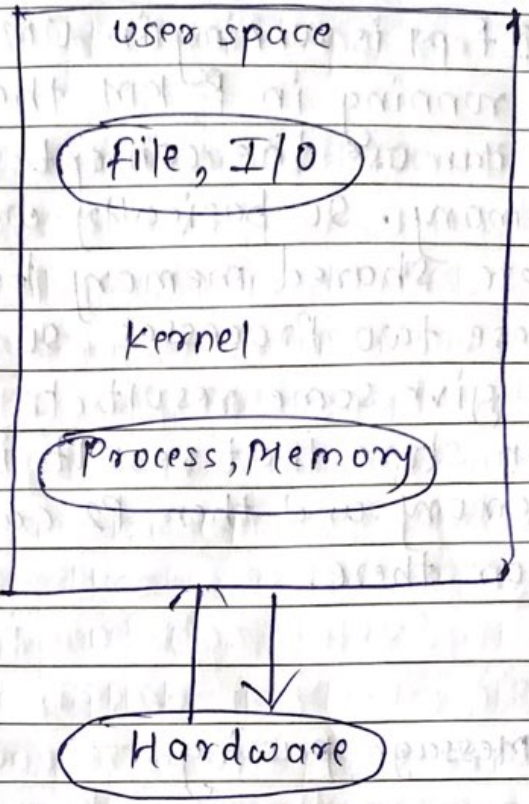
Ex: Linux, Unix and MS-DOS.

2) Micro Kernel:-

1) Only Major functions are in Kernel
   i) Memory management
   ii) Process management

2) file management and I/O are placed in user space.

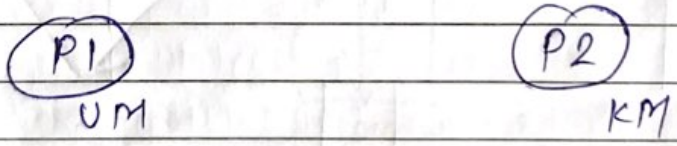
3) smaller in size
4) more reliable
5) More stable.
6) Performance is slow.



→ Interview Question:

How communication is done between user mode and Kernel mode ?

→ IPC (inter Process communication)



1) Shared memory.

If p1 is running in UM and P2 is running in KM then ~~there is~~ we can use the concept of shared memory. So basically we can use these shared memory between these two Processes, so if P1 has to give some result to P2 so it can store that result in shared memory and then P2 can get it from there.

2) Message passing,

Here we can establish a logical connection between these two process by using system call.

3) Hybrid kernel :-

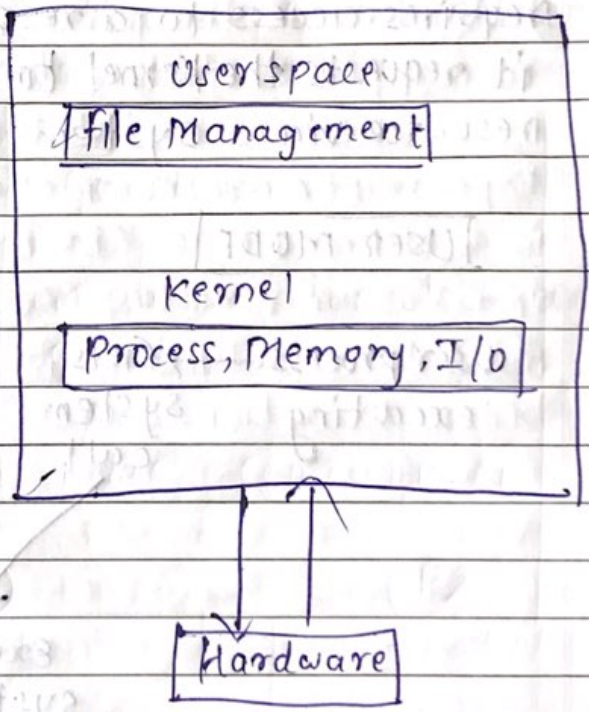
1) combination of Monolithic and Micro Kernel.

2) file management in user space and and rest in kernel.

3) Speed and design of Monolithic

4) Modularity and stability of micro.

Ex: MacOs, windows NT/7/10;

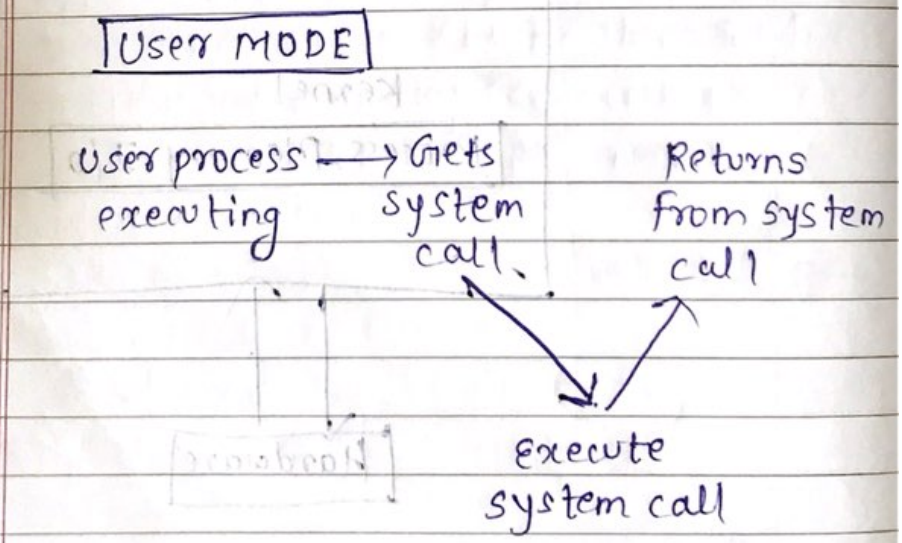


classmate
Date
Page

System calls in OS :-

The interface between a process and an OS is provided by system call. In general system calls are available as assembly language instructions. They are also included in the manuals used by assembly level programmers.

System calls are usually made when a process in user mode requires access to a resource. Then it requests the Kernel to provide the resource via a system call.



As can be seen from diagram, the processes execute normally in the user mode untill a system call interrupts this. Then the system call is executed on a priority basis in the kernel mode. After the execution of the system call, the control returns to the user mode and execution of user processes can be resumed

1) If a file system requires the creation or deletion of files. Reading and writing from files also require a system call.
2) creation and management of new process.
3) Network connection also requires system call.

classmate

Date
Page

→ User mode to kernel mode switching?

After the execution of process, a software interrupt is called.

↓

Telling the CPU to stop it's ongoing work midway and do another work.

→ How operating system Boots up?

⇨ Booting is the process of starting a computer. It can be initiated by hardware such as button press or by a software command. After it is switched on, a CPU has no software in its main memory, so some processes must load software into memory before execution.

→ An overview of Boot process

Power up / Reset	system startup	BIOS / Boot monitor
	~~stg~~ stage 1 Bootloader	Master Boot record
	Stage 2 Bootloader	LILO, GRUB. etc
	KERNEL	Linux
	Init	User-space

The boot process is something that happens every time you turn your computer on. You don't really see it, because it happens so fast.

The Bios chips tells it to look in a fixed place, usually on the lowest numbered hard disk (the boot disk) For a special program called a boot loader The boot loader is pulled in to memory and started. The boot loader job is to start the real OS.

→ Functions of BIOS :-

1) POST (power on self test) :-
POST happens each time you turn your computer on. It sounds complicated and that's because it kind of is.

1) It initializes various hardware devices.
2) It is an important process to ensure that all the devices operate smoothly without any conflicts. BIOSes following ACPI create tables describing the devices in the computer.
3) The POST first checks the bios and then tests the CMOS RAM.
4) If there is no problem with this then POST continues to check the CPU, hardware devices such as video card, and SSD, etc.
5) If some errors are found then an error message is displayed on screen.

2) Master Boot Record:

The MBR is a small program that starts when the computer is booting, in order to find the OS. This complicated process (called the Boot Process) starts with the POST and ends when the Bios searches for the MBR on the hard drive which is generally located in the first sector, first head, first cylinder.

The bootstrap loader is stored in the computer's EPROM, ROM, or another non-volatile memory. When the computer is turned on or restarted, it first performs the POST. If POST is successfull and no issues are found the bootstrap loader will load the OS for computer into memory.

3) init : init is the last step of the kernel boot sequence. It looks for the file /etc/init tab to see if there is an entry for init default. It is used to determine the initial run level of the system. A run-level is used to decide the initial state of the OS.

→ Difference between 32 bit Vs 64 bit Os:-

In computing there are two types of processors existing, i.e, 32 bit and 64 bit processors. These types of processors tell us how much memory a processor can access from a cpu register.

* A 32 bit system can access 2^{32} different memory addresses, i.e 4GB of RAM or physical memory ideally, it can access more than 4GB of RAM also.

A 64 bit system can access 2^{64} different memory addresses, i.e actually 18 Quintillion bytes of RAM.

→ Most early computers were 32 bit machines. The CPU register stores memory addresses, which is how the processor accesses data from RAM. One bit in the register can reference an individual byte in memory, so a 32 bit system can access ~~dd~~ address a maximum of 4GB of RAM.

A 64 bit register can theoretically reference 18,446,744,073,709,551,616 bytes of memory. The 64 bit computer can access more than 4 GB of RAM.

If a computer has 8GB of RAM, it better has a 64 bit processor.

→ Advantages of 64 bit over 32 bit

1) Using 64 bit one can do a lot in multi tasking, user can easily switch between various applications without any windows hanging problem.
2) Gamers can easily play High graphical games like Modern warfare, GTA V.

→ Types of storage:

Basically we want the programs and data to reside in main memory permanently.

This arrangement is usually not possible for following two reasons:-

1) Main memory is usually too small to store all needed programs and data permanently.
2) Main memory is a volatile storage device that loses it content when power is turned off or otherwise lost.

There are two types of storage devices:

1) Volatile : It loses it's contents when power is removed.
2) Non-Volatile : It does not loses its content when power is removed.

→ Secondary storage :-
It is used as an extension of main memory. secondary storage devices can hold the data permanently.

— Hierarchy :

- Registers
- cache
- Main Memory

} primary memory

- Electronic disk
- Magnetic disk
- Optical disk
- magnetic tapes

} secondary memory

Process :-

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

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

How OS creates a process?

1) Load the program and static data to memory (RAM).

2) Allocate Runtime stack.
part of memory used for local variable, function arguement and return value.

3) Allocate heap
part of memory used for dynamic allocation.

4) I/O tasks

5) OS handoffs control to main.

→ Architecture Of process:-

Stack	Local variables, function argument and return value
Heap	Dynamically allocated var.
Data	Global and static data
Text	compiled code

→ Program Vs Process?

A program is a set of instructions that we write on a file. And this file in its executable form is stored in hard disk.

A program becomes a process when it is being operated by the processor. That is a program becomes a process when its executable file is loaded into the main memory.

we can do these two ways:

1) Double clicking on executable file or program.
2) Entering the name of executable file on cmd.

Thus we call a process an active entity. A processor has a program counter that provides the address of the next instruction to be executed next. The process also has a set of resources for that particular process.

→ Types of errors:-

1) stackoverflow

2) out of memory: Deallocate the unnecessary objects.

→ Attributes of a process:-

P_1, P_2, P_3 and P_4 are the process, to be execute, so here the OS creates a process table and store these process in the table.

1	P_1
2	P_2
3	P_3
4	P_4

← Process table

Each entry of the process table is called as PCB (Process control Block).

→ PCB:

PCB stands for Process control Block. It is a data structure that is maintained by the OS for every process. The PCB should be identified by an integer process ID (PID). It helps you to store all the information required to keep track of all running processes.

→ PCB structure :-

Process ID
Program counter
process state
priority
Registers
List of open files
List of open devices

1) Process ID :

when a process is created a unique ID is assigned to the process which is used for unique identification of process.

2) Program counter:

A program counter stores the address of the last instruction of the process on which the process was suspended. The CPU uses this address when execution of process is resumed.

3) Process state :-

The process from its creation to ~~deletion~~ completion goes through various states which are new, ready, running and waiting.

4) Priority :

Every process has its own priority. The process with the highest priority among the processes gets the CPU first. This is also stored on PCB.

5) Registers

Every process has its own set of Registers which are used to hold the data which is generated during execution of process.

6) List of open files :

During execution, every process uses some files which need to be present in the main memory. OS also maintains a list of open files in PCB

7) List of open devices

OS, Also maintains a list of open devices during execution.

→ States of a process :-

1) New: Newly created process or being created process.

2) Ready: After creation process moves to ready state, i-e the process is ready for execution.

3) Run: currently running process in cpu

4) wait: when a process requests I/O access.

5) complete: The process completed its execution.

6) Suspended Ready: when the ready queue is full, some processes are moved to suspended ready state.

7) suspended block: when waiting queue becomes full.

→ Operations on process :-

1) Creation :-
Once the process is created, it will be ready and come into the ready queue (main memory) and will be ready for execution.

2) Scheduling :-
Out of many processes present in the ready queue, the OS chooses one process and start executing it. selecting the process which is to be executed next, is known as scheduling.

3) Execution:
Once the process is scheduled for execution, the processor starts executing it, Process may come to the blocked or wait state during execution then in that case the processor starts executing different processes.

4) Deletion:- purpose of the

Once the process gets over then the OS will kill the process. The context of the process (PCB) will be deleted and the process gets terminated by the OS.

→ Process Scheduler in OS :-

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

→ Process scheduling queue:

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

queue and moved to its new state queue.

Types of queue:-

1) Job queue:-
The queue keeps all the processes in the state.

2) Ready queue:-
These queue keeps a set of all the processes residing in main memory, ready and waiting to execute. A new process is always put in this queue.

→ Schedular:-

1) Job schedular:- picks process from the pool and loads them into memory for execution.

2) CPU schedular:- picks process from ready queue and dispatch it to CPU.

classmate
Date
Page

→ Swapping in OS :-

Swapping is a memory management scheme in which any process can be temporarily swapped from main memory to secondary memory so that the main memory can be made available for other processes. It is used to improve the main memory utilization.

→ The concept of swapping is divided into two core concepts,

* Swap in and swap out, is done by Medium term scheduler.
1) swap-out is a method of removing a process from RAM and adding it to Hard-disk.
2) swap in vice-versa of swap out.

→ Medium term Scheduler :-

Medium term scheduling is a part of swapping. It removes the processes from memory, and

it reduces the degree of multiprogramming. The medium term scheduler is in charge of handling the swapped ~~of~~ out processes.

Ex:-

If there are 4 processes i.e, P_1, P_2, P_3 and P_4 in ready queue.

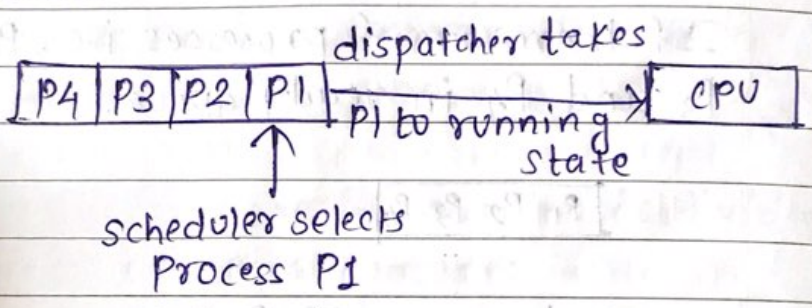
P_1	P_2	P_3	P_4

And suppose if P_1 takes more amount of space and P_3 and P_4 don't get space in ready queue, so at that time we swap-out P_3 and P_4 from ready queue with the help of MTS and place them in Secondary storage and after the termination of P_1 we swap-in both the process i.e, P_3 and P_4, in ready queue.

* In secondary memory, the place where the swapped out process is stored is called swap space.

→ Example of scheduler and dispatcher:-

Suppose four processes, P_1, P_2, P_3 and P_4 are in ready queue. Their arrival times are T_1, T_2, T_3 and T_4 respectively. And a FIFO algorithm is used in this whole process or task.



The Process P1 arrives first, so the scheduler will decide it is the first process to be executed, and the dispatcher will remove P1 from ready queue and give it to the CPU, and so on.

classmate
Date
Page

→ Context switching :-

Context switching involves storing the context or state of a process so that it can be reloaded when required and execution can be resumed from the same point as earlier. This is a feature of multitasking OS and allows a single CPU to be shared by multiple processes.

for ex :-

Initially P1 is running. P1 is switched out and P2 is switched in because of an interrupt or an system call. Context switching involves saving the state of P1 into PCB1 and loading the state of P2 from PCB2, and vice versa.

→ ORPHAN Process:-

1) The process whose parent process has been terminated
2) Orphan processes are adopted by init process
3) Init is the first process of CPU.
4) Kernel detects an orphan process in OS and assigns a new parent process.

→ Zombie Process:-

Zombie process is also known as dead process. Ideally when a process completes its execution, it's entry from the process table should be removed but this does not happen in case of zombie process.

Zombie process usually occur for child processes, as the parent process still needs to read its child exit status. Once this is done

Using the wait system call, the Zombie process is eliminated from process table. This is known as reaping the zombie process.

It is because parent process may call wait() on child process for a longer time duration and child process got terminated much earlier.

As entry in the process table can only be removed after the parent process reads the exit status of child process. Hence, child process remains a zombie till it is removed from process table.

Scheduling Algorithm:-

1) Non-preemptive Scheduling:

Non-preemptive scheduling is used when a process terminates, or a process switches from running to waiting state. In this scheduling, once the resources are allocated to a process, the process holds the CPU till it gets terminated or reaches a waiting state. In this case of non-preemptive does not interrupt a process running CPU in the middle of execution.

2) Preemptive Scheduling:

*) Preemptive scheduling is used when a process switches from running state to ready state or from waiting state to ready state.
2) CPU is taken away from a process after time quantum expires along with terminating or switching to wait state.

classmate
Date
Page

→ Difference :

1) Starvation : Non-preemptive ↑
preemptive ↓

2) CPU utilization : Preemptive ↑

3) Overhead : Preemptive ↑

→ Goals Of CPU scheduling :

a) Maximum CPU utilization
b) Minimum Turn around time
c) Minimum wait time.
d) Minimum Response time
e) Maximum throughput of system

→ Terms :-

1) Throughput : No. of processes completed per unit time.

2) Arrival time : Time when process is (AT) arrived at ready queue.

3) Burst : The time required by the (BT) process for its execution.

4) Turnaround time : Time taken from (TAT) first time process enters ready State till it terminates. (CT - AT)

5) wait : Time process ~~wait~~ spends (WT) waiting for CPU. (TAT - BT)

6) Response time : Time duration between process getting in to ready queue and process getting CPU for the first time.

7) Completion : Time taken till process (CT) gets terminated.

→ Scheduling Algorithm :-

1) First come first serve [FCFS] :-

FCFS scheduling algorithm simply schedules the Jobs according to their arrival time. The Job which comes first in the ready queue will get the CPU first. The lesser the arrival time of Job, the sooner will the Job get the CPU. FCFS scheduling may cause the problem of starvation if the burst time of first process is the longest among all jobs.

→ Advantage :

1) Simple
2) easy
3) First come, first serve.

→ Disadvantage :

1) Starvation

classmate
Date
Page

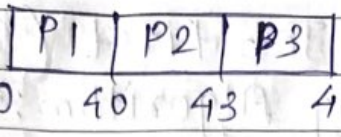
→ Convoy effect in FCFS:-

FCFS may suffer from convoy effect if the burst time of first Job is the highest among all.

If the CPU gets the process of the higher burst time at the front of the ready queue then the processes of lower burst time get blocked which means they may never get the CPU if the Job in execution has a very high burst time. This is called convoy effect or starvation.

Ex: We have 3 processes named P_1, P_2 and P_3. The Burst time of P1 is highest.

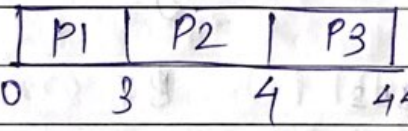
Process ID	AT	BT	CT	TAT	WT
1	0	40	40	40	0
2	1	3	43	42	39
3	1	1	44	43	42



Avg waiting time = 81/3

→ In second scenario if P1 has arrived at last of queue and P2 and P3 at earlier then problem of starvation will not be there.

Process ID	AT	BT	CT	TAT	WT
1	1	40	44	43	3
2	0	3	3	3	0
3	0	1	4	4	3



Avg Waiting time = 6/3

2) Shortest Job first (SJF) scheduling Algorithm:-

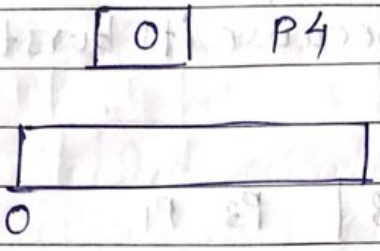
SJF is an algorithm in which the process having the smallest Burst time is choosen for the next execution. This scheduling method can be preemptive or non-preemptive.

→ Non-preemptive SJF:-

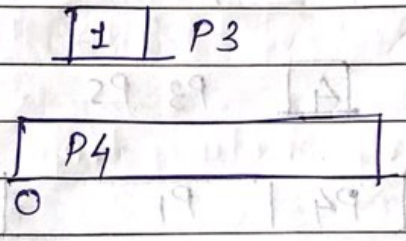
In non-preemptive scheduling once the CPU cycle is allocated to a process, the process holds it till it reaches a waiting state or terminated.

Processes	BT	AT
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4

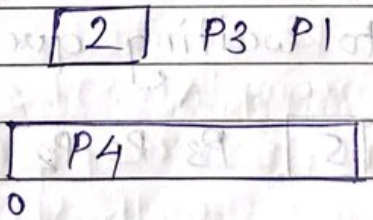
Step 0) At t=0, P4 arrives and starts execution.



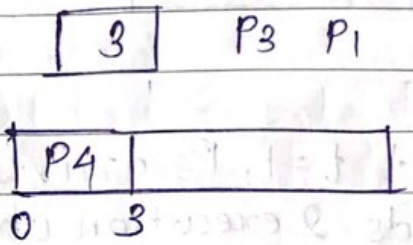
Step 1) At t=1, P3 arrives. But P4, still needs 2 execution units to complete.



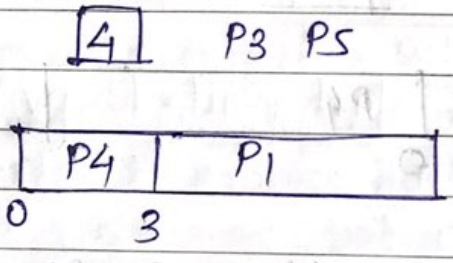
Step 2) At, time = 2, P1 arrives.



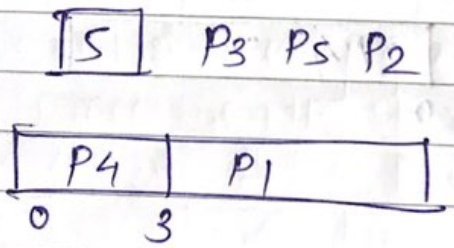
Step 3) At t=3, P4 will finish its execution. The burst time of P3 and P1 is compared. Process P1 is executed because its burst time is less.



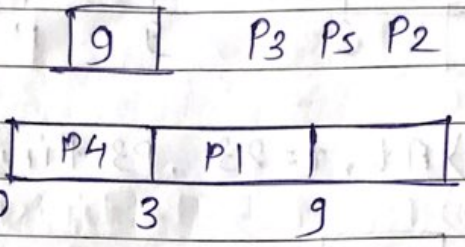
Step 4) At t=4, P5 arrives



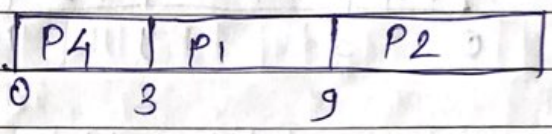
Step 5) At t=5, P2 arrives and added to waiting queue.



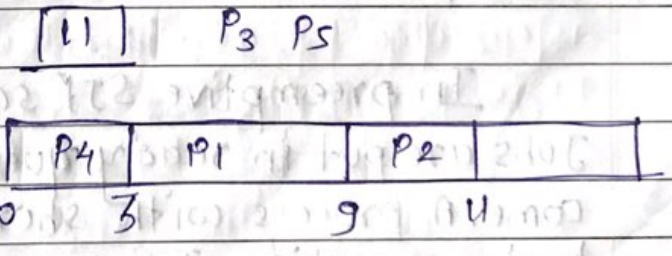
Step 6) At t = 9, P1 will execute. Burst time P3, P5 and P2 will be compared. P2 is executed because of less BT.



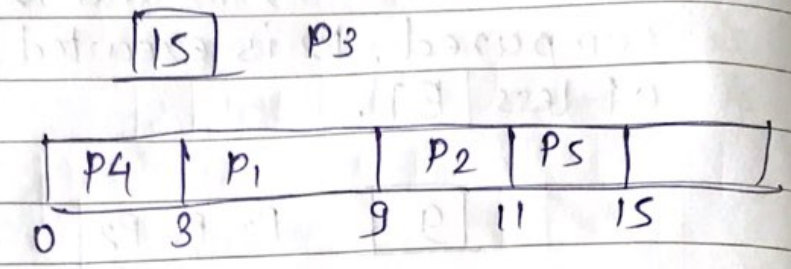
Step 7) At, t = 10, P2 is executing



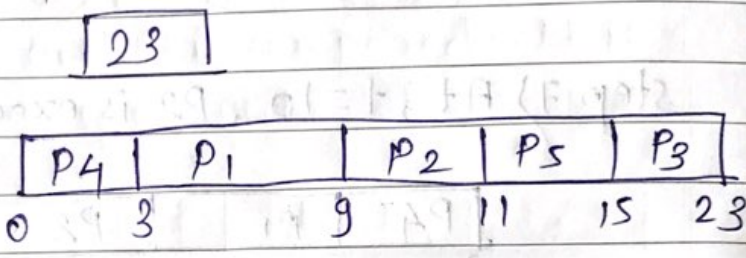
Step 8) At, t = 11, P2 finishes. P3 and P5 are compared. P5 is executed because of less BT.



step 9) At, t=15, P5 finishes



Step 10) At, t=23, P3 finishes.

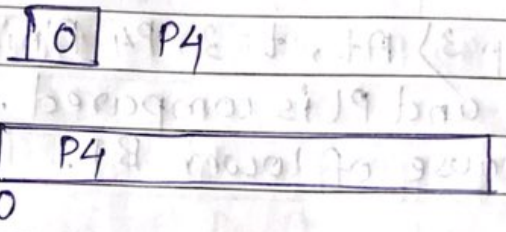


Avg waiting time = 26/5
= 5.2

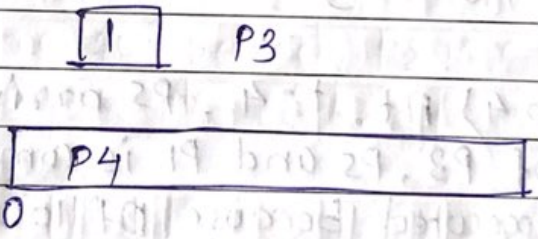
→ Preemptive SJF:

In preemptive SJF scheduling, Jobs are put in ready queue as they come. A process with shortest BT begins execution. If a process with even a shorter burst time arrives, the current process is removed from execution and shorter Jobs is allocated cycle.

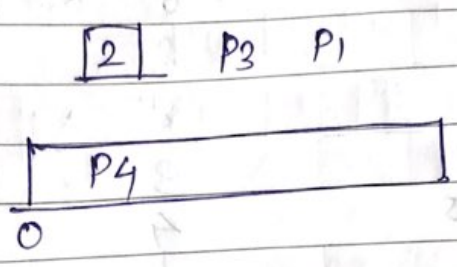
Processes	BT	AT
P1	6	2
P2	2	5
P3	8	1
P4	3	0
P5	4	4



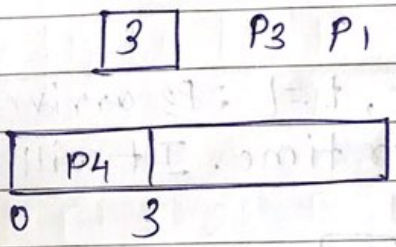
Step 1) At, t=1, P3 arrives. But P4 has shorter time. It will continue.



Step 2) At t=2, P1 arrives but BT=6, P4 < P1.

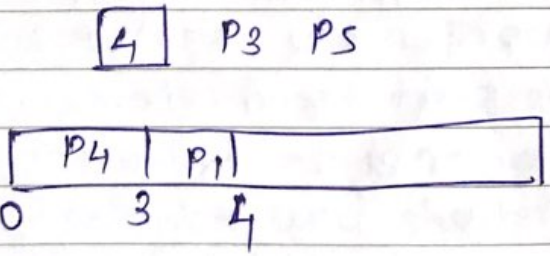


Step 3) At, t=3 P4 finishes. BT of P3 and P1 is compared. P1 is executed because of lower BT.



Step 4) At, t=4, P5 arrives. BT of P3, P5 and P1 is compared. P5 is executed because of lower BT. P1 is preempted.

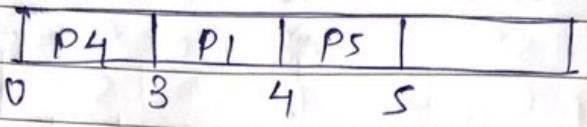
~~AT~~ BT of P1 = 6-1 = 5



Step 5) At, t=5, P2 arrive. P2 is executed.

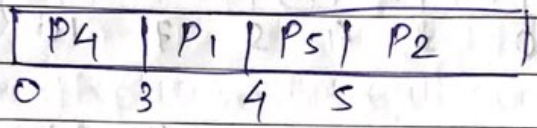
BT of P5 = 4-1=3

| 5 | P1 | P3 | P2 |



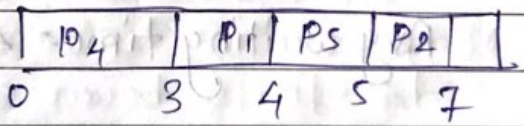
Step 6) At, t= 6. P2 is executing

| 6 | P1 | P3 | P5 |



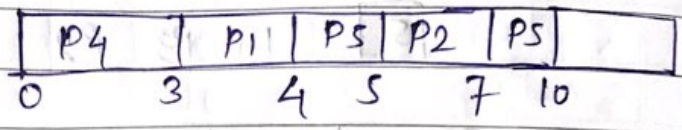
Step 7) At, t=7, P2 finishes. P1, P3 and P5 compared. P5 is executed.

| 7 | P1 | P3 | P5 |



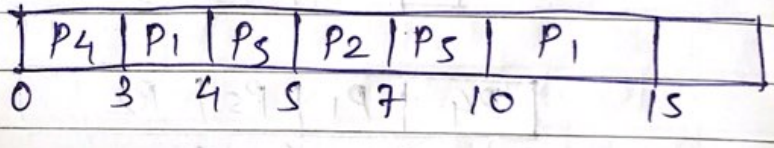
Step 8) At, t=10 P5 finishes. BT of P1 and P3 is compared. P1 is executed.

10 | P1 | P3



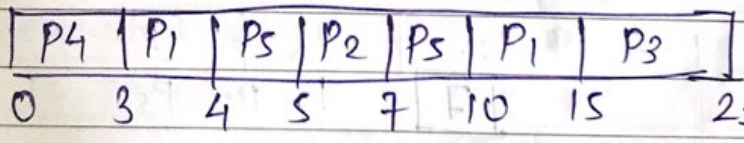
Step 9) At, t=15. P1 finishes.

15 | P3



Step 10) At, t=23 P3 finishes

23



Avg waiting time = 4.6

-3) Priority Scheduling Algorithm :-

Priority scheduling is a method of scheduling processes that is based on priority. In this algorithm, the scheduler selects the tasks to work as per the priority.

→ Types of priority scheduling

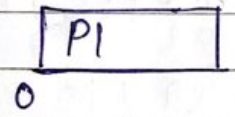
i) Preemptive scheduling.
ii) Non-preemptive scheduling.

1) Preemptive scheduling :

In preemptive scheduling the tasks are mostly assigned with their priorities. Sometimes it is important to run a task with a higher priority before another lower priority task, if the lower priority task is still running. The lower priority task holds for some time and resumes when higher priority task finishes.

Process no	Priority	AT	BT
1	2	0	4 → 3
2	4	1	2 → 1
3	6	2	3 → 2
4	10	3	5 → 3
5	8	4	1
6	12	5	4
7	9	6	6

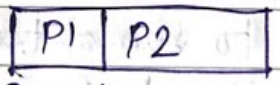
→ At, t = 0



→ At, t = 1

P2 starts, because of higher priority, P1 is preempted.

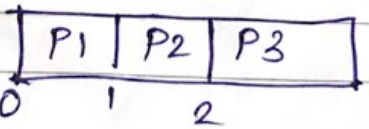
remaining time of P1 = 4 - 1 = 3



→ At, t = 2

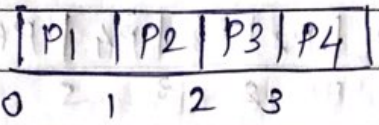
P3 starts, because of higher priority, P2 is preempted.

remaining time of P2 = 2 - 1 = 1

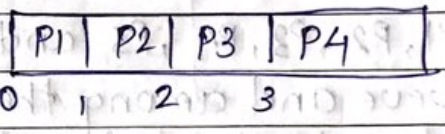


classmate
Date
Page

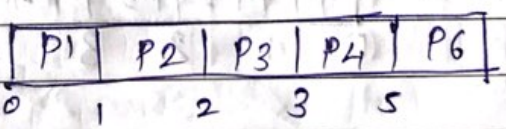
→ At, t=3
P4 starts, P3 is preempted
P3 = 3-1=2



→ At, t=4
P5 arrives, but P4 has higher Priority so P4 continues.

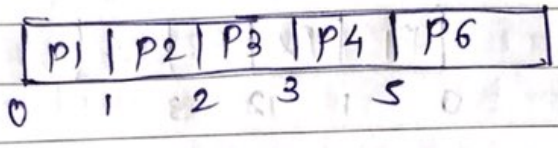


→ At, t=5
P6 arrives, P6 has higher Priority so P6 continues.
P4 = 5-2=3



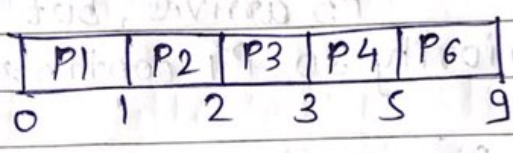
→ At, t = 6

P7 arrives, but P6 has higher priority so it continues.



→ At, t = 9

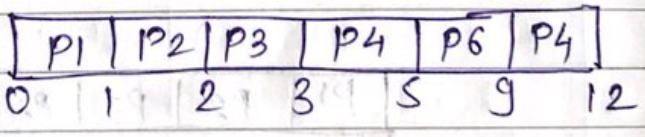
P6 finishes



P1, P2, P3, P4, P5 and P7 are in queue and among these P4 has higher priority so it continues.

→ At, t = 12

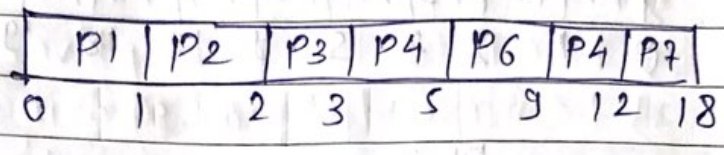
P4 finishes



P1, P2, P3, P5 and P7 in queue P7 starts because of priority.

→ At, t = 18

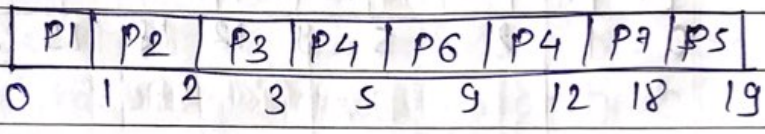
P7 finishes



P1, P2, P3 and P5 in queue.
P5 starts.

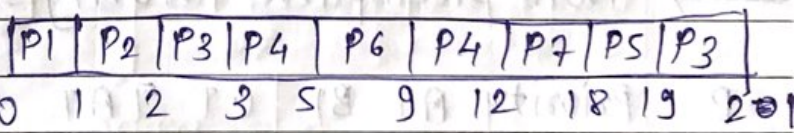
→ At, t = 19

P5 finishes



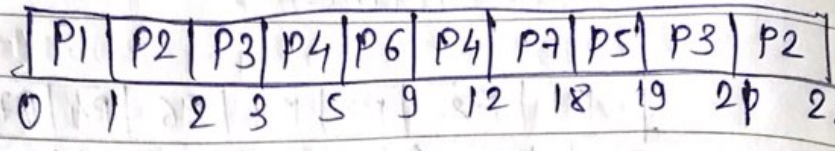
P1, P2 and P3 in queue
P3 starts.

→ At, t = 201



P1 and P2 in queue.

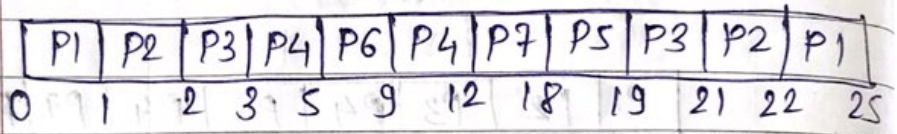
→ At, t = 22



P1 in queue.

→ At, t = 25

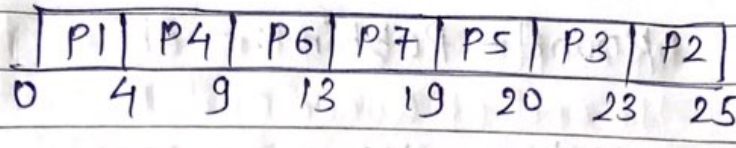
P1 finishes.



Avg. Weight time = 11.4

→ Non-preemptive Priority scheduling

P	Priority	AT	BT	CT	TAT	WT
1	2	0	4	4	4	0
2	4	1	2	25	24	22
3	6	2	3	23	22	19
4	10	3	5	9	6	1
5	8	4	1	20	16	15
6	12	5	4	13	8	4
7	9	6	6	19	13	7 / 68



Avg. Weight time = 9.71 sec

→ Drawback :-

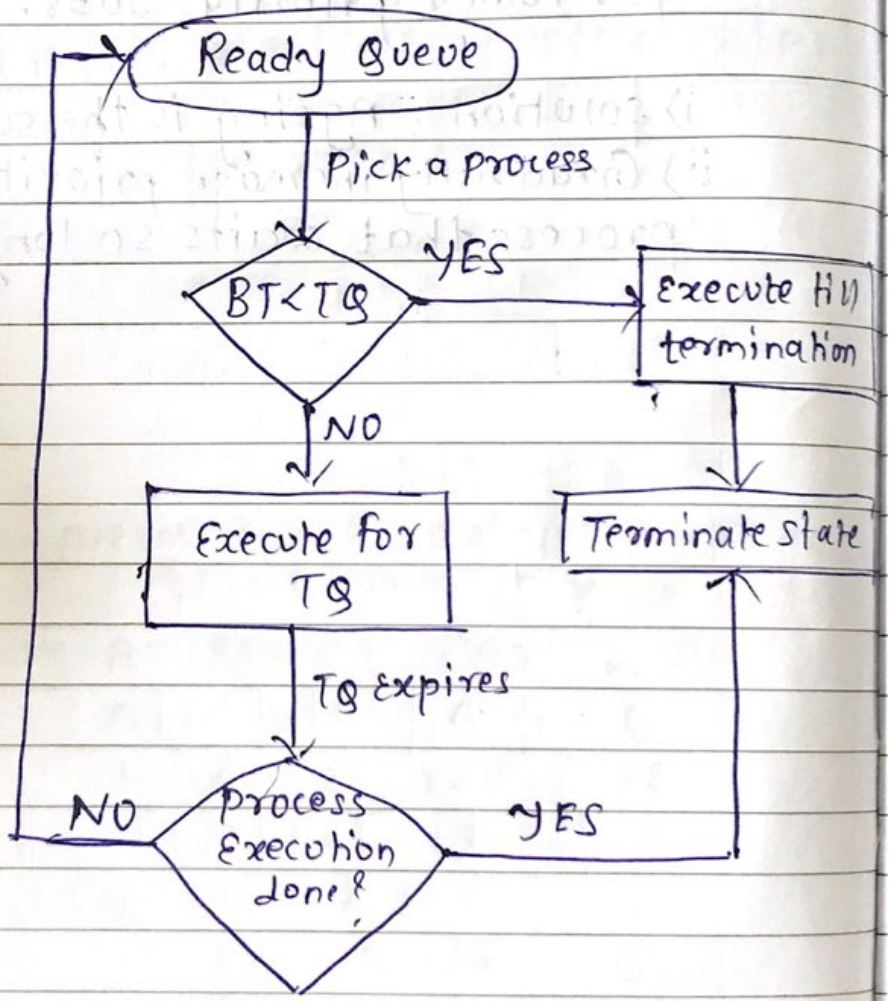
i) Indefinite waiting (starvation) for lower priority Jobs.

i) solution : Ageing is the solution.
ii) Gradually increase priority of process that waits so long..

4) Round - Robbin :-

1) Most popular.
2) Like FCFS but preemptive.
3) Designed for time sharing systems.
4) No process is going to wait forever, very low starvation.
5) Easy to implement.

→ Process Flow:



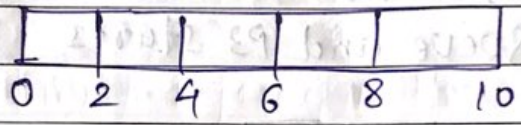
classmate
Date
Page

→ Ex:-

Process Queue	Burst time
P1	4
P2	3
P3	5

[P1 P2 P3] in queue

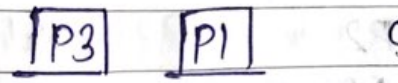
Time slice = 2

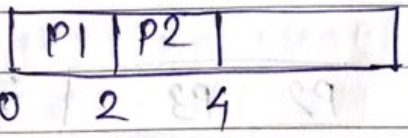


Step 1) Execution begin with P1, which has BT=4. P2 and P3 are still in queue.

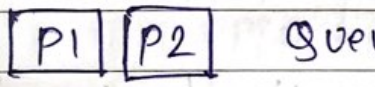
[P2] [P3] Queue.

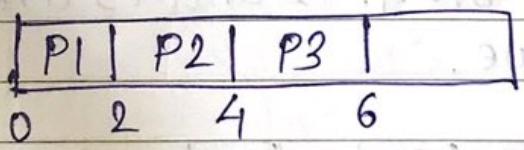
Step 2) At, t = 2 P1 is added to the end of Queue and P2 starts



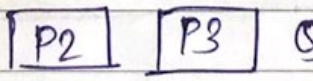


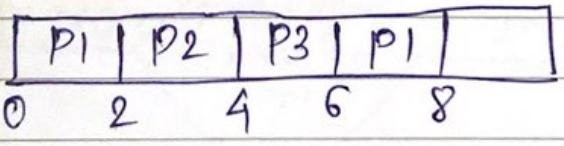
Step 3) At, t = 4, P2 is added at end of Queue and P3 starts





Step 4) At, t = 6, P3 is added at end of Queue. P1 starts.





Step 5) At, t=8, P1 has completed. P2 starts.

| P3 | queue

| P1 | P2 | P3 | P1 | P2 |
0 2 4 6 8 9

~~At~~, Step 6) At, t=9, P2 finishes P3 starts.

| empty | queue

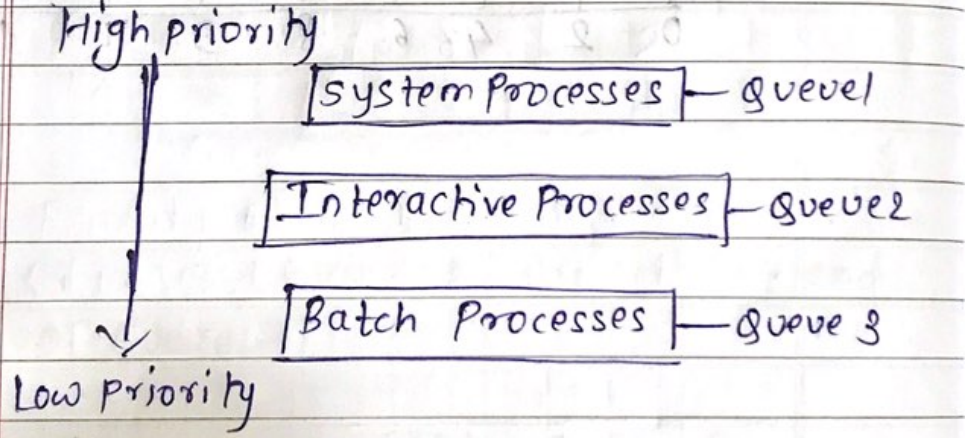
| P1 | P2 | P3 | P1 | P2 | P3 | P3 |
0 2 4 6 8 9 11 12

CLASSMATE
Date
Page

→ Multi level Queue Scheduling (MLQ):-

1) In MLQ, Ready Queue is divided into multiple queues depending upon priority.
2) A process is permanently assigned to one of queues based on some property of process, memory, size, process priority or process type.

3) Three different types of processes System processes, interactive processes, and Batch processes. All three processes have their own queue.



→ description of process:-

1) system processes:- The CPU itself has its own process to run which is generally termed as system process

2) Interactive Process: Needs user input

3) Batch Process:- Runs silently, no user input required.

→ Each Queue uses its own scheduling Algorithm. Queue1 and Queue2 uses Round Robbin and Queue3 uses FCFS.

→ Scheduling among queue:-

1) Fixed priority preemptive scheduling: Each Queue has absolute priority over lower priority queue. Let us consider following priority order queue1 > queue2 > queue3.

→ ~~Multi level Queue scheduling (MLQ)~~ :-

According to this algorithm, no process in the batch queue can run unless queue1 and queue2 are empty. If any batch process is running and an system or interactive process entered redy queue the batch process is preempted.

Ex:

process	AT	BT	Q.N
P1	0	4	1
P2	0	3	1
P3	0	8	2
P4	10	5	1

Priority of Q1 is greater than Q2.
Q1 uses RR and Q2 uses FCFS.

P1	P2	P1	P2	P3	P4	P3

0 2 4 6 7 10 15 20

Working:-

1) At starting both queues have process, but because of higher priority of Q1 both process of Q1 runs in RR fashion and completes after 7 units.

2) Then process in Q2 starts running. But while it is running P4 comes in Q1 and interrupts P3 and starts running for 5 sec, and after its completion P3 executes.

→ Multi level feedback Queue scheduling (MLFQ):

1) MLFQ CPU scheduling is like MLQ but in this processes can move between the queues. And thus much more efficient than MLQ.

2) Allows the process to move between Queues. The idea is to separate process according to the characteristics of their BT. If a process uses too

much time, it will be moved to lower priority queue. This scheme leaves I/O bound and interactive processes in the higher priority queue.

In addition, a process that waits too much in a lower priority queue may be moved to a higher priority. This form of ageing prevents starvation.

——X——

→ Concurrency:-

Concurrency is the execution of multiple instruction sequences at the same time. It happens in OS when there are several process threads running in parallel. The running process threads always communicate with each other through shared memory or message passing.

→ Thread scheduling:

Threads are scheduled for executions based on their priority. Even though threads are executing within the runtime, all threads are assigned processor time slices by OS.

→ Threads context-switching:

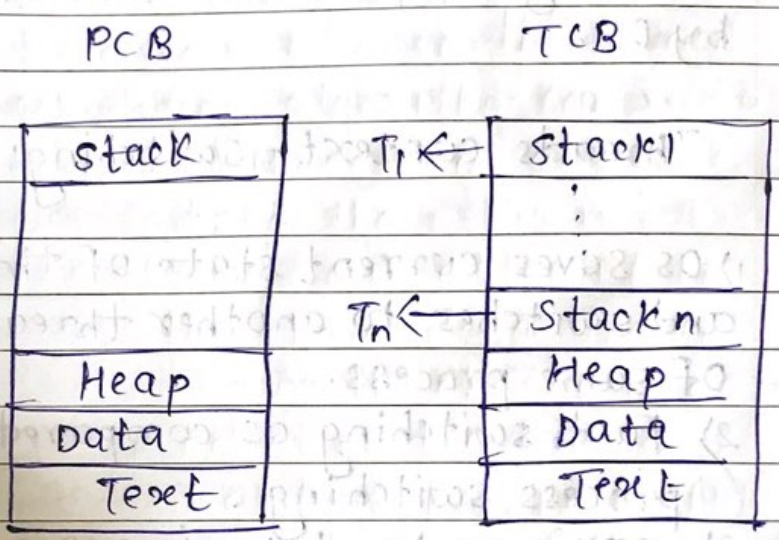
1) OS saves current state of thread and switches to another thread of same process.
2) Fast switching as compared to process switching.
3) CPU's cache status is reserved.

→ How each thread get access to the CPU

1) Each thread has its own program counter.
2) Depending upon the thread scheduling algorithm, OS schedule these threads.
3) OS will fetch instructions corresponding to PC of ~~any~~ that thread and execute instruction.

→ Thread control Block [TCB]:

Very similar to PCB which represents processes, Thread control Block [TCB] represents threads generated in the system.



The architecture of PCB and TCB are same, But in TCB there will 'n' stacks for 'n' threads.

→ Benefits of Multi-threading :-

1) Responsiveness.
2) Resource sharing.
3) Economy
4) Threads allow utilization of multiprocessor architecture to a greater scale and efficiency.

→ Critical section in OS :-

1) Critical section is a part of program which tries to access shared resources. That resource may be any resource in a computer like memory location, data structure CPU or any I/O device.

2) critical section refers to the segment of code where processes/threads access shared resources.

→ Major thread scheduling Issue :

a) Race condition :-

A Race condition occurs when two or more threads can access shared data and try to change it at the same time. Because the thread scheduling algorithm can swap between threads at any time, you don't know the order in which the threads will attempt to access the shared data. Therefore the result of the change in data is dependent on the thread scheduling algorithm, i.e, both threads are "racing" to access/change the data.

→ solution to Race condition :

1) Mutual Exclusion :

Mutual exclusion implies that only one process can be inside the critical section at any time. If any

Other processes require critical section, they must wait untill it is free.

2) Progress:

Progress means if a process is not using C.S, then it should not stop any other process from accessing it. In other words, any process can enter a critical section if it is free.

3) Bounded waiting:

Bounded waiting means that each process must have a limited waiting time. It should not wait endlessly to access the critical section.

———x———

→ Mutex/locks:

a) locks can be used to implement mutual exclusion and avoid race condition by allowing only one thread/process to access critical section.

Disadvantages

i) contention: one thread has acquired the lock, other threads will be busy waiting, what if thread that had acquired the lock dies, then all other threads will be in infinite waiting.

ii) Deadlock.

—X—

→ conditional variable :-

→ The conditional variable is a synchronization primitive that lets the thread wait untill a certain condition occurs.

→ Works with a lock.

→ Threads can enter a wait state only when it has acquired a lock. When a thread enters the wait state, it will release the lock and wait untill another thread notifies that the event has occured the lock immediately and starts executing.

→ Why to use conditional variable?

To avoid busy waiting.

Busy waiting is a condition where T1 thread has locked the critical section, T2 thread waits for it's execution at the entry point. Due to this condition T2 thread wastes the CPU cycles.

→ There are two types of actions that can be performed with condition Variables:

1) wait 2) signal.

1) We use wait instruction in a thread if we want to halt the execution of that thread till a certain condition is met.
2) We use signal instruction if we want to continue executing the leading thread in waiting queue.

classmate
Date
Page

Semaphores :-

Semaphore is the variable which stores the entire wake up calls that are being transferred from producer to consumer. It is a variable on which read, modify and update happens automatically in kernel mode.

Semaphore can be divided into two categories :

1) counting semaphore.
2) Binary semaphore.

counting semaphore :-

There are scenarios in which more than one processes need to execute in critical section simultaneously. However, counting semaphore can be used when we need to have more than one process in critical section at same time.

classmate
Date
Page

In this mechanism, the entry and exit in the critical section are performed on the basis of the value of counting semaphore. The value of counting semaphore at any point of time indicates the maximum no. of processes that can enter in the critical section at the same time.

A process p which wants to enter in critical section first decrease the semaphore value by 1, and then check whether the value gets negative or not. If its negative then the process is pushed in the list of blocked processes, otherwise it gets enter into critical section.

When a process exits from the critical section, it increases the counting semaphore by 1 and then checks whether it is negative or not. If it's negative then that means that atleast one process is waiting hence, to ensure bounded

waiting, the first process in the list of blocked processes will wake up and gets enter in CS.

The processes in the blocked list will get waked in the order in which they slept. If the value of counting semaphore is negative then it states the no. of processes in the blocked state while if it is positive then its state the no. of slots available in CS.

→ Binary semaphore:

In counting semaphore, Mutual exclusion was not provided because we have set processes of required to execute in CS simultaneously.

However, binary semaphore strictly provided mutual exclusion. Here, instead of having more than 1 slots available in CS, we can only have atmost 1 process in CS.

The semaphore can have only two values, 0 or 1.

Parameters	Semaphore	Mutex
Mechanism	It is a type of signaling mechanism	It is a locking mechanism
Data type	Semaphore is an integer variable	Mutex is just an object
Modification	The wait and signal operations can modify a semaphore	It is modified only by the process that may request or release a resource
Thread	You can have multiple program threads	You can have multiple program threads in mutex but not simultaneously.

→ Producer consumer problem :-

1) In OS producer is a process which is able to produce data/item.
2) consumer is a process that is able to consume the data/item produced by the producer.
3) Both producer and consumer shared a common memory buffer. This buffer is a space of certain size in the memory of the system which is used for storage. The producer produces the data into the buffer and the consumer consumes the data from buffer.

→ What is the Producer - consumer problem ?

1) Producer process should not produce data when buffer is full.
2) consumer process should not consume data when the shared buffer is empty.
3) The access to the shared buffer should be mutually exclusive, i.e,

at a time only one process should be able to access the shared buffer and make changes to it.

→ solution of producer-consumer problem using semaphores.

Three semaphore variables are used.

1) FULL.
The full variable is used to track the space filled in the buffer by the producer process. It is initialized to zero initially.

2) Empty.
The empty variable is used to track the empty space in variable. It is initialized to the size of buffer initially.

3) Mutex

Mutex is used to achieve mutual exclusion. mutex ensures that at any particular time only the producer or consumer is accessing the buffer.

→ We will use signal() and wait() operation in above mentioned semaphores.

Signal() function increases value of semaphore by 1 and wait() decreases value by 1

→ Code for Producer process:

```
Void Producer (){
    while (true) {
        wait(Empty);
        wait(mutex);
        add();
        signal(mutex);
        signal(Full);
    }
}
```

1) After the item is produced, wait operation is performed on empty. This indicates that the empty space in buffer has decreased by 1. The wait operation is carried out on mutex so that consumer process cannot interfere.

2) add() method is used to add the item in buffer

3) After the item is put in buffer signal operation is performed on mutex and full. The former indicates that the consumer process can act and latter shows that the buffer is full by 1.

→ code for consumer process:

```
Void consumer (){
    while (true){
    wait (full);
    wait (mutex);
    consume();
    signal (mutex);
    signal (Empty);
    }
}
```

The wait operation is carried out on full. This indicates that the item in buffer is decreased by 1. wait operation is performed on mutex so producer does not interfere.

consume() removes item from buffer. After that signal operation is performed on mutex and empty.

———x———

→ Reader–Writers Problem :-

The reader-writers problem relates to an object such as a file that is shared between multiple processes. Some of these processes are readers i.e, they only want to read the data from the object and some of the processes are writers i.e they want to write into object.

The reader writer problem is used to manage synchronization so that there are no problems with the object data.
Ex: if two readers access the object at the same time there is no problem with the object data.
However if two writers or a reader and writer access the object at the same time, there may be problems.

To solve the problem, a writer should get exclusive access to an object i.e when a writer is accessing object, no reader or writer.

→ Three variables are used: mutex, wrt and readcnt.

1) semaphore mutex, wrt:
semaphore mutex is used to ensure mutual exclusion when readcnt is updated i.e, when any reader enters or exit from critical solution and semaphore wrt is both used by readers and writers.

2) int readcnt : (rc) :
readcnt tells no. of processes performing read in CS.

→ Writer Process :-

```
do {
    wait(wrt);
    // do write operation
    signal(wrt);
} while(true);
```

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

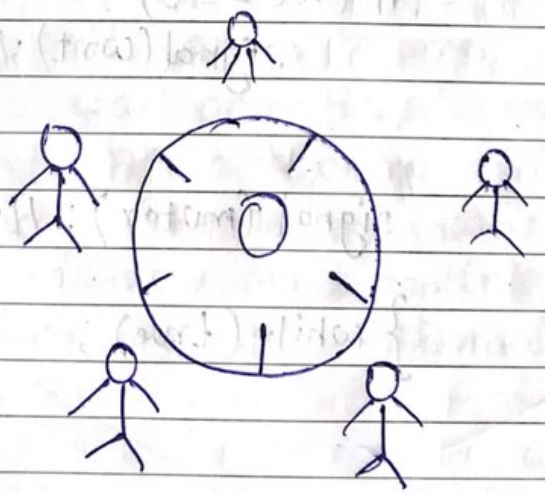
→ Reader Process :-

```
do {
    wait(mutex);
    rc++;
    IF (rc==1)
        wait(wrt);
    signal(mutex); //Other readers can enter while this current reader is inside critical solution.
        wait(mutex); //reader wants to leave
    rc--;
    if(rec==0)
        signal(wrt); //writer can enter
    signal(mutex); //reader leaves
} while(true);
```

1) Reader request the entry to critical section.
2) If allowed:
i) It increments the count of no. of readers inside the critical section. If this reader is the first reader entering, it locks cort, that no writer should enter.
ii) It then, signals mutex as any other reader is allowed to enter while others are ~~readi~~ already reading.
iii) After ~~red~~ reading it exits cs. then cort can enter.

—X—

→ Dining Philosopher problem:-



1) We have 5 philosophers.
2) They spend their life just being in two states:
a) thinking b) eating.

3) They sit on a circular table surrounded by 5 chairs, in the centre of a table is a bowl of noodles and the table is laid with 5 single forks.

4) Thinking state: When a philosopher (ph) think's he doesn't interact with others.

5) Eating state: When a ph, gets hungry, he tries to pick up the 2 forks adjacent to him. He can pick one fork at a time.

6) one can't pick a fork if it is already taken.

7) When ph has both forks at the same time, he eats without releasing Forks.

8) Soln: can be given using semaphores

1) Each fork is binary semaphore.
2) A ph calls wait() to acquire a fork.
3) Release fork by calling signal.
4) Semaphore fork[5]{1}.

→ Some methods to avoid deadlock:

1) Allow atmost 4 ph to sit.
2) Allow a ph to pick up his fork only if both forks are available.

3) Odd-even rule:
an odd ph picks up first his left fork and then his right fork, whereas an even ph. picks up his right fork and then left fork.

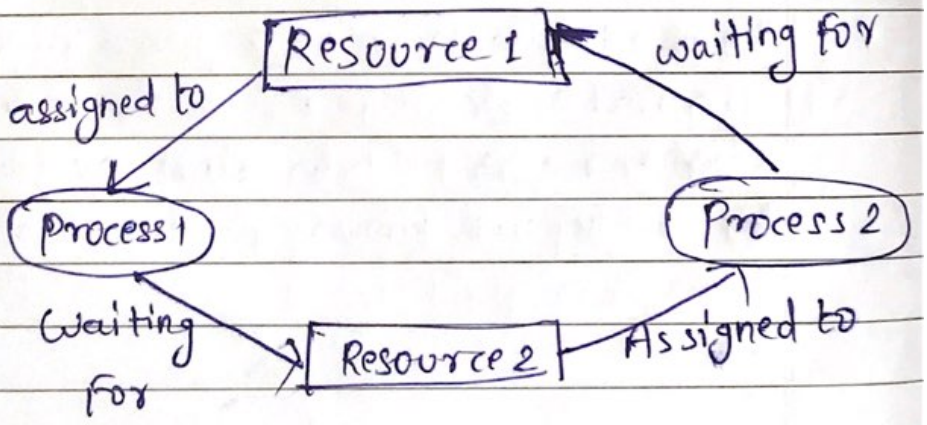
→ structure of chopstick :-

```
semaphore chopstick[5];

do{
    wait(chopstick[i]);
    wait(chopstick[(i+1)%5]);

    //eating

    signal(chopstick[i]);
    signal(chopstick[(i+1)%5]);

}while(1);
```

DEADLOCK:

All the processes in a system require some resources such as CPU, file storage, i/o devices, etc to execute it. Once the execution is finished, the process releases the resource it was holding. However when many processes run on a system they also compete for these resources they require for execution. This may arise a deadlock situation.

A deadlock is a situation in which more than one process is blocked because it is holding a resources and also requires some resource that is acquired by some other process. Therefore, none of the process gets executed.



classmate
Date
Page

→ Necessary condition for Deadlock:

1) Mutual exclusion: only one process can use a resource at any given time i.e, the resource are non-sharable.

2) Hold and wait: A process is holding atleast one resource at a time and is waiting to acquire other resources held by some other process.

3) No preemption: The resource can be released by a process ~~voline~~ voluntarily i.e after execution of process.

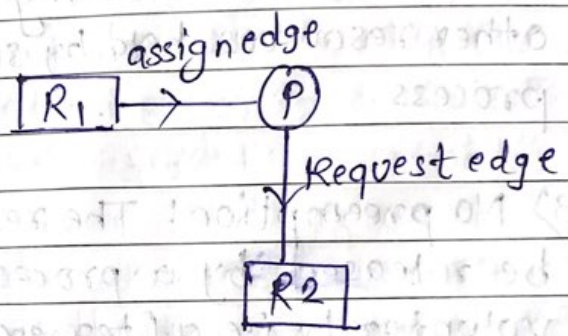
4) circular wait: A set of processes are waiting for each other in circular fashion.

Ex: Lets say there are a set of processes $\{P_0, P_1, P_2, P_3\}$ such that P_0 depends P_1, P_1 depends on P_2 and ... P_3 depends on P_0.

→ Resource allocation Graph:

1) Vertex : i) Process vertex (P)
ii) Resource vertex [R]

2) Edges : i) Assign
ii) Request



→ Methods to handle deadlock:

1) Deadlock Prevention:

This is done by restraining the ways a request can be made. Since deadlock occurs when all the above four conditions are met, we try to prevent any one of them, thus preventing a deadlock.

2) Deadlock Avoidance :-

When a process request a resource, the deadlock avoidance algorithm examines the resource-allocation state. If allocating that resource sends the system into an unsafe state, the request is not granted.

3) Deadlock Detection and Recovery

We let the system fall into a deadlock and if it happens, we detect it using a detection algorithm and try to recover it.

4) Deadlock Ignorance :

In the method, the system assumes the deadlock nevers occurs. Since the problem of deadlock situation is not frequent, some system simply ignore it. It is an example of Ostrich algorithm.

→ Deadlock Prevention: By ensuring atleast one of the necessary condition cannot hold.

a) Mutual exclusion:

It is not possible to dis-satisfy the mutual exclusion because some resources, such as the tape drive and printer, are inherently non-shareable.

b) Eliminate Hold and wait

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

c) Eliminate No-preemption :

preempt resources From the process when resource required by other high priority processes.

d) Eliminate circular wait :

P1 and P2 both require R1 and R1 , locking on these resource should be like both try to lock R1 then R2. By this way which ever process first locks R1 will get R2.

—X—

→

Deadlock	starvation
1) Deadlock is a situation where no process got blocked and no process proceeds.	1) starvation is a process where the low priority process got blocked and high priority proces proceed.
2) Deadlock is an infinite waiting	2) starvation is long waiting but not infinite.

3) Every deadlock is always a starvation	3) Every starvation need not to be deadlock.
4) The requested resource is blocked by other process.	4) The requested resource is continously be used by the higher priority process.

—— X ——

→ Deadlock avoidance :-

Deadlock avoidance is a process used by the OS to avoid deadlock.

→ But how OS avoids deadlock ?

OS avoids deadlock by knowing the maximum resources requirements of the process initially, and also OS knows the free resource available at that time. OS tries to allocate the resources according to the process requirement and checks if the allocation can lead to safe

state or unsafe state. If the resource allocation leads to an unsafe state, then OS does not proceed further with the allocation sequence.

→ How does deadlock avoidance work?

Process	Maximum required	current available	Need
P1	9	5	4
P2	5	2	3
P3	3	1	2

Let's consider three processes P1, P2 and P3. Some more information on which the processes tells the OS are :

1) P1 process needs a maximum 9 resource to complete its execution. P1 is currently allocated with 5 resources and needs 4 more to complete its execution.
2) P2 process needs 5 resource and in currently allocated with 2 resource and needs 3 more.

3) P3 process needs a maximum of 3 resources and is currently allocated with 1 resource, and needs 2 more.

4) OS knows that only 2 resources out of total available resources are currently free.

→ soln:

As only 2 resources are free for now, then only P3 can satisfy its need for 2 resources. If P3 takes 2 resources and completes its execution, then P3 can release 3 (1+2) resources. Now the free resources which P3 released can satisfy the need of P2. Now, P2 after taking the three free resources, can complete its execution and then release 5 (2+3) resources. Now 5 resources are free. P1 can take 4 out of 5 and execute. So with 2 free resource available initially, all the processes were able to complete their execution leading to safe state. Order of execution P3, P2, P1.

What if initially there was only 1 resource, None of the processes would be able to complete its execution, leading to unsafe state.

→ Safe state and unsafe state:

→ If the operating system is able to allocate or satisfy the maximum resource requirements of all the processes in any order then the system is said to be in safe state.

So, safe state does not lead to deadlock.

→ unsafe state: If OS is not able to prevent processes from requesting resources which can also lead to deadlock,

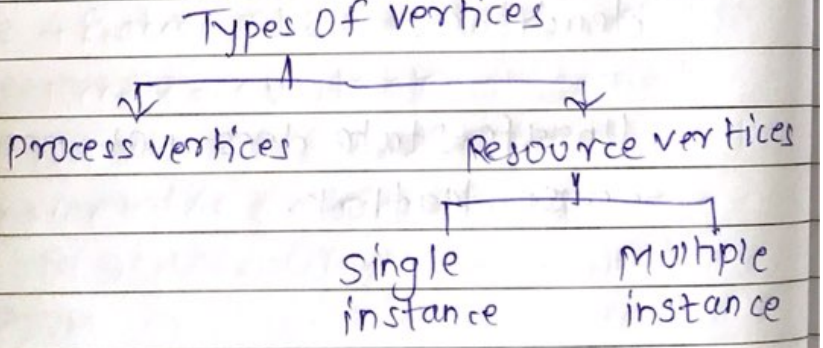
Unsafe state does not necessarily cause deadlock.

→ Deadlock avoidance soln:

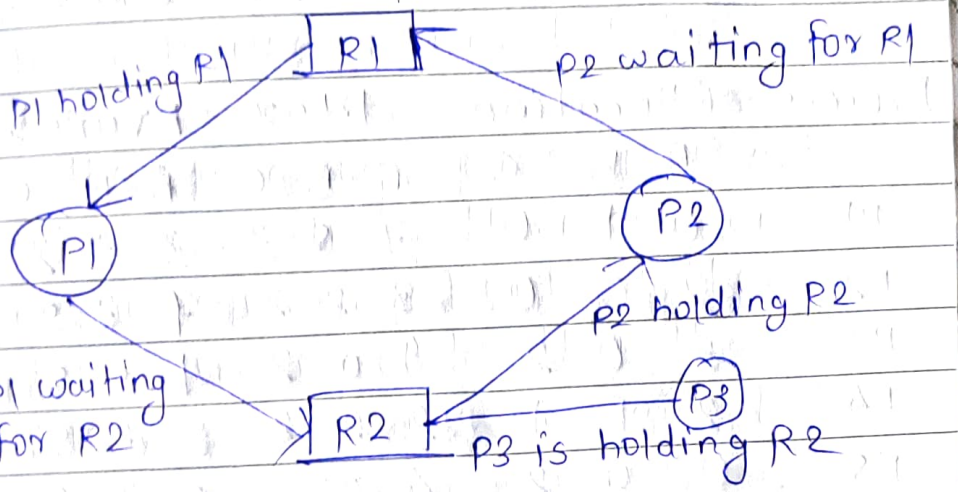
1) Resource allocation Graph
2) Banker's Algorithm

→ Resource allocation Graph:-

RAG is used to represent the state of the system in the form of a Graph. The Graph contain all processes and resources which are allocated to them and also the requesting resources of every process. Sometimes, if the no. of processes is less, we can easily identify a deadlock in the system just by observing a Graph, which cannot be done easily by using tables we use in Banker's algorithm.



RAG has a process vertex represented by a circle and resource vertex represented by a box. The instance of the resources is represented by a dot inside the box.



→ Banker's Algorithm:

The banker's algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an "s-state" check to test for possible activities, before deciding allocation should be allowed to continue.

Ex: Consider an system that contains five processes P1, P2, P3, P4, P5 and the three resource types A, B, C following are the resource types A has 10, B has 5 and C has 7.

Process	Allocation			Max			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P1	0	1	0	7	5	3	3	3	2	7	4	3
P2	2	0	0	3	2	2	7	3	5	1	2	2
P3	3	0	2	9	0	2	1	5	5	6	0	0
P4	2	1	1	2	2	2	8	3	5	0	1	1
P5	0	0	2	4	3	3	6	2	4	4	3	1

→ Apply Banker's Algorithm

Available Resources of A, B and C are 3, 3 and 2.

Step 1: for Process P1:

Need <= available

7, 4, 3 <= 3, 3, 2 condition is false.

So, we examine another process.

classmate
Date
Page

step 2 : for process P2 :

Need <= available

1,2,2 <= 3,3,2 condition true

New available = available + allocated

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

Examine P3

step 3 : for Process P3 :

P3 Need <= available

(6,0,0) <= (5,3,2) condition is false

we examine another process P4.

step 4 : for Process P4

P4 Need <= available

(0,1,1) <= (5,3,2) condition is true

New available = available + allocated

(7,4,3) = (5,3,2) + (2,1,1)

classmate
Date
Page

Step 5: For process P5:

P5 Need <= available
(4,3,1) <= (7,4,3)
condition is true

New available = available + allocation
(7,4,5) => (7,4,3) + (0,0,2)

step 6: For process P1

P1 Need <= available
7,4,3 <= 7,4,5 is true

New available = available + allocation
(7,4,5) + (0,1,0) => (7,5,5)

Examine, another Process P3

step 7: For Process P3

P3 Need <= available
(6,0,0) <= (7,5,5) is true

New available = available + allocated
(10,5,7) -> = (7,5,5) + (3,0,2)

How The sequence for process execution is : P2, P4, P5, P1, P3.

—X—

→ Memory Management in Os :

In a multiprogramming computer, the Os system resides in a part of memory and the rest is used by multiple processes. The task of subdividing the memory among different processes is called memory management. Memory Management is a method in the Os to manage operation between main memory and disk during process execution. The main aim of memory management is to achieve efficient utilization of memory.

→ Why Memory Management is Required ?

1) Allocate and de-allocate memory before and after process execution.
2) To keep track of unused memory space by processes.
3) To minimize fragmentation issues.
4) To proper utilization of main-memory.

5) To maintain data integrity while executing of process.

→ Logical Address and physical Address:-

→ What is a logical address?

A logical address is a address that is generated by CPU during program execution. The logical address is a virtual address as it does not exist physically, and therefore it is also known as virtual address. This address is used as a reference to access the physical memory location by CPU. The term logical address space is used to set all logical addresses generated from a program's perspective.

A logical address usually ranges from 0 to max. These logical address (generated by CPU)

Combines with the base address generated by MMU to form the physical address. The hardware device MMU is used for mapping logical addresses to their corresponding physical address.

→ Range: ~~0 to~~ 0 to max

→ What is a physical Address?

The physical address identifies the physical location of required data in memory. The user never directly deals with the physical address but can access it by its corresponding logical address. The physical address is in the memory unit. It's a location in the main memory physically. The set of all physical addresses corresponding to the logical addresses is commonly known as physical address space.

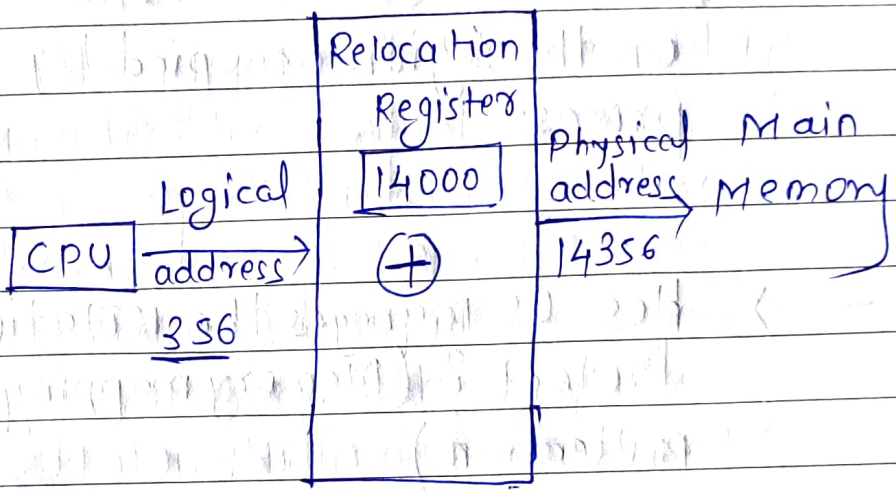
* Range: (R+0) to (R+max), for a base value R.

→ Physical Address → MMU computed

logical Address → CPU Generated

Terms	Logical address	Physical address
Definition	The CPU generates the logical address while the program is running	The physical addre is a location in memory.
Location	Logical address does not exist in memory, and known as virtual address.	Physical address is a location in memory unit.
Access	The logical address is used as a reference to access the physical address	The physical address cannot be accessed directly.
Address space	Set of logical addresses generated by CPU is called logical address space	Physical address mapped to the logical address is called physical address space.

→ The user's program mainly generates the logical address, and the user thinks that the program is running in this logical address, but the program mainly needs physical memory in order to complete its execution.



→ What is Memory Management unit (MMU)?

The run time mapping between the virtual and physical addresses is done by a hardware device known as MMU. The OS will handle the processes and move the processes between disk and memory in memory management. It keeps track of available and used memory. The memory management unit is a combination of these two

registers,

1) Base Register: It contains the starting physical address of the process.

2) Limit Register: It mentions the limit relative to the base address on the region occupied by the process.

→ How OS manages the isolation and protect? (Memory mapping and protection)

→ a) OS provides this Virtual address space (VAS) concept.
b) To separate memory space, we need the ability to determine the range of legal addresses that the process may access and to ensure that the process can access only these legal addresses.
c) The relocation register contains value of smallest physical address [Base address [R]], the limit register

contains the range of logical addresses.

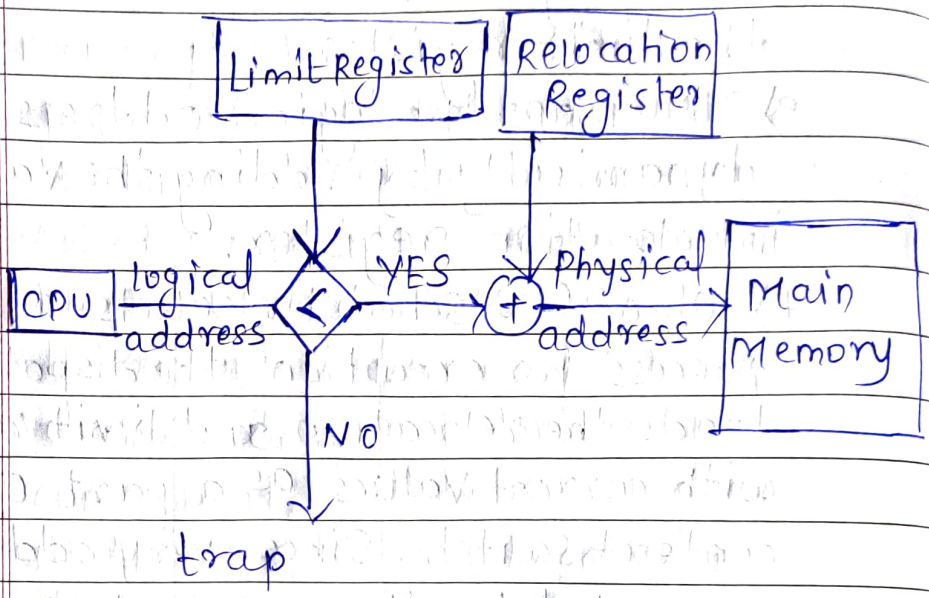
d) Each logical address must be less than the limit register.

e) MMU maps the logical address dynamically by adding the value in relocation register.

f) When CPU scheduler selects a process for execution, the dispatcher loads the relocation and limit registers with correct values as a part of context switch. Since every address generated by the CPU is checked against these registers, we can protect both OS and other users programs and data being modified by running process.

g) Any attempt by a program executing in user mode to access the OS memory or other uses' memory results in a trap in the OS, which treat the attempt as a fatal error.

h) Address Translation



→ Allocation methods on physical Memory:

a) Contiguous Allocation.
b) Non-contiguous Allocation.

1) Contiguous Memory Allocation:

It is the type of memory allocation method. When a process requests the memory, a single contiguous section of memory blocks is allotted depending on its requirement.

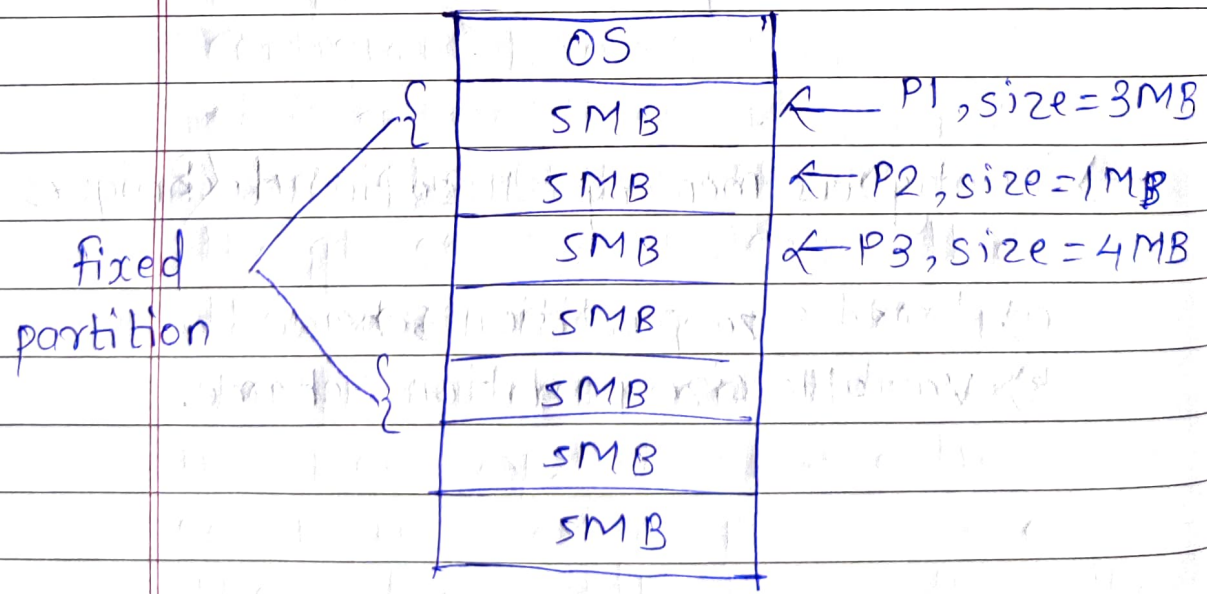
It is completed by partioning the memory into fixed-sized partitions and assigning every partition to a single process. However it will limit the degree of multiprogramming to the number of fixed partitions done in memory.

→ contiguous memory allocation techniques:

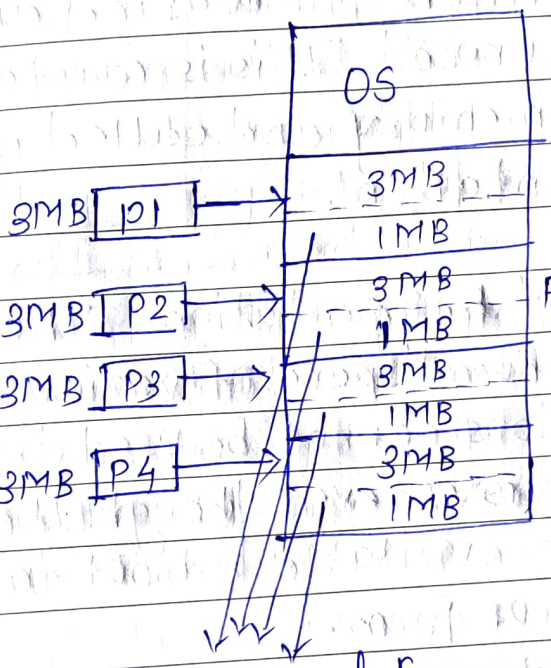
a) fixed size partition scheme
b) variable size partition scheme.

1) Fixed size Partition Scheme :

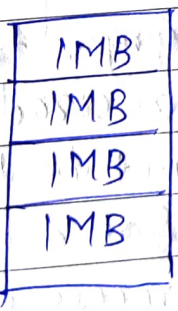
In this type of contiguous memory allocation technique, each process is alloted a fixed size continous block in main memory. That means, there will be continous blocks of fixed size into which the complete memory will be divided, and each time a process comes in, it will be alloted one of the free blocks. Because irrespective of the size of the process, each is alloted a block of same size memory space.



Note: The no. of processes that can stay in the memory at once is called the degree of multiprogramming of the system.



Internal fragmentation



Unused space of the four partitions (External fragmentation)

~~Proc~~ Process P5 4MB

P5 can't be executed even though there is a 4MB space available but not contiguous.

→ Limitations:-

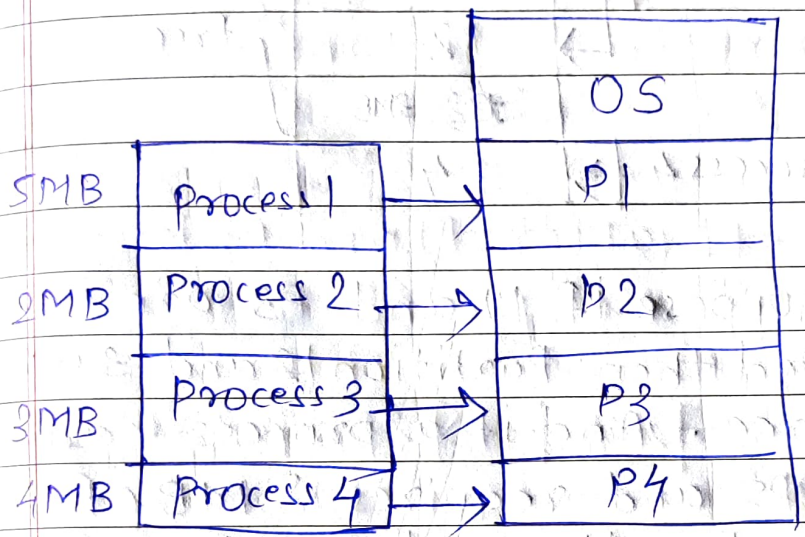
1) Internal fragmentation:
If the size of the process is lesser than the total size of the partition then some size of the partition gets wasted and remain unused. This is wastage of the memory and called internal fragmentation.

2) External fragmentation: The total unused space of various partitions cannot be used to load the processes even though there is space available but not in contiguous form.

3) Limitation on process size: If the process size is larger than the size of maximum sized partition then that process cannot be loaded into memory. Therefore a limitation can be imposed on process size that it cannot be larger than the size of largest partition.

2) Variable size partioning:

In this teahnique the partition size is not declared initially. It is declared at the time of process loading.

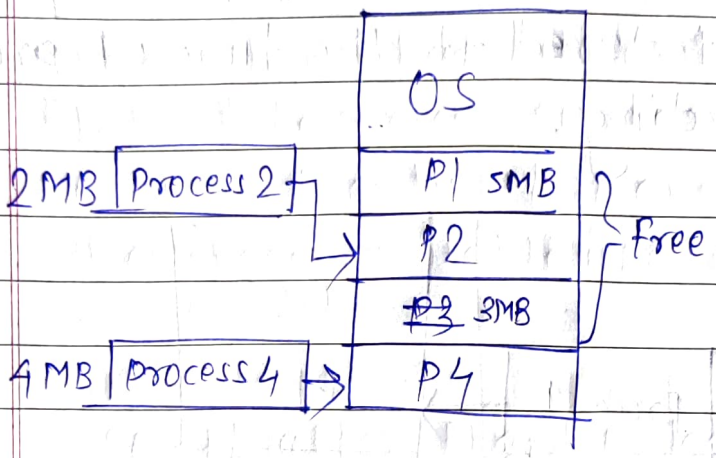


Advantages over fixed partioning

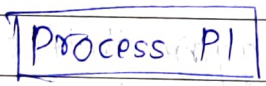
1) No internal fragmentation
2) No limit on size of process
3) Better degree of multiprogramming.

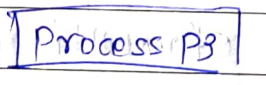
→ Limitation:

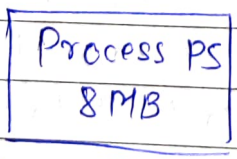
1) External fragmentation:



suppose if P1 and P3 are executed then Partition 2 and 3 will be free. And if a process came i.e. P5 and requires 8MB of space but it could not be allocated because memory is available but it is not contiguous.

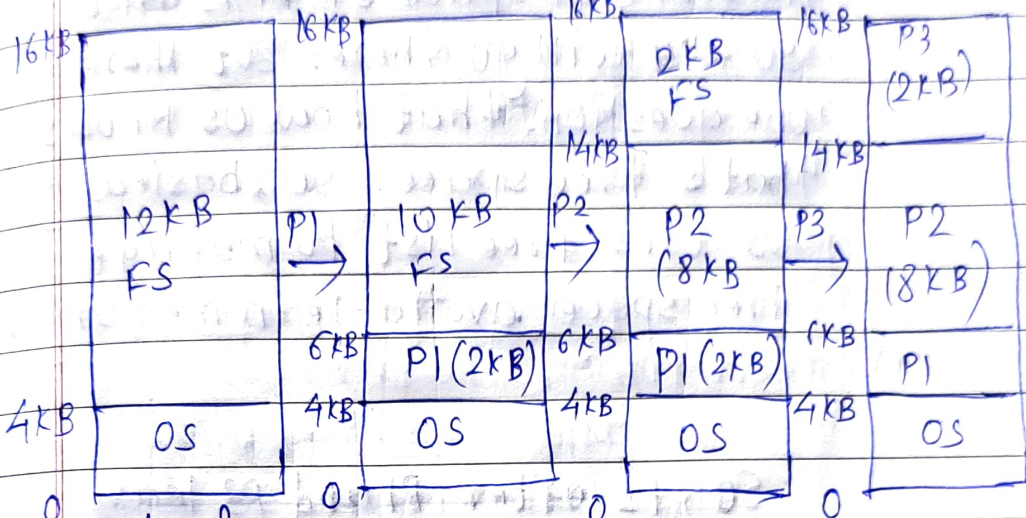






P5 can't be loaded into memory because it is not contiguous.

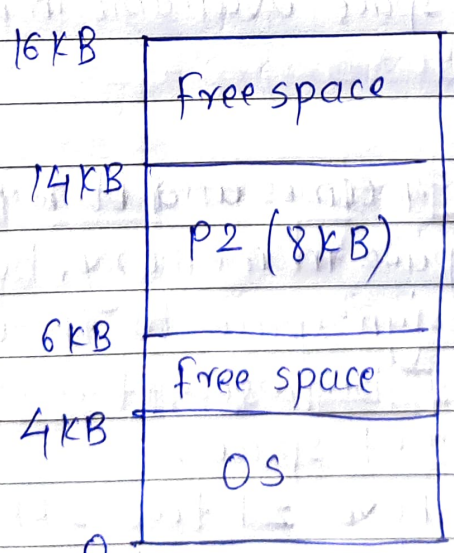
→ How OS manages free space?



Initial

free list: 4KB → 10KB → NULL 2KB → NULL NULL

After sometimes P1 and P3 executes and they leave the main memory.



main memory after P1 and P3 leaves.

→ If After sometimes P4 comes and require 2KB of space and if know that 2KB of space of P1 is left so, P4 will go there. But there is one question, that how OS knows that's free space. so, basically OS uses free list to manage free space available in memory.

So, FL after P1 and P3 leaves main memory.

FL : [2KB] → [2KB] → NULL

So, OS knows that it has 2, 2KB free space available in Memory.

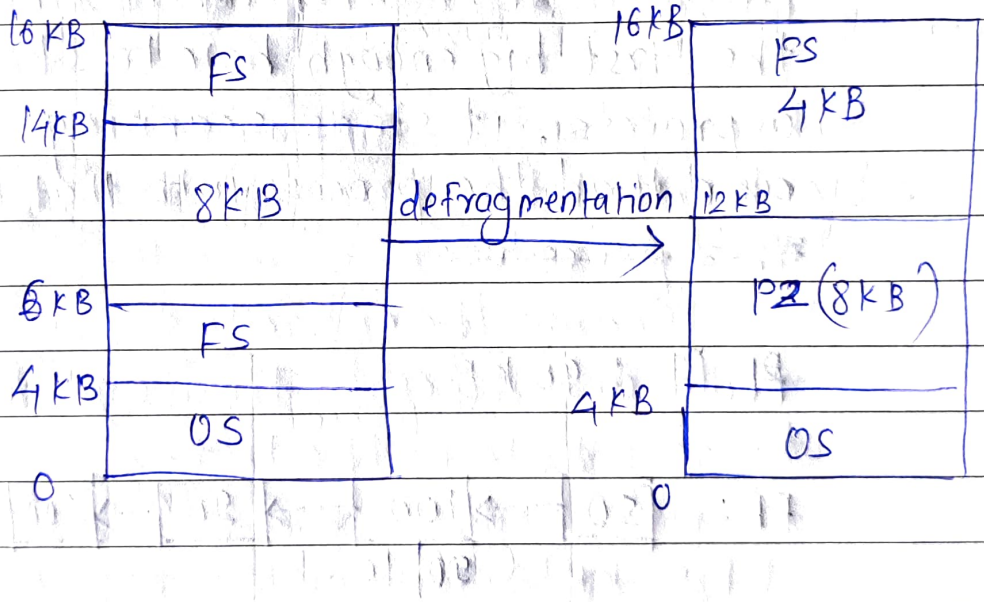
→ Now, if P4 comes and it requires 3KB of space in memory, but OS can't assign space to these process in memory because, it doesn't have 3KB of space.

It have ~~2~~ two 2KB of space available, i.e total 4KB

but it is not contiguous, it is fragmented. so to solve these problem, we use defragmentation.

→ Defragmentation :

So, in defragmentation we move the occupied space of memory in one place and the free space to one side.



Free list after defragmentation

FL : [4KB] → NULL

→ Partioning Algorithms :-

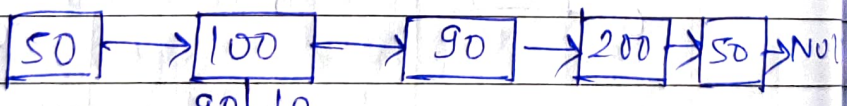
There are various algorithms which are implemented by OS in order to find out holes in the linked list and allocate them to process.

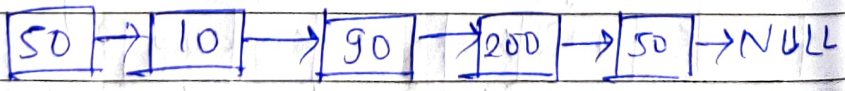
1) first fit :-

first fit algorithm scans the linked list and whenever it finds the first big enough hole to store a process, it stops scanning and load the process into that hole.

~~This procedure~~

P1 : 90 KB



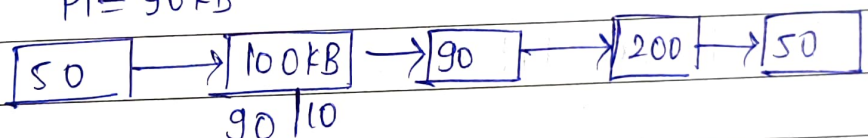


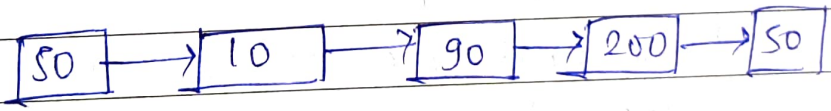
2) Next Fit :

Next fit algorithm is similar to first fit algorithm except that fact, that Next fit scans the linked list from the node where it previously allocated a hole.

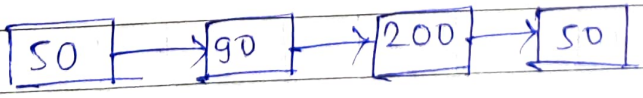
The idea behind the next fit is the fact that the list has been scanned once therefore the probability of finding the whole is larger in remaining part of list.

P1 = 90kB





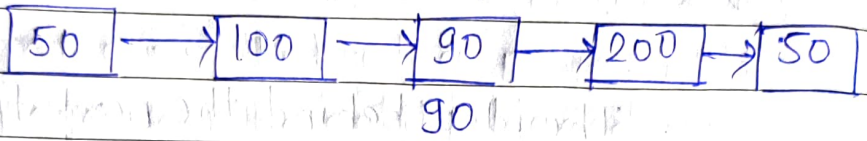
P2 = 10kB , starts from second whole where it previously allocated a space.



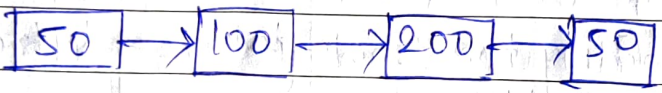
3) Best fit:

Allocates smallest hole that is big enough.

P1: 90KB



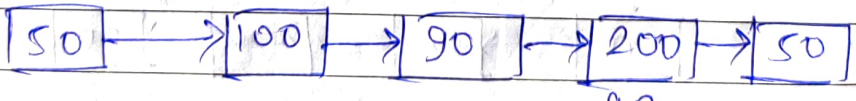
90



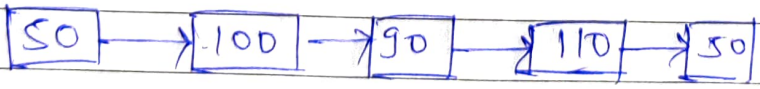
4) Worst fit:

Scans the entire list every time and tries to find out the biggest hole in the list.

P1: 90KB



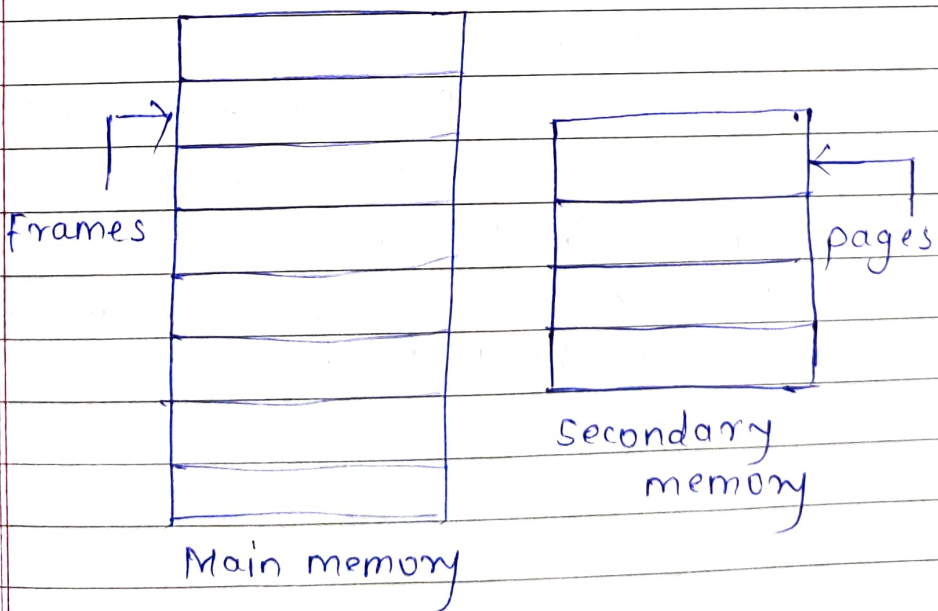
90



→ Paging in OS?

Pagig is a fixed sized memory allocation, storage and management scheme. Paging does two types of memory division.

1) The main memory is divided into small fixed sized blocks of physical memory which are known as frames. The main memory can be referred to as collection of frames.
2) The logical or secondary memory is divided into small fixed sized blocks which are known as pages.

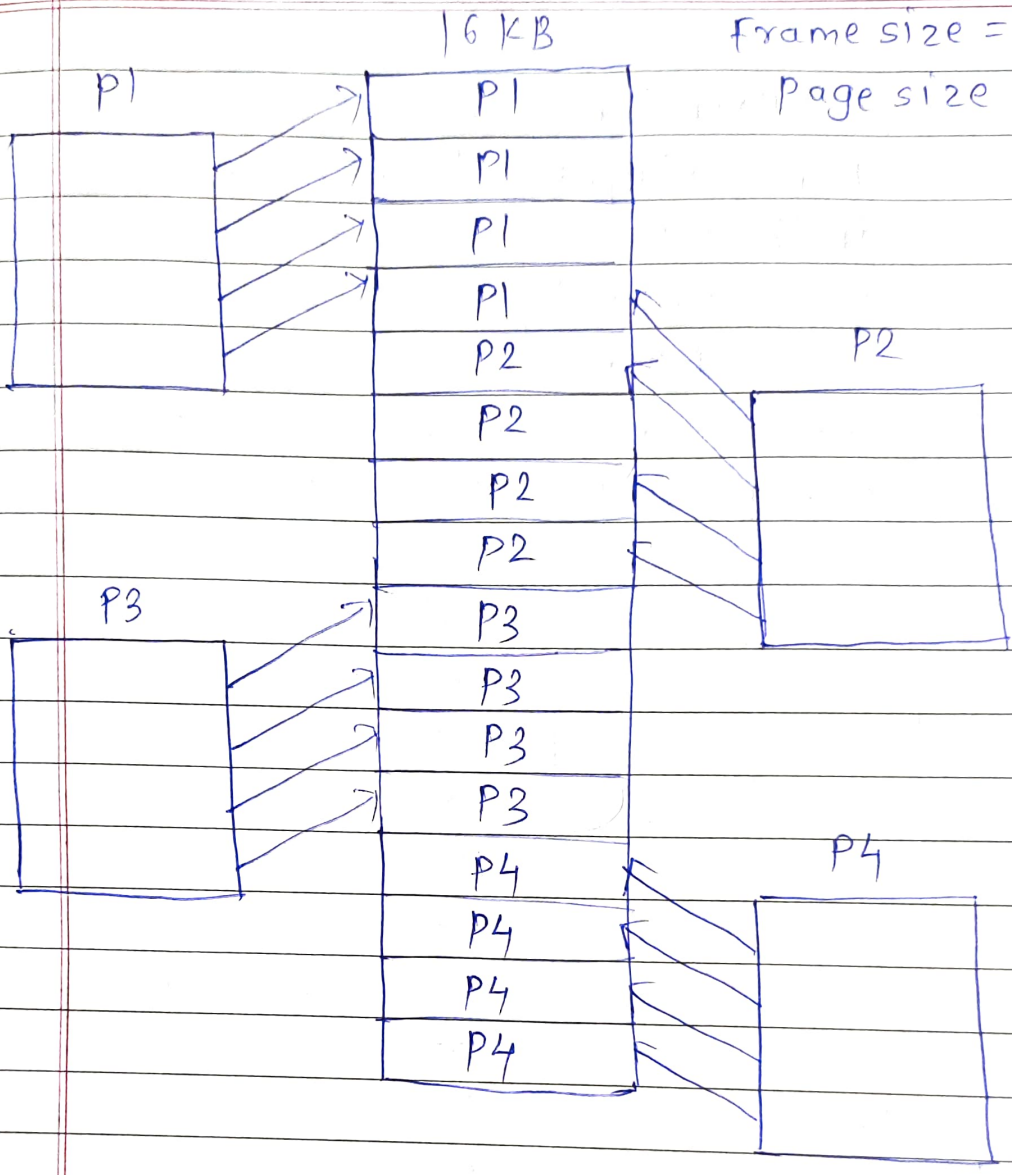


Paging keep track of all the free frames of main memory and load the pages of secondary memory or processes for the CPU to work with.

NOTE: The size of the frame is same as that of a page so that pages can be easily loaded into frames. The same size of frames and pages also help to maximize the memory and CPU utilization.

→ Example of paging in OS :-

1) Size of main memory = 16 KB
2) size of one process = 4 KB
3) size of one Frame and page = 1 KB
4) Theare are four processes P1, P2, P3 and P4 in the system to be execute
5) There is also a process namely P5 of 8 KB to be executed.



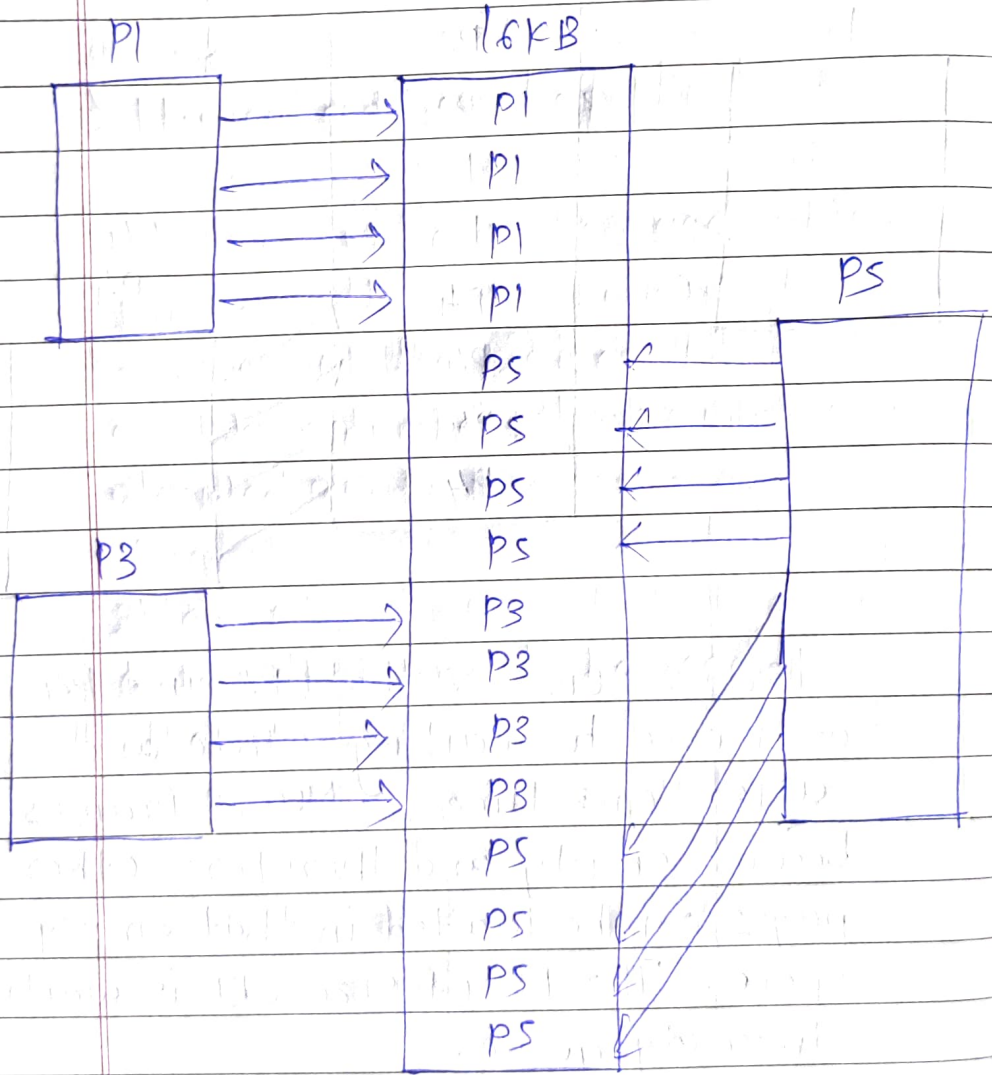
Let us consider that, P2 and P4 are moved to waiting state ~~and~~ after some time. Now, 8 frames become empty and therefore other pages can be loaded in that empty space. The P5 of size 8KB is waiting in ready queue.

classmate

Date

Page

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



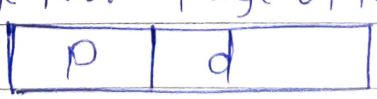
→ Translation of logical address into physical address :-

The address generated by the CPU is not ~~available~~ actual address in main memory. The address generated by the CPU is known as logical address and the address of the main memory is termed as physical Address.

The logical address generated by CPU has two parts:

1) Page Number : Page Number contains the base address of each page present in the physical memory. Page number is denoted with P.

2) Page offset : Page offset is the number of bits required to present a word of data into the page. Page offset is denoted with d.



The logical address is translated into the physical address by MMU. This translation of logical memory to physical memory is known as address translation. Address translation results in the actual location of the instruction present in the RAM.

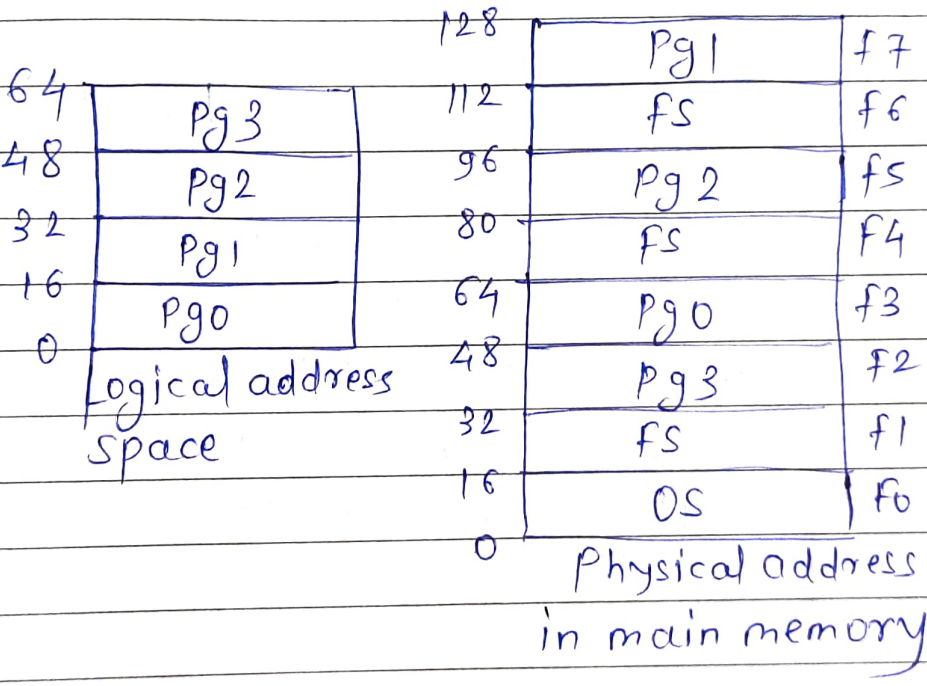
→ Physical address has two Parts:

1) Frame Number: The page table provides the corresponding base address of the frame which is known as frame number. Frame number is denoted as f.

2) Page offset: offset is the no. of words that have to be read from that page.

→ Page table :-

1) A data structure stores which page is mapped to which frame.
2) The page table contains the base address of each page in physical memory.



→ page table :

Logical Page No.	Physical frame No.
00 Pg0	F3 011
01 Pg1	f7 111
10 Pg2	f5 101
11 Pg3	f2 001

→ Each process has its own page table.

Ex :-

Suppose, process P0 is of 64 bytes. To uniquely identify each byte we require, $[2^6 = 64]$. 6 bits.

representation in bits

6 bits ⇒ 25 ⇒ 01 | 001

page no (p) | offset (d)

So basically, 25th byte is located in Page 1 and it's position is 9 bytes away from starting of Page 1.

Page 1 → Base = 16

offset = 9

Base + offset = 16 + 9 = 25

→ So, the instruction which are there in 25th byte of logical address is mapped to frame 7 in main memory.

So, the byte location is,

112 + 9 = 121

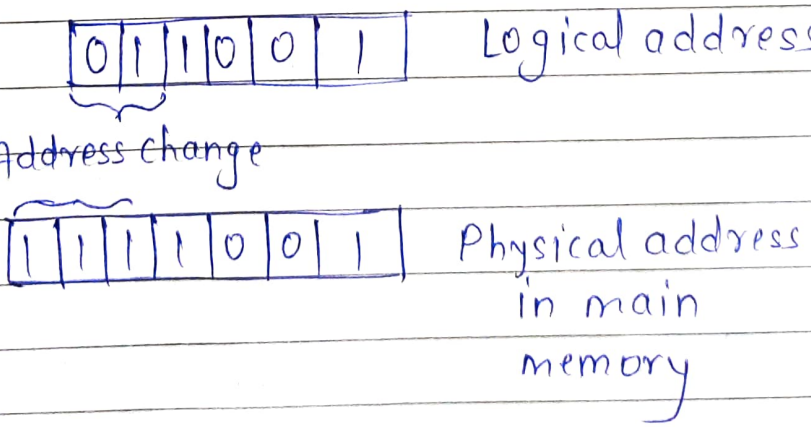
112 is the starting address of frame 7 and 9 is the offset.

→ 121 in bingary = | 111 | 1001 |

frame no (f) | offset (d)

Frame no = 111 in binary
7 in decimal

So, we can acknowledge that pg1 is mapped to f7 in main memory.



→ 1) Page table is stored in main memory at the time of process creation and its base address is stored in PCB.
2) A page table Register (PTBR) is present in the system that points to the current page table. changing page tables require only this one register, at the time of context switching.

→ Translation lookaside Buffer (TLB):-

→ Drawbacks of paging:

1) size of page table can be very big and therefore it wastes main memory.
2) CPU will take more time to read a single word from main memory.

→ How to decrease page table size:

1) The page table size can be decreased by increasing the page size but it

will cause internal fragmentation and there will also be page wastage.
2) Otherway is to use multilevel paging but that increases the effective access time therefore this is not a practical approach.

→ TLB :-

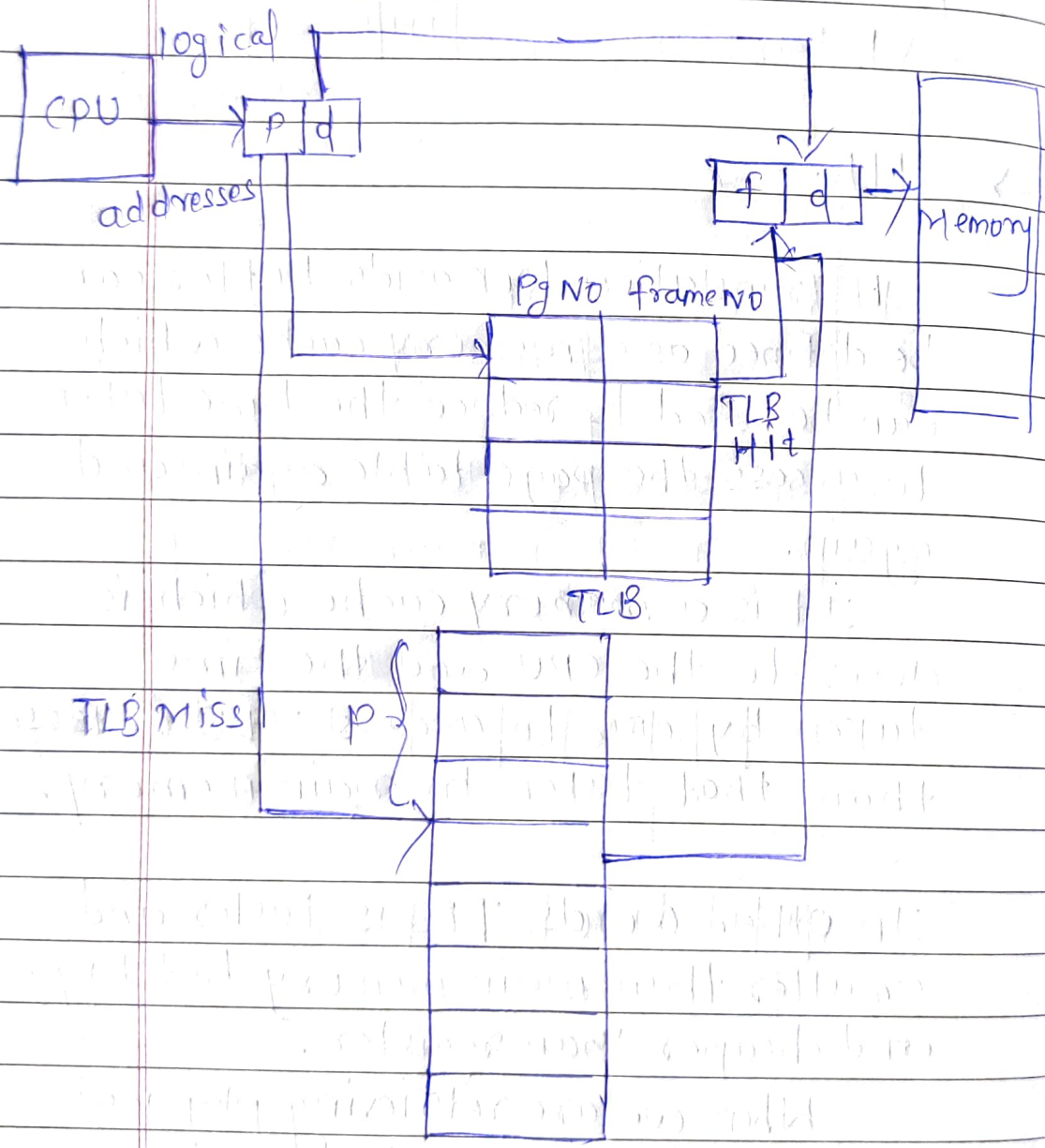
A Translation look aside buffer can be defined as a memory cache which can be used to reduce the time taken to access the page table again and again.

It is a memory cache which is closer to the CPU and the time taken by CPU to access TLB is lesser than that taken to main memory.

In other words TLB is faster and smaller than main memory but bigger and cheaper than register.

When we are retrieving physical address using page table, after getting frame address corresponding to page number, we put an entry

into TLB. So that next time, we can get values from TLB directly without referencing actual pages table.



So, initially when a process comes it looks for the page no in TLB to map that block in main memory if it is found so it is called as TLB hit, and if it is not then it is called as TLB Miss, and after that OS stores that page NO in TLB with respect to its frame no, so whenever again if it wants to access that location and if it is present in TLB it, can be done in lesser time.

But during context switching we know that the page table changes with respect to the process. Each unique process has it's own page table. But what about TLB when context switching occurs?

So, instead of flushing the TLB data, we add a one more block in TLB that is, ASID (Address space identifier) so ASID uniquely identifies each process and is used to provide address space protection and allow TLB to contain

entries for several different processes.

What is segmentation?

In Os, segmentation is a memory management technique in which the memory is divided into the variable size parts. Each part is known as segment which can be allocated to a process.

The details about each segment are stored in a table called segment table. Segment table is stored in one of the segments.

Segment table contains only two information about segment:-

1) Base: It is the base address of segment.
2) Limit: It is the length of segment.

classmate
Date
Page

→ Why segmentation is required ?

Till now, we are using Paging as our main memory management technique. Paging is more close to the OS rather than User. It divides all the processes into the form of pages regardless of the fact that a process can have some relative parts of function which need to be loaded into same page.

It may divide the same function into different pages and those pages may or may not be loaded at the same time into the memory. It decreases the efficiency of system.

It is better to have segmentation which divides the process into segments. Each segments contains the same type of functions such as the main function can be included in one segment and the library function can be included in other segment.

classmate
Date
Page

→ Segment Table:

~~The address~~

Segment Table is used to store information of all segments of the process. As we know, the CPU generates a logical address and for its conversion to the physical address a segment table is used. The mapping of a two dimensional logical address to the one dimensional physical address is done by using segment table.

→ segment table has two components:

1) segment base : The segment base is also known as base address of a segment. The segment base contains the starting physical address of segments residing in memory.

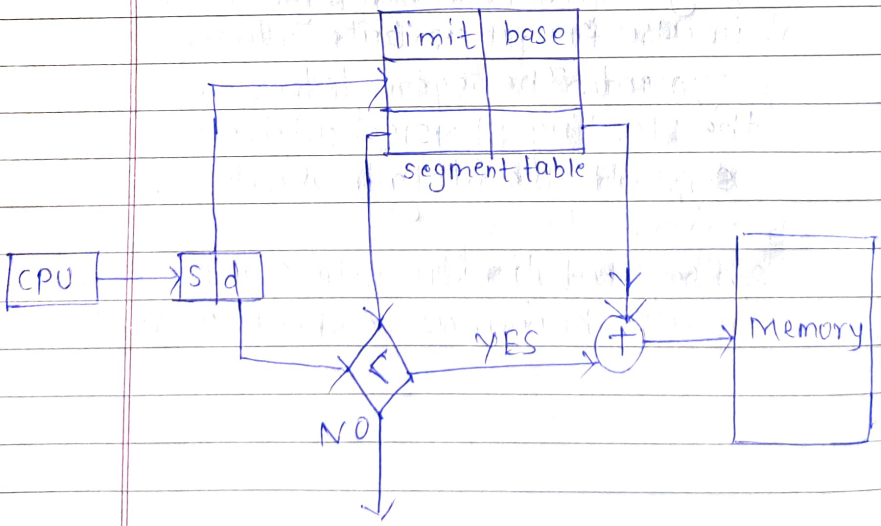
2) segment limit : The segment limit is also known as segment offset.

Segment Table:

	Limit	Base
Segment 0 →	1400	1400
1 →	400	6200
2 →	1100	4400
3 →	1300	4800

STBR: STBR stands for segment table base Register. STBR stores the base address of segment table.

STLR: STLR stands for segment table length Register. STLR stores the no. of segments used by a program.



Suppose, the logical address generated by CPU is,

01 111
s d

Value: 1 7

limit : 15

Base : 20

So, the address of that segment in main memory is 20+7=27

→ Advantages:

1) No internal fragmentation.
2) One segment has a contiguous allocation, hence efficient working within segment.
3) The size of segment table is generally less than the size of page table.
4) Results in more efficient system.

→ disadvantage:

External fragmentation

→ Virtual Memory in OS ?

Virtual memory is a storage scheme that provides user an illusion of having a very big memory. This is done by treating a part of secondary memory as main memory.

In this scheme, user can load the bigger size processes than the available memory by having the illusion that the memory is available to load the process.

Instead of loading one big process in main memory, the OS loads the different parts of more than one process in main memory.

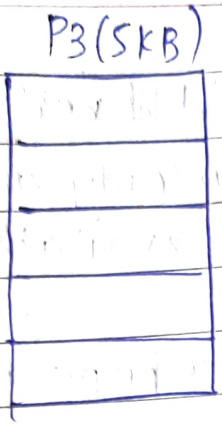
By doing theses, the degree of multiprogramming will be increased and therefore the CPU utilization will also be increased.

Programmer is provided very large virtual memory when only a smaller physical memory is available.

Demand paging is a popular method of virtual memory management.

Ex:

Suppose there are three processes P1, P2 and P3 for execution and the size of the processes are 6KB, 5KB and 5KB and one main memory of size 12KB.



Now only one 1KB of free space is available in memory. So we can allocate one Page of P3 to one frame in main memory.

But, when we allocate a full process to a memory, then all the pages of that process is not used at a time. Some page will be used by memory and some will be used later

So, in demand paging we remove the pages of the processes which are not to be used by memory at that time. So, these pages are stored at swap space in secondary memory.

So with these process the main memory is not full and other processes can also fit in main memory.

So, basically it increases the degree of multiprogramming of CPU. So, the pages of the processes which are used later are placed in swap space in secondary memory.

→ Virtual Memory = RAM + Swap space in secondary Memory

→ So, in demand paging the pages of a process which are least used get stored in secondary memory.

A page is copied to the main memory when its demand is made, or page fault occurs.

→ Page fault: page faults means when a page is which is required is not available in memory.

There are various page replacement algorithm which are used to determine the pages which will be replaced.

Rather than swapping the entire process into memory we use Lazy sweeper. A Lazy sweeper never swaps a page into memory unless that page will be needed.

We are viewing a process as a sequence of pages rather than one large contiguous address space, using the term sweeper is technically incorrect. A swapper manipulates entire processes, whereas a pager is concerned with individual pages of a process.

→ How demand paging Works?

1) When a process is to be swapped in, the pager guesses which pages will be used.
2) Instead of swapping in a whole process, the pager brings only those pages into memory. This, it avoids reading into memory pages

that will not be used anyway.

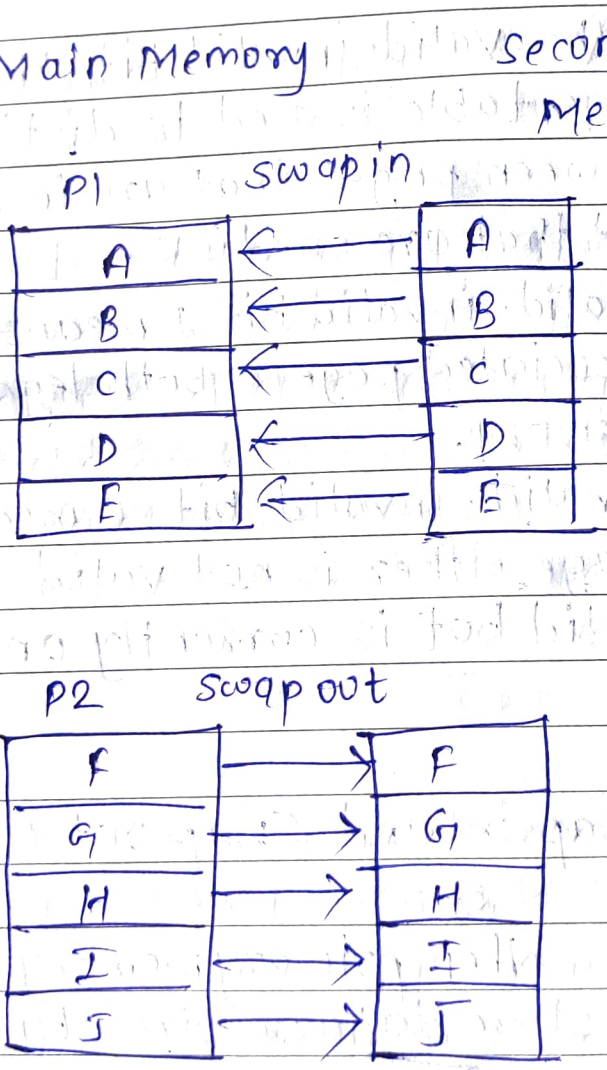
3) Above way, OS decreases the swap time and the amount of physical memory needed.

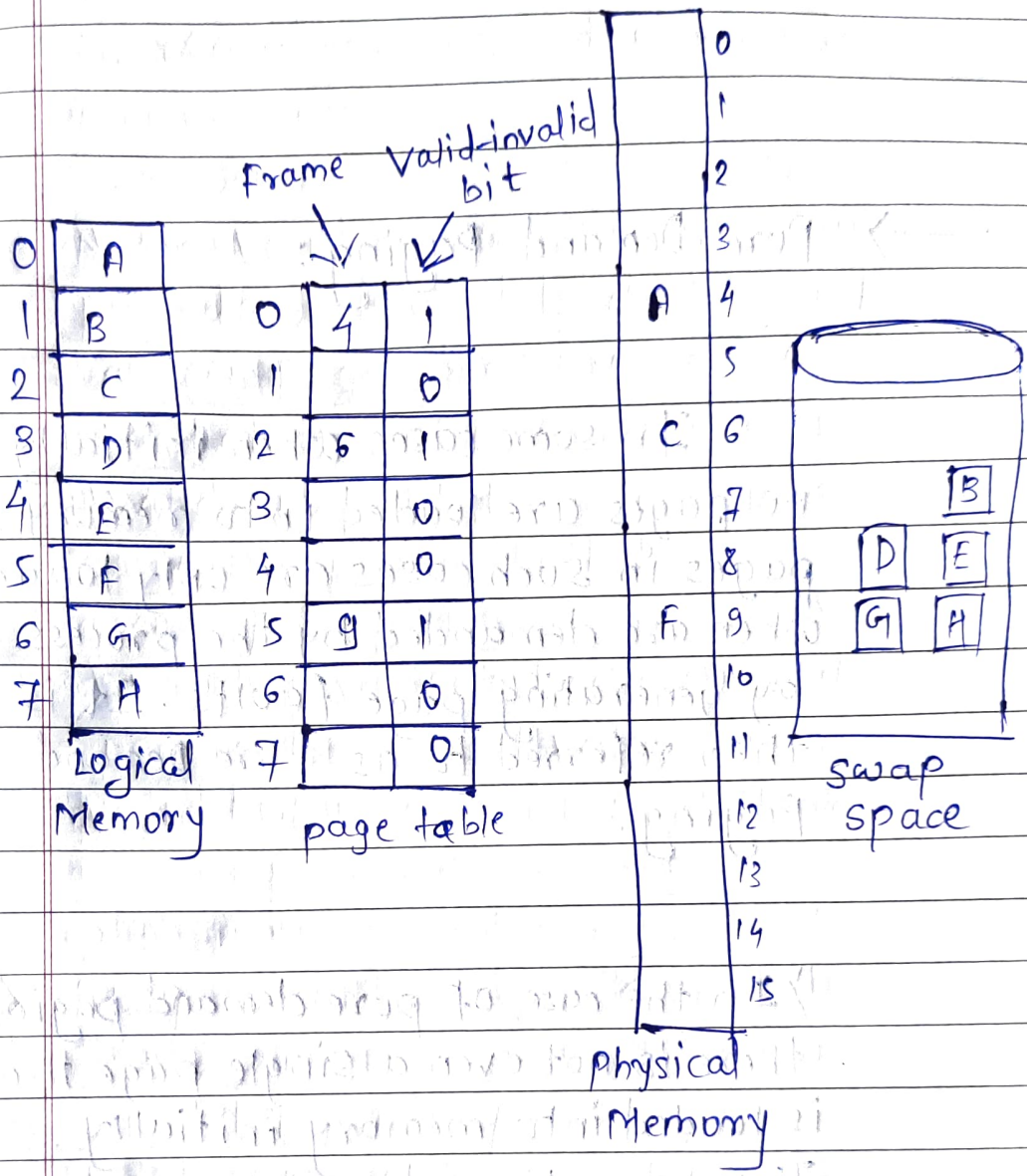
4) The valid-invalid bit scheme in the page table is used to distinguish between pages that are in memory and that are on disk.
i) valid-invalid bit 1 means, the associated page is both legal and in memory.
ii) valid-invalid bit 0 means, the page either is not valid ~~or~~ or is valid but is currently on the disk.

→ Swap in and Swap out!

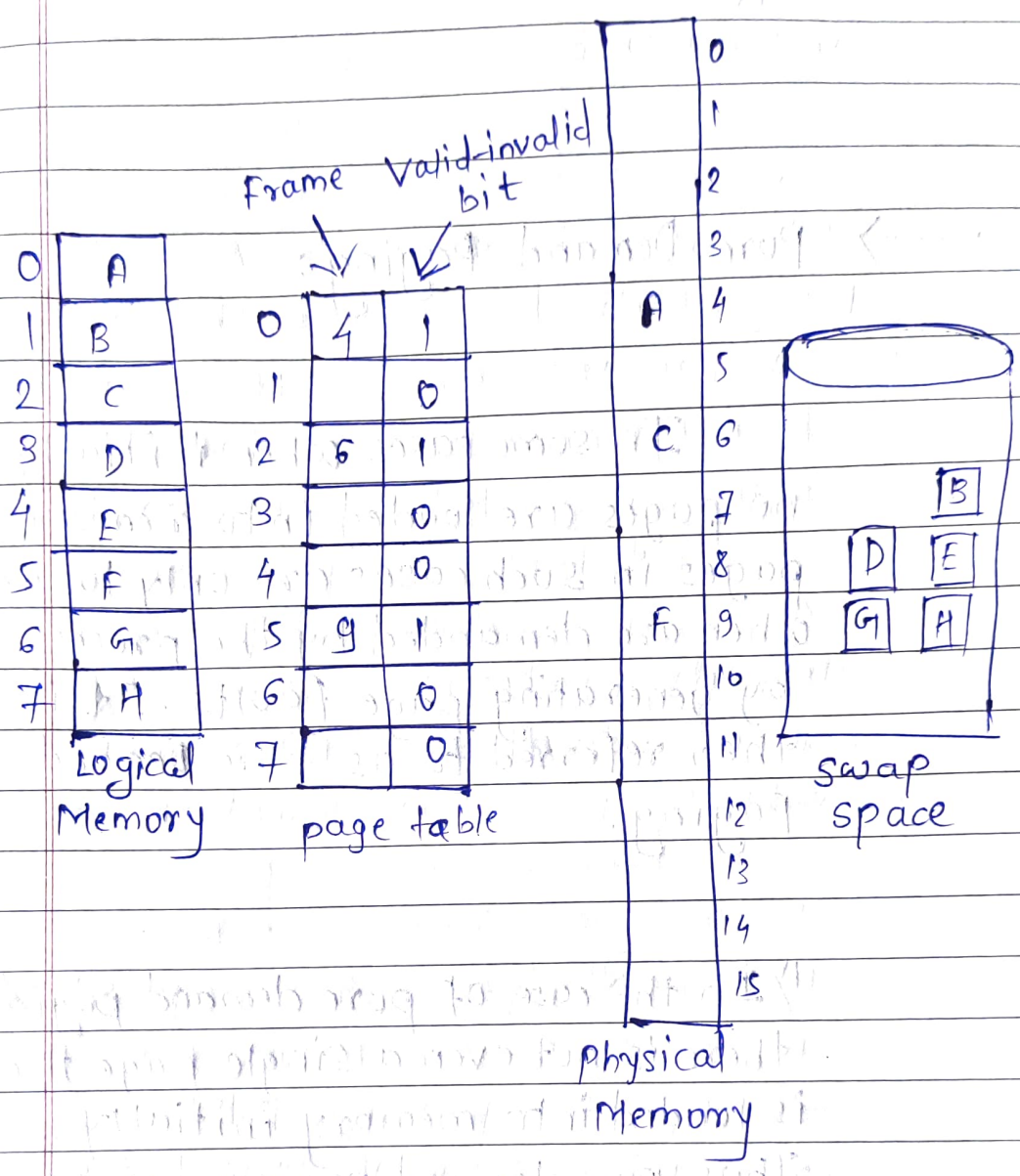
When primary memory is insufficient to store data required by several applications, we use a method known as swap out to transfer certain programs from RAM to hard drive.

Similarly, when RAM becomes available we swap in the applications from hard disk to RAM. We may manage many processes inside RAM by using swaps.





So, if the memory wants E page so it looks in page table and the value of valid-invalid bit is '0', so it will get that page from



So, if the memory wants E page so it looks in page table and the value of valid-invalid bit is '0', so it will get that page from

Swap space and then change the valve from '0' to '1' of valid-invalid bit.

→ Pure Demand Paging:

In some cases when initially no pages are loaded into memory, pages in such cases are only loaded when are demanded by the process by generating page faults. It is then referred to as Pure Demand paging.

1) In the case of pure demand paging there is not even a single page that is loaded into memory initially. Thus pure demand paging causes page fault.
2) When execution starts with no pages in memory, then the OS

sets the instruction pointer to the first instruction of the process and that is a non-memory resident page and then in case the process immediately faults for the page, and the page is brought in memory.

3) This scheme is referred as Pure Demand paging: means never bring a page into memory untill it is required.

→ Advantages of Virtual Memory:

a) Degree of multiprogramming will be increased.

b) User can run large apps with less real physical memory.

→ Disadvantage:

1) System can become slower as swapping takes time.

2) Thrashing may occur.

→

Paging	Segmentation
1) A page is of fixed size block.	1) A segmentation is of variable size.
2) It may lead to internal fragmentation	2) It may lead to external fragmentation.
3) In paging, the hardware decides the page size.	3) The segment size is specified by user.
4) A process address space is broken into fixed-sized block, which is called pages.	4) A process address space is broken in differing sized block called section.
5) The paging technique is faster ~~than~~ for memory access.	5) segmentation is slower than paging method.

⇒

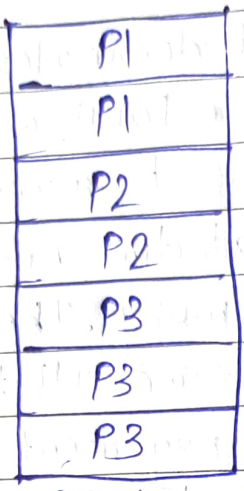
classmate
Date
Page

→ Page fault in OS?

Page fault dominates more like an error. A page fault will happen if a program tries to access a piece of memory that does not exist in physical memory. The fault specifies the OS to trace all data into virtual memory management and the relocate it from secondary memory to its primary memory, such as hard disk.

A page fault trap occurs when the requested page is not loaded into memory. The page fault primarily causes an exception, which is used to notify the operating system to retrieve the pages from virtual memory to continue operation. Once all of the data has been placed into physical memory, the program resumes normal operation. The page fault process occurs in the background, and thus the user is unaware of it.

ex: for ex,



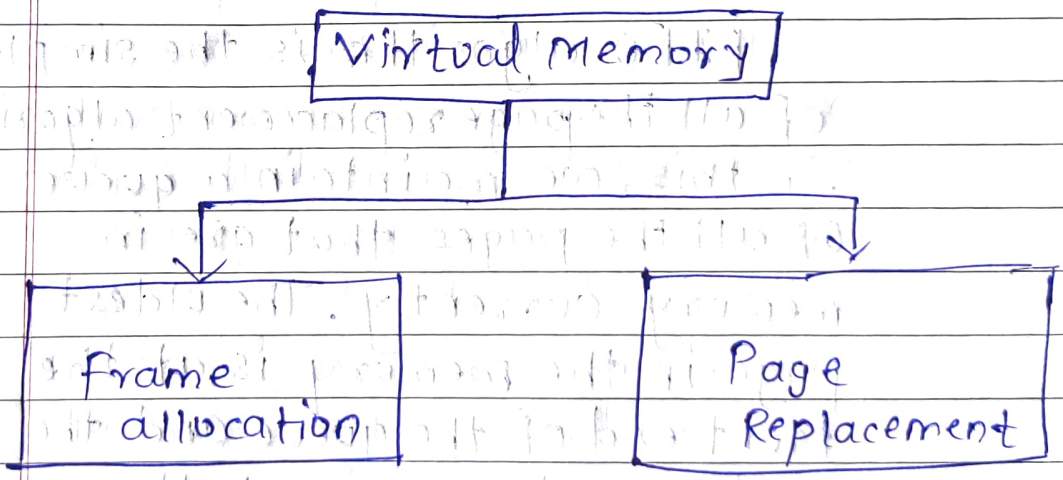
In RAM, we store the pages of the process which are to be used by the CPU, to increase the degree of multiprogramming.

Suppose if, P1 process is running and according to instruction of the program if we want to access page 3 of P1, and as we can see that page 1 and 2 are there in memory. so, we have to swape out one page of any process from memory and swap in the page 3 of Process1.

So to swap out which page should be remove from memory we use a algorithm called as Page Replacement Algorithm.

→ Page Replacement Algorithm :-

The page replacement algorithm decides which memory page is to be replaced. The process of replacement is sometimes called swap out or write to disk. Page replacement is done when the requested page is not found in the main memory.



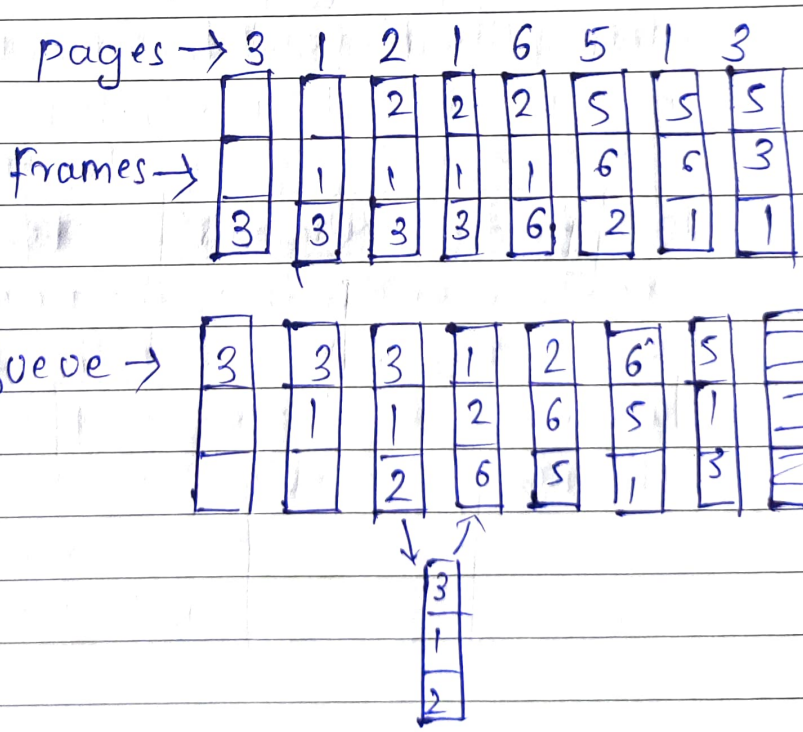
There are two main aspects of virtual memory, frame allocation and Page Replacement. It is very important to have optimal frame allocation and Page replacement algorithm. frame allocation is all about how many frames are to be allocated to a process while the page replacement is all about determining the page number which needs to be replaced in order to make space for requested page.

1) FIFO:

FIFO algorithm is the simplest of all the page replacement algorithms. In this, we maintain a queue of all the pages that are in memory currently. The oldest page in the memory is at the front end of the queue and the most recent page is at the back of the queue.

Whenever a page fault occurs, the OS looks at the front end of the queue to know the page to be replaced by newly requested page. It also adds this newly requested at the rear end and removes the oldest page from front-end of queue.

Ex: Consider the page reference string as 3, 1, 2, 1, 6, 5, 1, 3 with 3-page frames. Let's try to find out no. of page faults.



→ **Belady's Anamoly:-**

Proves that it is possible to have more page faults when increasing the number of page frames while using FIFO page replacement algorithm. for ex, If we consider reference string 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4 and 3 slots, we get 9 page faults and if we take 4 frames we get 10 page faults.

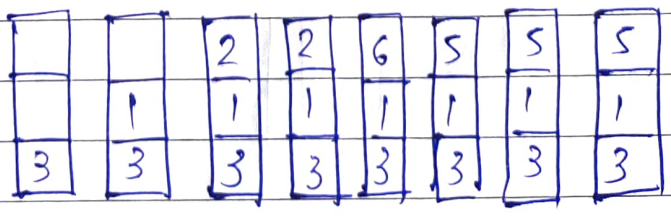
→ Optimal Page Replacement in OS :-

Optimal page replacement is the best algorithm as this algorithm results in the least number of page faults. In this algorithm the pages are replaced with the ones that will not used for the longest duration of time in future. In simplest terms, the pages that will be referred farthest in future are replaced in this algorithm.

Ex:

Let's take the same page reference string 3, 1, 2, 1, 6, 5, 1, 3 with 3 page frames as we saw in FIFO. This also helps you understand how optimal page replacement works the best.

pages → 3 1 2 1 6 5 1 3



3) Least Recently Used [LRU]

The least recently used page replacement algorithm keeps the track of usage of pages over a period of time. This algorithm works on the basis of the principle of locality of reference which states that a program has a tendency to access the same set of memory locations repetitively over a short period of time. So pages that have been used heavily in the past are most likely to be used heavily in future also.

In this algorithm, when a page fault occurs, then the page that has not been used for the longest duration of time is replaced by newly requested page.

Ex :

Pages	3	1	2	1	6	5	1	3
Frames	3	1 3	2 1 3	2 1 3	2 1 6	2 5 6	1 5 6	1 5 3

Also, it can be implemented by two ways,

1) Counter
   i) Associate time field with each page table entry.
   ii) Replace the page with smallest time value.

Input :-

	7	0	1	2	0	3
count	7	7	7	2	2	2
0→2,4,5		0	0	0	0	0
1→3			1	1	1	3
2→						
3→5						
4→						
7→1						

2) Stack based approach:

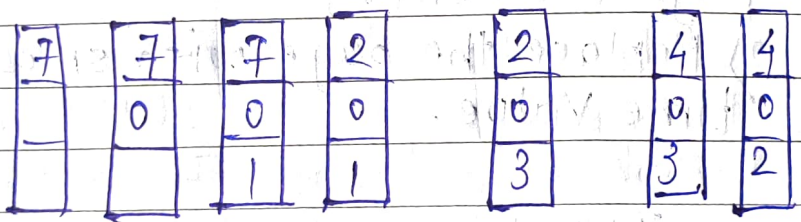
a) Keep a stack of page number.

b) Whenever page is referenced, it is removed from the stack and put on the top.

c) By this, most recently used is always on the top, and least recently used is always on the bottom.

d) As entries might be removed from the middle of stack, so doubly linked list can be used.

Ex: 7 0 1 2 0 3 0 4 2 3

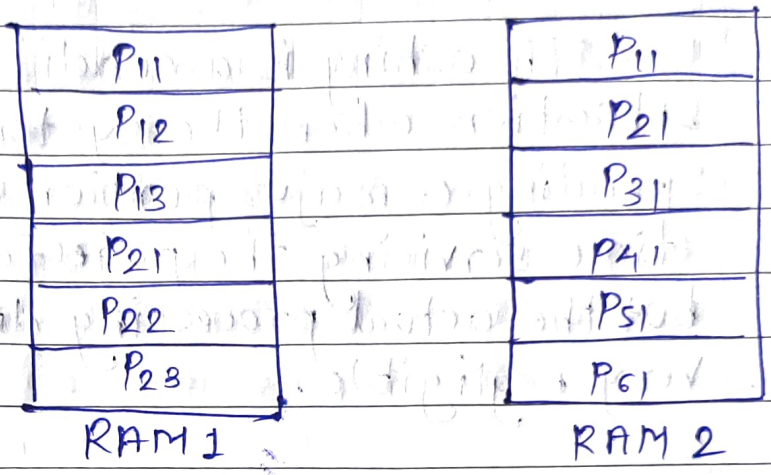


Stack:

2	0	3	0	4
1	2	0	3	0
0	X	2	2	3
~~7~~				

Thrashing :-

Ex :-



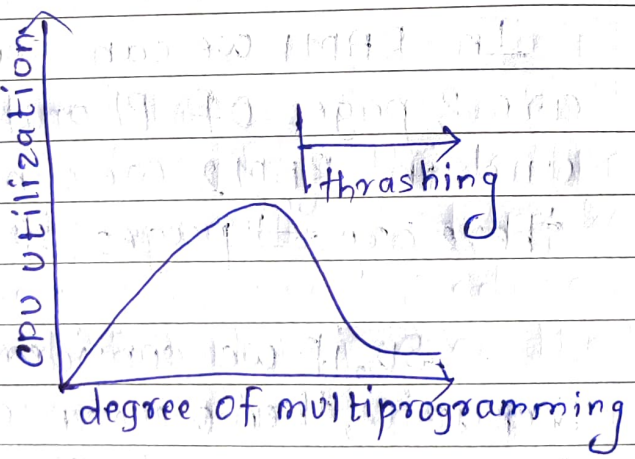
In RAM1 we can see that there are 3 pages of P1 and P2 processes and in RAM2 we can see that there are ~~all~~ one pages of each process.

So, if we consider these scenario, then we can see that in RAM1 there will be less page fault problem and in RAM2 there will be high degree of multiprogramming as there are each process having one of it's page in

Memory, and also there will be more page fault.

→ Thrashing:

Thrashing is a condition or situation when the system is spending a major portion of its time servicing the page fault, but the actual processing done is very negligible.



→ causes of thrashing :-

1) If cpu utilization is too low we increase the degree of multiprogramming by introducing a new system. A global page replacement algorithm is used. The cpu scheduler sees the decreasing cpu utilization and increases degree of multiprogramming.

2) As degree of MP increases, CPU utilization also, increases.

3) If the degree of MP is increased further, thrashing sets in, and CPU utilization drops sharply.

4) so, at these point to increase CPU utilization and to stop thrashing, we must decrease the degree of MP.

→ Algorithms during thrashing :

1) Global Page Replacement Algorithm (GPR) :

Since, GPR can bring any page, it tries to bring more pages whenever thrashing is found. But what actually will happen is that no process gets enough frames, and as a result, the thrashing will increase more and more.

2) Local page Replacement algorithm (LPR) :

Unlike GPR, LPR will select pages which only belong to that process. So there is a chance to reduce thrashing. But it is proven that there are many disadvantages if we use LPR.

→ Techniques to prevent Thrashing :-

→ Locality Model :

A locality is a set of pages that are actively used together. The locality model ~~needs~~ states that as a process executes it moves from one locality to another.
A program is composed of several different localities which may overlap.

For ex, when a function is called, it defines a new locality where memory references are made to the instructions of function call, it's local and global variables, etc. Similarly when the function is ~~called~~ exited, the process leaves the locality.

→ Technique :

1) Working set model :
i) This model is based on the concept of the locality model.
ii) The basic principle states that

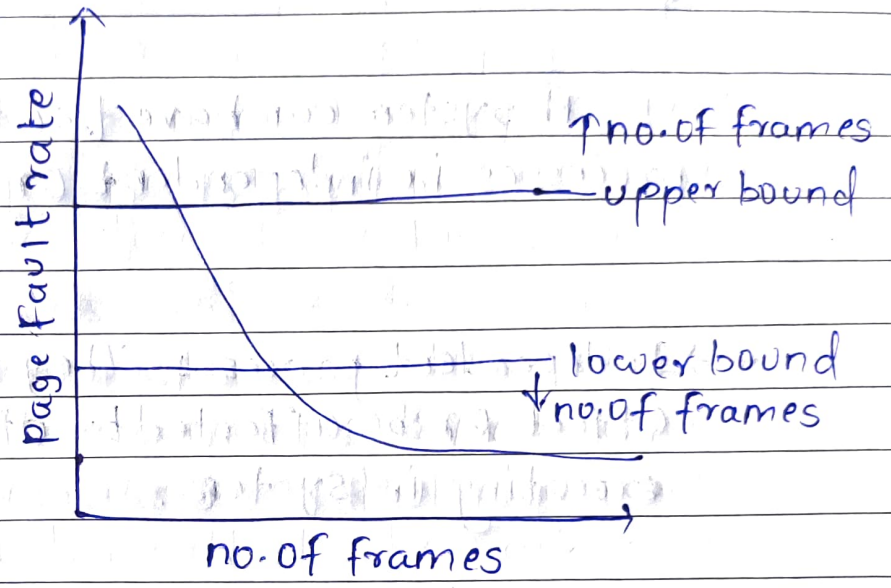
if we allocate enough frames to a process to accomodate its current locality, it will only fault whenever it moves to some new locality. But if the allocated frames are lesser than the size of current locality, the process is bound to thrash.

ii) Page fault frequency:

1) Thrashing has a high page fault rate
2) We want to control the page fault rate.
3) When it is too high, the process needs more frame. Conversely, if page fault rate is too low, then the process may have too many frames.
4) We establish upper and lower bounds on the desired page fault rate.
5) If page-fault rate exceeds the upper limit, allocate the process another frame, if page-fault rate

fails Falls below the lower limit, remove a frame from process.

6) By controlling Page fault rate, thrashing can be prevented.



Interprocess communication

IPC is a type of mechanism usually provided by the OS. The main aim or goal of this mechanism is to provide communications in between several processes.

A system can have two types of processes i.e independent or cooperating.

→ Independent process :- They cannot affect or be affected by other processes executing in system.

→ Cooperating process :- They can affect or be affected by other processes executing in system.

* Any process that shares data with other processes is called co-operating process.

co-operating processes require an IPC mechanism that will allow them to exchange data and information.

→ There are two fundamental models of IPC:

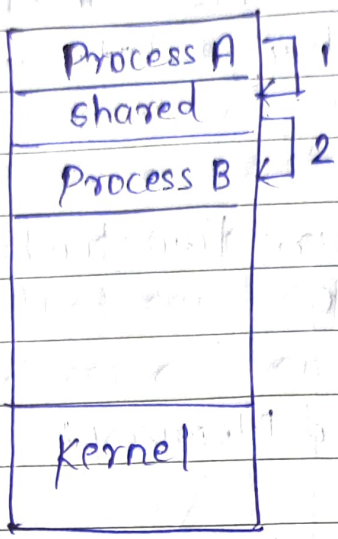
1) Shared Memory:

In shared Memory model, a region of memory that is shared by co-operating processes is established.
processes can then exchange information by reading and writing data to the shared region.

2) Message passing:

In Message passing model, communication takes place by means of messages exchanged between co-operating processes.

→ Shared Memory systems :-



1) IPC using shared memory requires communicating processes to establish a region of shared memory.

2) Typically, a shared memory region resides in the address space of the process creating the shared memory segment.

for ex: If process A, has to communicate with process B, then a shared memory will be created in the address space of process A.

3) other processes that wish to communicate using this shared memory segment must attach it to their address space.

for Ex: If process B, also wants to communicate with process A then they have to attach ~~this~~ the address space of process A with process B.

4) Normally, the OS tries to prevent one process from accessing another process's memory.

5) Shared memory requires that two or more processes agree to remove this restriction.

→ Producer-consumer problem

A producer process produces information that is consumed by a consumer process.

1) one solution to the producer - consumer problem uses shared memory.

2) To allow producer and consumer processes to run concurrently, we must have available a buffer of items that can be filled by the producer and emptied by consumer.

3) one soln to producer consumer problem uses shared memory.

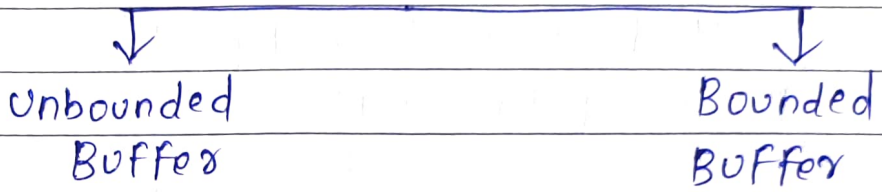
4) To allow producer and consumer processes to run concurrently, we must have available a buffer of items that can be filled by the producer and emptied by consumer.

5) This buffer will reside in a region of memory that is shared by producer and consumer processes.

6) A producer can produce one item while the consumer is consuming another item.

7) The producer and consumer must be synchronized, so that the consumer does not try to consume an item that has not yet been produced.

→ Two kinds of Buffer



Places no practical limit on the size of the buffer. The consumer may have to wait for new items, but the producer can always produce new items.

Assume a fixed buffer size. In this case, the consumer must wait if the buffer is empty, and the producer must wait if the buffer is full.