Module - 1

→ Introduction to operating system.

→ Program – set of Instructions / Statements (Implements a task)

→ Software - set of related programs – Implements a set of activities / tasks

1) System software
2) Application software

System software

① SW which provides services to other software in computer System.
Ex: OS, compiler, driver, Interpreter
each programing language has its own compiler.

② Installed along with OS.

③ To control the operations of Hardware components.

④ Users cannot interact with System software.

⑤ Provides a platform for running applications software.

Application software

① It is a kind of software which depends on system software for services is called application Software.
Ex: browser, VLC player, MS word.

② Installed based on user required after installing OS.

③ To implement specific activities of the users.

④ Users can interact with Application software.

⑤ cannot run without system software.

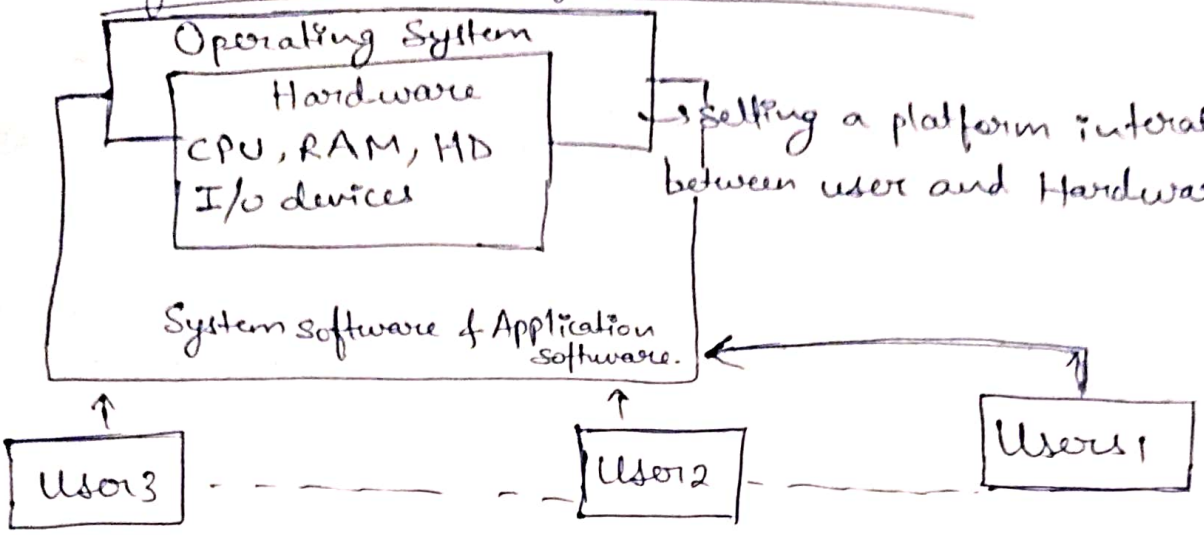
Operating system

→ It is a system software.

Role of operating system in a computer system.

→ To make a platform for the interaction of user and hardware.

* Logical division of computer System



OS provides an interface (platform) for interaction b/w user & hardware

User1 Java program

1) Editor – Application software
2) Keyboard - H/W component
3) Save – HD – HW
4) Compile – RAM – Javac, CPU
5) Interpreter
6) Monitor

* Views of operating System
  - Related to role of OS

1) User view
2) System view

User view (Role of operating System)

→ 1) Personal computers
2) Thin clients
3) Client-server systems
4) Hand devices

1) Personal computers

→ are the computer systems which are used by single users.

Key roles are:

1) Easy use of the computer system
Effective
2) ↑ Utilization of services
3) ↑ Performance of computer system
Improving

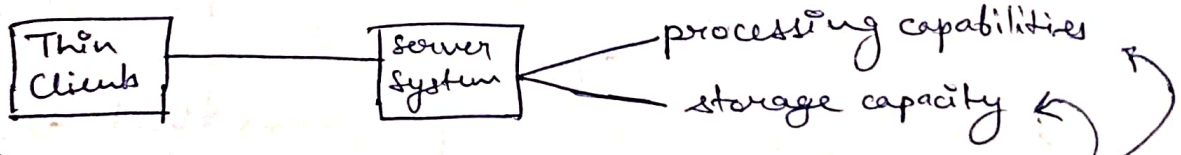
---

2) Thin clients

→ The computer system without processing capabilities and storage capacity.

→ An operating system is running in thin clients
Thin client is connected to server system

1) Utilization of services

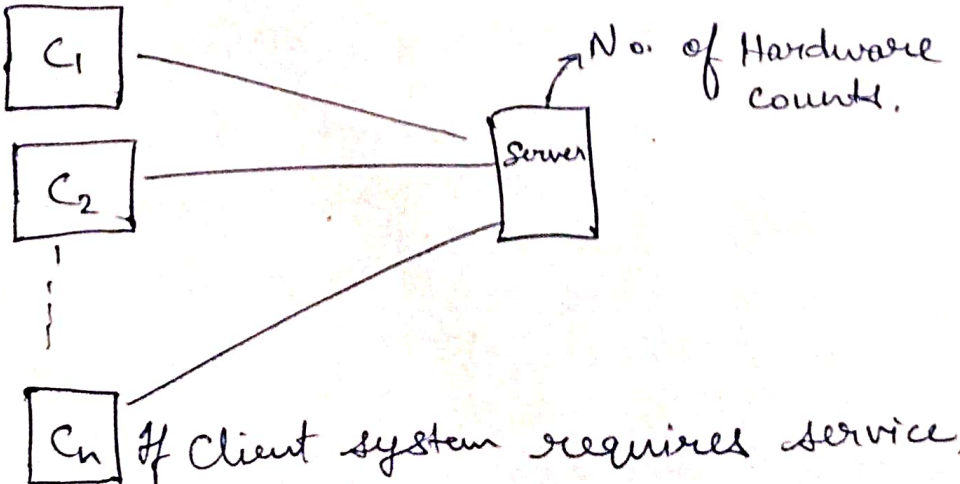


OS focusses on server system in thin client for

---

3) Client - Server Systems

No. of Client Systems



If Client system requires service, it request to server system.

1) Any client system requests to server system
2) server can request to any client.

*OS runs in any of the client.

→ Utilization of resources is available at both clients as well as servers.

---

4) Home Devices

→ Small computer systems

1) Microwave oven, Washing machines

↓

OS runs

↓

→ focusses on automatic working of computer system.

Role of Operating System from System point of view

↓

Computer system

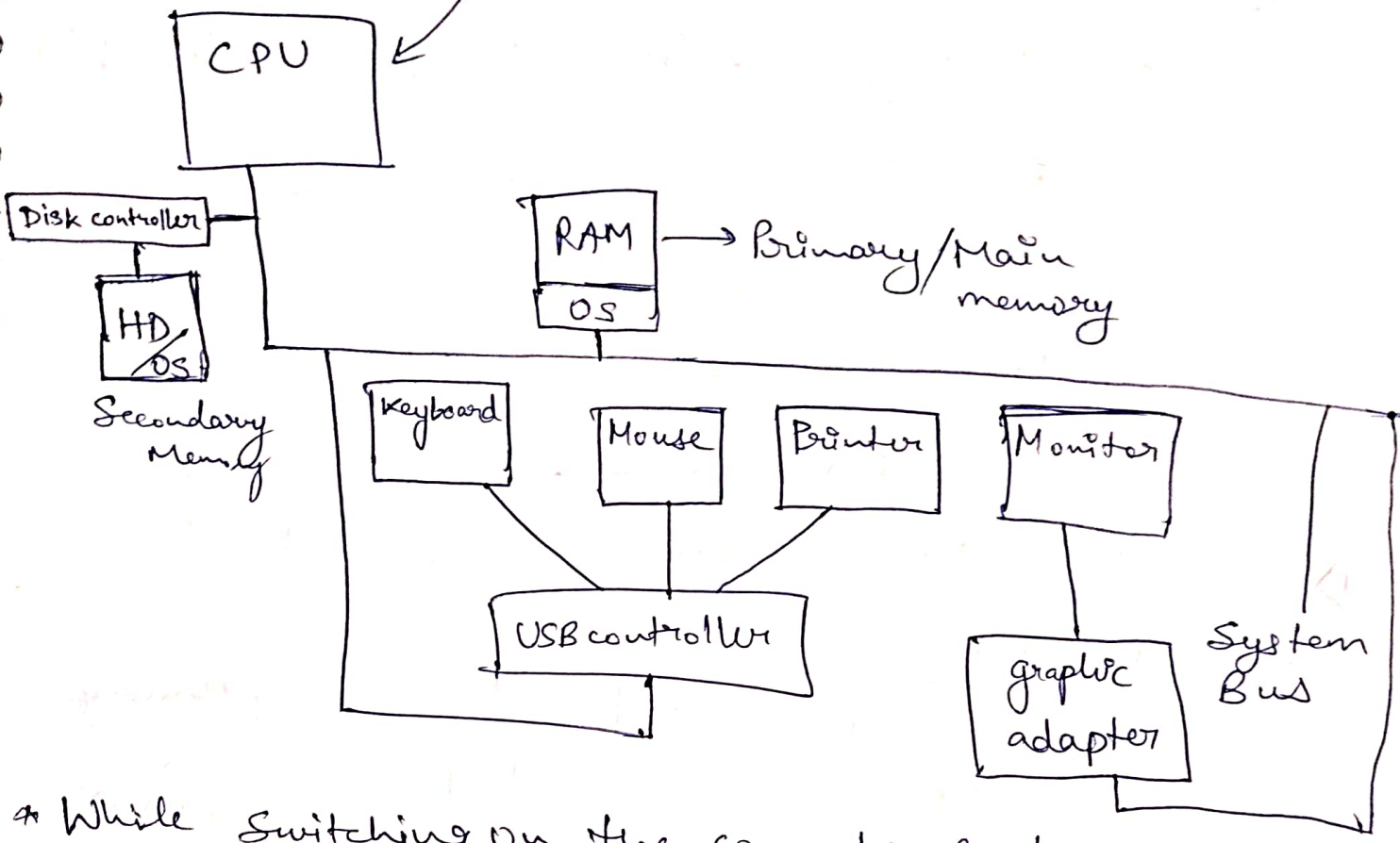
↑

Hardware/Software component

1) Allocating Resources to programs that are running in the computer system.

Computer System organization

1) Major Hardware components related to CS



* While Switching on the computer system

1) Bootstrap program is activated.
   - present in → ROM
   - Identifies OS in HD

OS → HD → RAM

→ In RAM ~~Exec~~ execution of OS starts.
then whole CS will get operated by OS

Operating system operations / functions

Major operations performed by OS are:

1) Process Management
2) Memory management
3) File Management
4) I/O device management
5) Protection
6) Security
7) Networking.

1) Process Management

→ Process – A program in execution state

1) Creating & deleting the process done by OS
2) Suspending & Resuming the process
3) Providing Synchronization
5) Handling deadlocks
6) CPU scheduling ⇒ roles done by OS related to process Management
7) Providing communication

1)

* Create a process

1) Move the program from Secondary Memory (copy) to primary Memory (RAM)
2) Allocating the CPU to the program in RAM by OS

* Deleting a process

1) OS removes the complete program from RAM to Harddisk

(P.M) ⟶ (S.M)

---

2) Suspending & Resuming the process

1) Suspending the process

→ First the output will get generated by Os and at another command it will suspend the program to execute another program.

2) Resuming a suspended process

```
Class A {
    psvm ( ) {
        ===
        ===
        ===
        output
        sc.next();
        ===
        ===
        ===
    }
}
```

output → suspended (CPU →)

sc.next(); → Resumed

→ OS has to submit output to CPU while suspension of program.

then after suspension OS restores the command to CPU

3) Providing Synchronization

1) Sharable
2) Non sharable

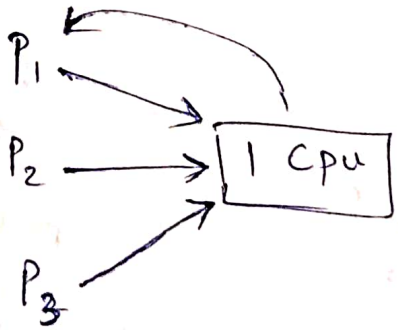
→ can be used by no. of programs at a time.

→ only one program can use at a time.

Sharable resource Eg: Harddisk, RAM

Non sharable Eg: CPU, printer

→ OS has to provide synchronous access to non sharable resource



7) Providing communication

$P_1 \longleftrightarrow P_2$

1) Share the data
2) Access the data

Shared memory
Message passing

---

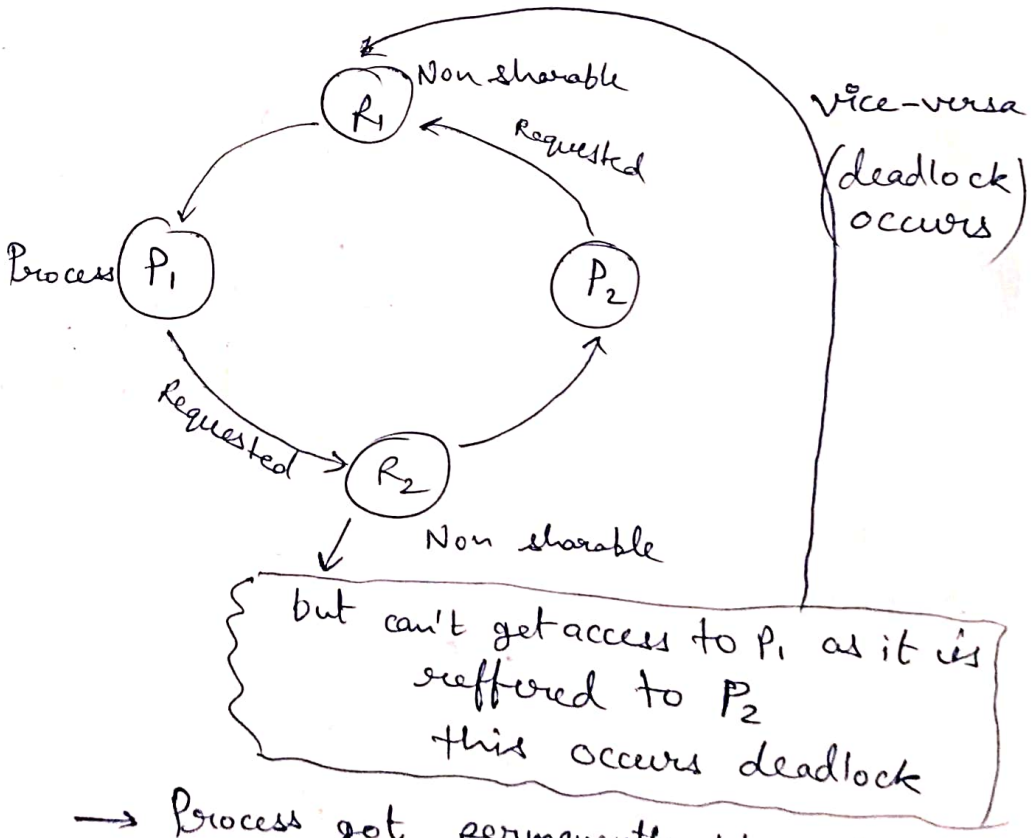
5) Handling deadlock

→ **Deadlock** – **Permanent blocking** of the process

↓

Program (No response to CS)
←(ie: deadlock occur)

→ OS has to avoid that deadlock

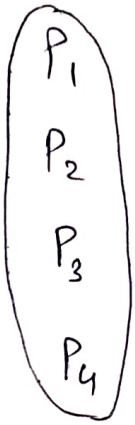


→ Process got permanently blocked and OS will avoid that deadlock by stopping that process.

* How OS will avoid deadlock?

→ It will verify the request sent by process.

6) CPU scheduling



1CPU

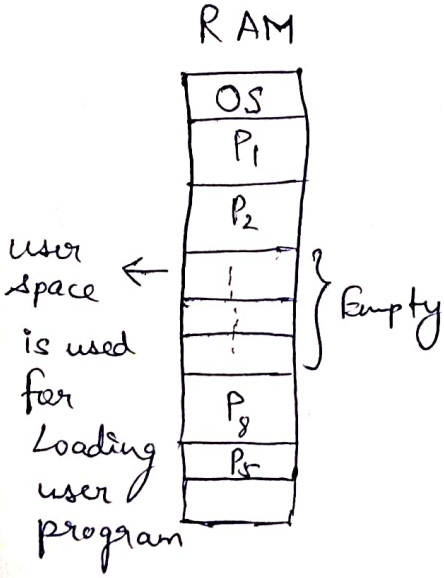
→ CPU will ~~not~~ switch one after one completion of process

algo like: FCFS.

Memomery Management (2nd function of OS)

(Major memory components)

1) Main memory / Primary memory / RAM
2) Secondary memory / HD



1) Main Memory Management

1) Knowing which parts of RAM are free

OS will verify the free spaces in RAM

2) Allocating free spaces in the RAM

OS will get info about free spaces then it will allocate the programs

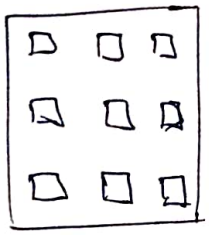
- Contiguous
- paging
- Segmentation

3) Delocating the space in the RAM

4) Deciding which program to be moved b/w RAM and HD

2) Secondary Memory Management :-

Used Hard disk as secondary Memory



HD

→ Hard disk is divided into no. of blocks in logical view.

→ Each block is identified by numbers or address

Activities :

1) Free space Management
2) Storage allocating
3) Disk scheduling

→ Time to time, hard disk manages to stores information about free space. There are 4 types of methods to main free space information.

→ Bit vector
→ Linked List
→ Grouping
→ counting.

→ Users wants to store any data and storing data is used by the method.

→ contiguous
→ Linked
→ Indexed.

3) Disk scheduling

→ Read data from system and then scheduling the disk with different methods:-

FCFS	e-SCAL	different algorithms
SSTF	COOK	
SCAN	C-COOK	

File Management

File → container of information in a program or database.

Directory/Folder → container of files and sub direction.

Activity :-

1) Creating file: create a new file in operating system.

2) Deleting file: Supporting delete a file that stores in ~~create folder~~

3) Sorting file: ~~Delete the folder~~ create folder

4) Deleting folder: Delete the folder.

5) Permitions for managing file or folder :-
transfer from one folder to another folder.

6) Backing up the files/folder.

→ Duplicate copy of data/file in other drive C, D or E. Eg: stable storage devices

→ maintaining copy of data.

I/O device management

1) Input
2) Output
} devices

Ex: PC, printer

OS sends to drivers as an software to connect with devices controller to control to provide setup to software & hardware devices.

1) Networking

→ A network is a collection of systems to each other.

The OS which is used in computer systems which is connected to a network must provide support for

1) Sharing the resources available in the network.
2) Dividing the program into parts and distributing them to different systems ~~int~~ in the network.
3) Providing transparency to the users.

2) Protection & Security

↳ restricting access to the resources of the system from the ~~resources~~ processes that are running in the system.

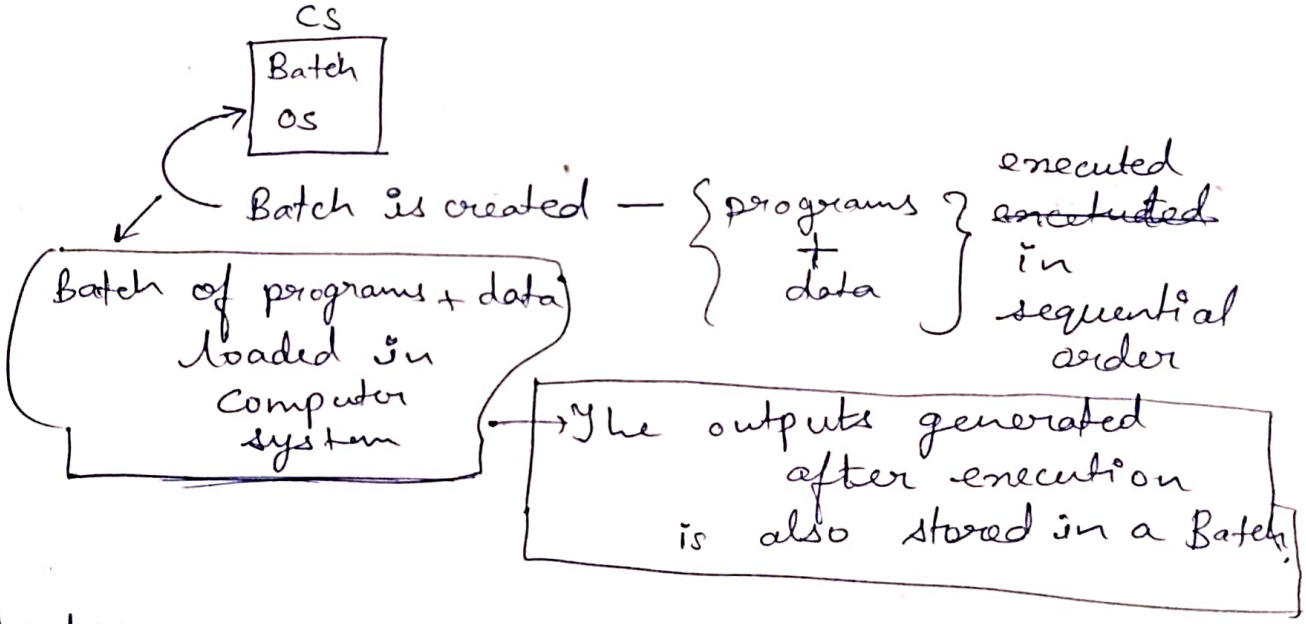
Ex: 2 or more process request the CPU at a time then OS will restrict the access to the CPU by allocating the CPU to only one process at a time.

Security :- restricting access to the system by the users

Types of Operating system

1) Batch processing OS
2) Multi programming OS
3) Time sharing OS
4) Distributed OS
5) Real time OS
6) Embedded OS
7) Mobile OS

1) Batch processing OS – It is the oldest OS → Eg: z/OS – IBM (downloaded by)



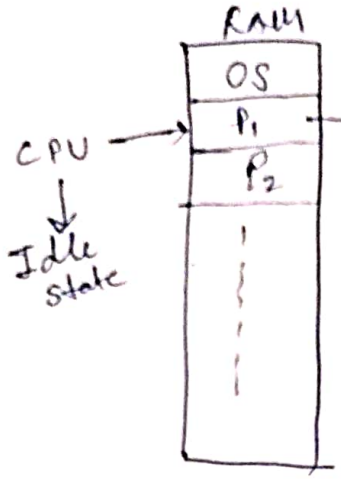
Drawbacks:
1) Concurrent execution of programs is not allowed.
2) User interaction is not allowed with the program.

---

2) Multiprograming OS – Windows, LINUX Examples

↓

It loads no. of programs in RAM at a time



I/O operation

Drawback will

→ CPU must be in Idle state after execution of only one program.
(performance decreases)

2) Doesn't allow user interaction with program during execution.

Multiprogramming OS switched CPU in two cases

1) current program is in waiting state
2) when current program execution is completed.

---

3) Time sharing OS - It also loads no. of programs at a time into RAM.

→ Each process is allowed to execute for a certain amount of time.

→ It uses a parameter time slice for the execution of each program at a particular range of time

→ If Time slice = 5 sec
and P_1 takes 20 sec
it will switch program execution after 5 sec.

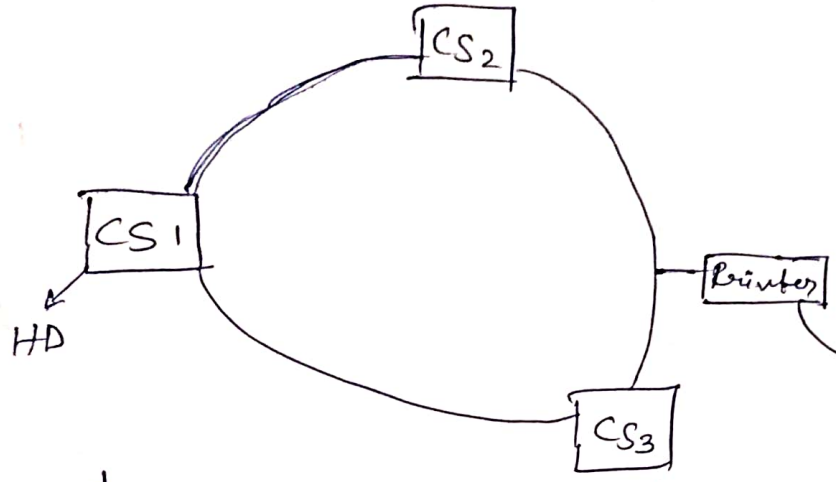
CPU switches
① waiting state
② completion of program
③ alloted time is completed.

→ Time sharing enables equal chance for CPU to process each ~~prog~~ process.

~~Additional~~

4) Distributed OS - Ex: Amoeba, LOCUS

→ control operations of set of computer S.



Features

1) Sharing of Resources

It is used by all CS

2nd feature - Reliability of Resources

→ If one cs is failed then other cs can use its resources.

3) speed up of computations.

Program → 30 minutes → It will ~~be~~ distributed to all the set of computer system to execute a large program.
So that it get executed in less amount of time.

4) Providing Transparency.

5) Real time OS - Windows CE, Symbian

Real time applications

→ Fixed time constraints on execution.

Ex: ~~sta~~ satellite sends data for every 100 sec

and a [CS] → received data from satellite and stores it in CS.

running an application to

the time constraint to store it is 99 sec

1 to 100 sec.

→ These CS has Real time OS which executes Real time applications

---

6) Embedded OS - WINDOW CE, Free BSD

→ It is used in small CS (Embedded computers)

↓

Limited resources

→ Extreamely efficient & very compact

---

7) Mobile OS - Ex: Android, IOS, Blackberry OS.

↑ Google ↑ Apple

→ Wireless communication b/w the devices

→ mobile applications.

System Calls

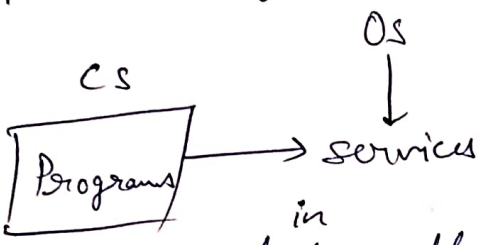
→ Function/Method in any programing language

```
main ( )
{
  =
  =
  =
}
```

→ System calls is used when in a programs to call services
Ex: Reading data from keyboard is an example of system calls.

opening a file is a service
Writing data file is a service
Closing data file " " ".

During execution of program it ~~requests~~ provides no. of services which is processed by OS.



in
System calls developed, Assembly language, C & C++

→ System calls are executed in Kernel of OS.

→ Relation between API & system calls

Application programing Interface

JAVA API

imported pakages like io, & Scanner
Packages contains classes and Interfaces.

→ methods & variables

System.out.println

System call Interface ← user mode / Kernel mode

pre-defined function goes under
1) Identifying the function call

Types of System calls

1) Process control system calls
2) file management system calls
3) Device " "
4) Information Maintainence system calls
5) Communication " "
6) protection " "

1) # Process control system calls

1) end, abort
2) Load, execute
3) Create process, Terminate process — Parent process ↓ Child process
4) wait, Signal
5) get process attributes, set process attributes.

2) File Management system calls

1) It is identified based on operations work on the file. (open, close, create, delete, read, write, i.e position indicator by the file. (file pointer).
2) Get file attributes, set file attributes.

3) Device management System

1) Request device and release device

2) read & write

↓ Input device

↓ output device

4) Information maintenance system calls

1) get date on time, set data on time

2) get system data, set system data.

↓

- no. of users
- version of OS
- Free space in RAM, HD

5) Communication Maintenance system calls

Mod 2

Process - A process is a set of ~~inst~~ instructions which collectively implements a task.

A process is a program being executed

Program resides in secondary memory

Process resides in primary memory

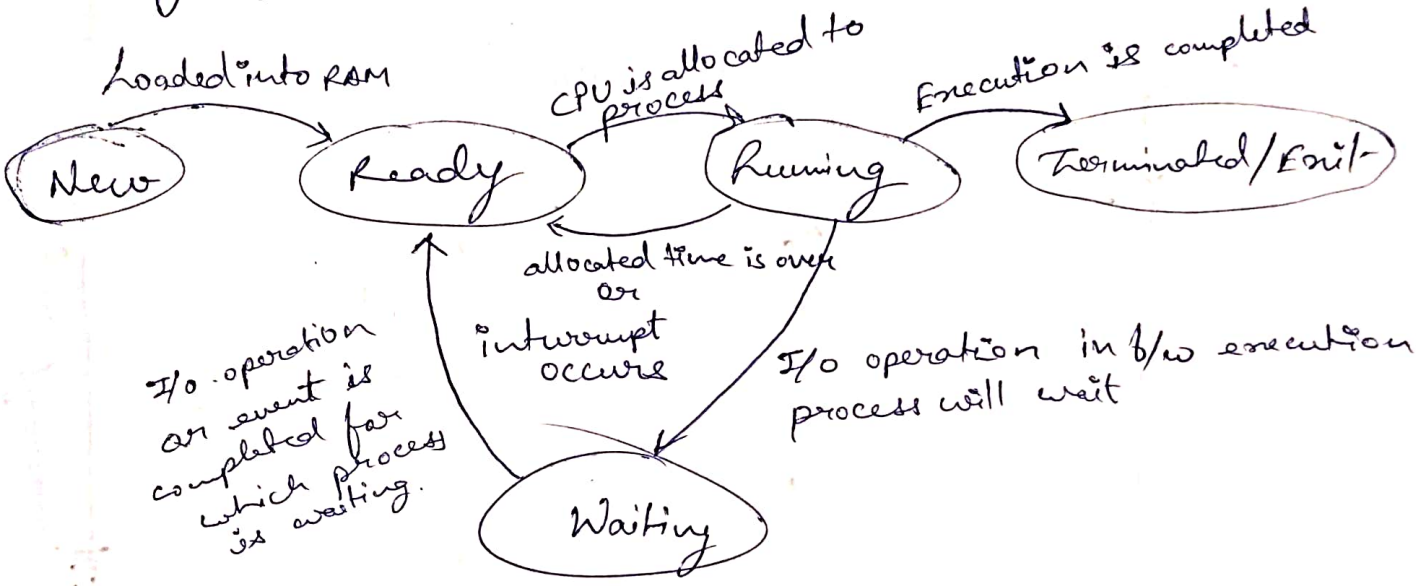
1) New → while loaded into RAM

2) Ready
3) Running
4) Waiting

5) Terminated/Exit

Process state diagram

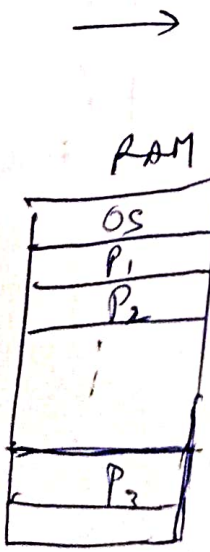
→ It indicates different situation in which process change 1 state to another.



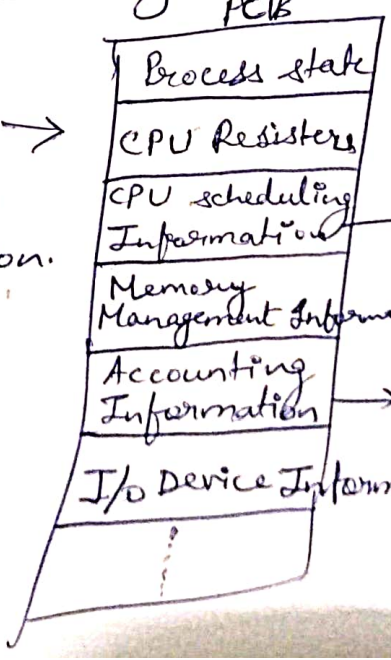
Process control block (PCB)

→ When OS created new process in computer system

→ It is used for storing information about the process

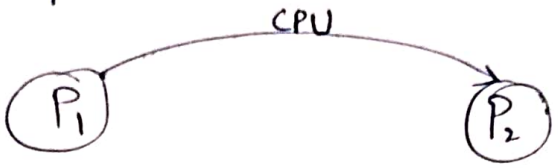


Inside PCB OS stores large amount of information.



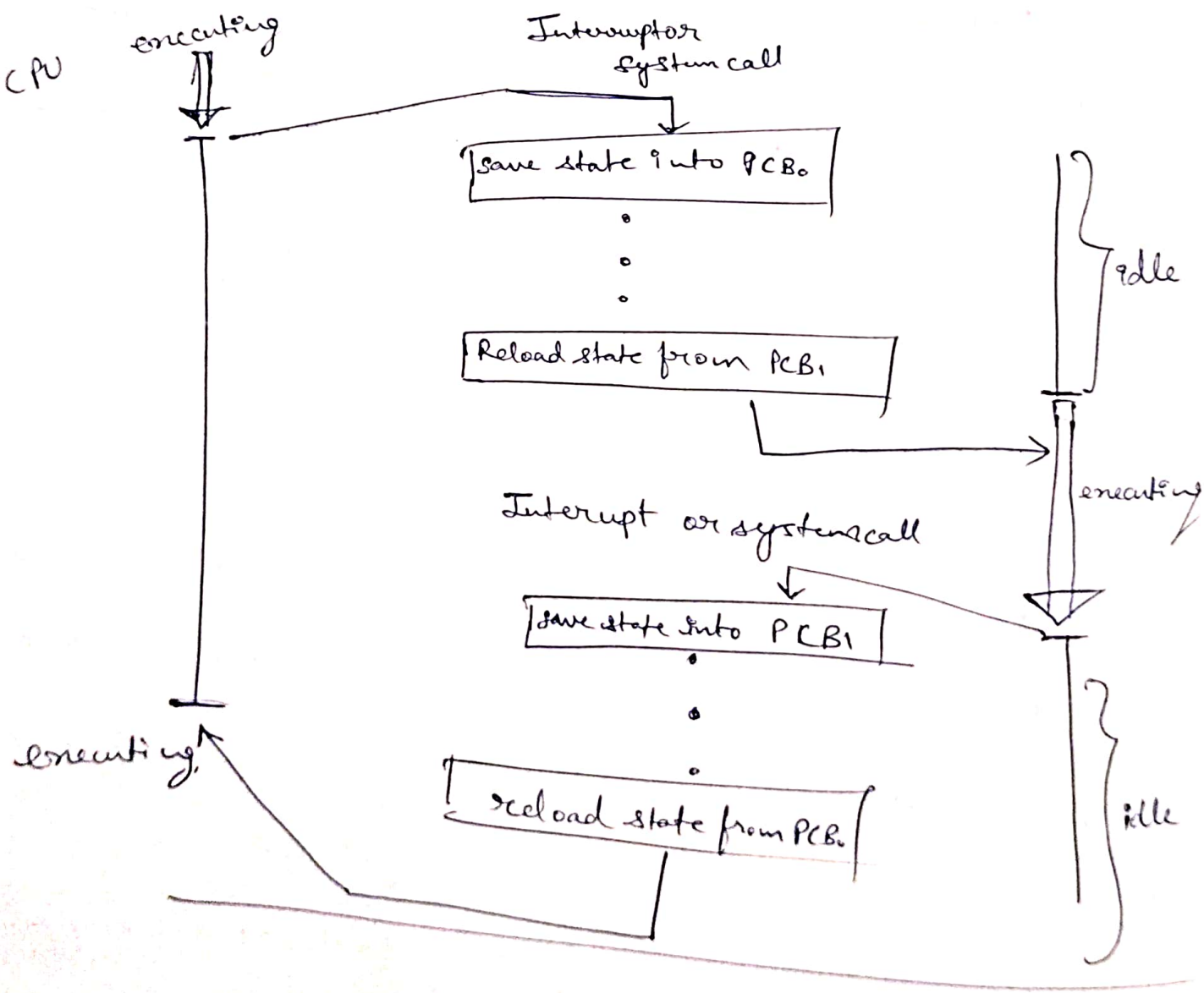
Content switch

→ switching the CPU from one process to another process.



before switching OS performs some act

CPU scheduling from P_1 to P_2



Process scheduling

RAM

P_1

P_2

P_3

P_4

P_5

5 process ready for process execution

1 CPU can execute 1 program at a time.

P_2, P_5, P_1, P_3, P_4

1) Arrival Time
2) Priority
3) Burst time

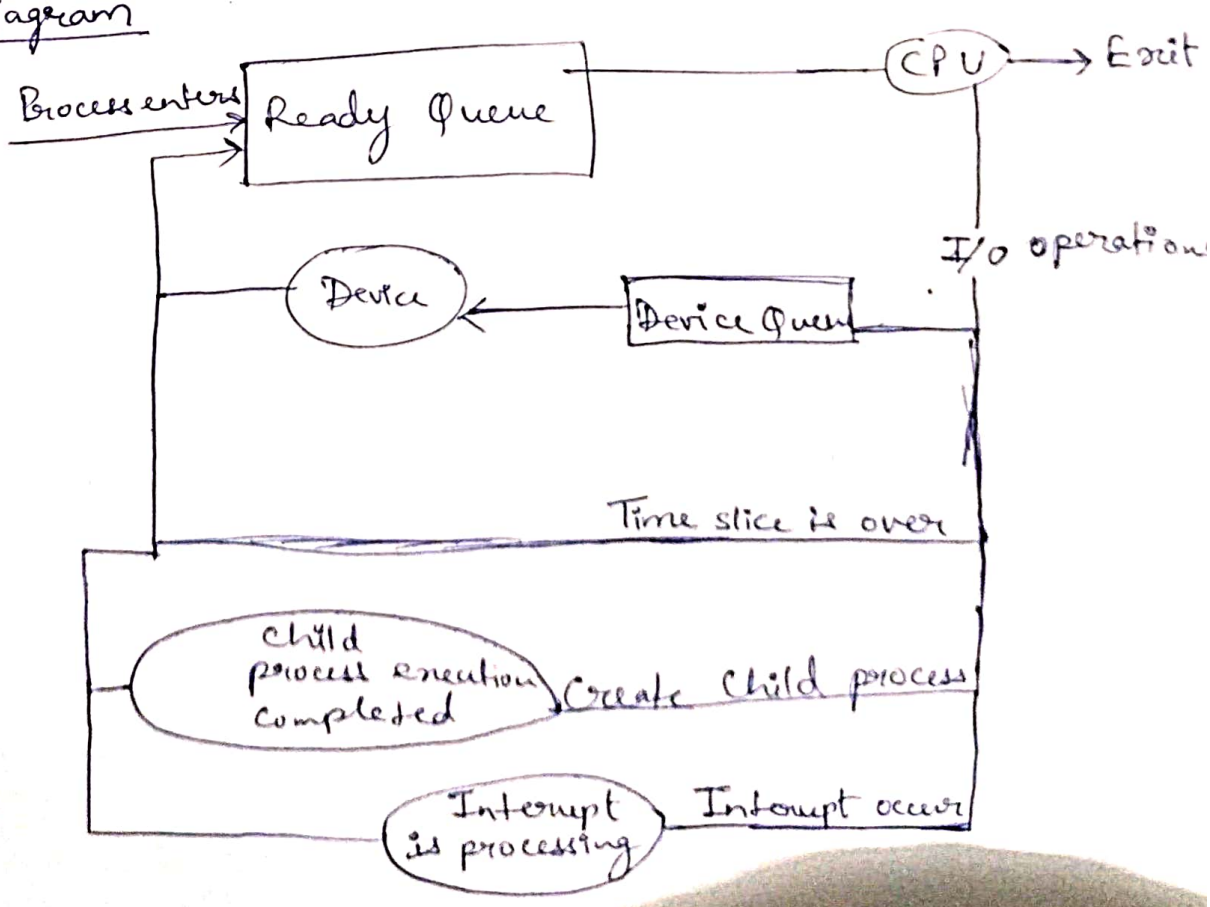
1) Scheduling Queues

1) Ready Queues - contains PCB's of the processes that are ready for execution.

2) Device Queue - An OS will maintain seperate device queue for each device.

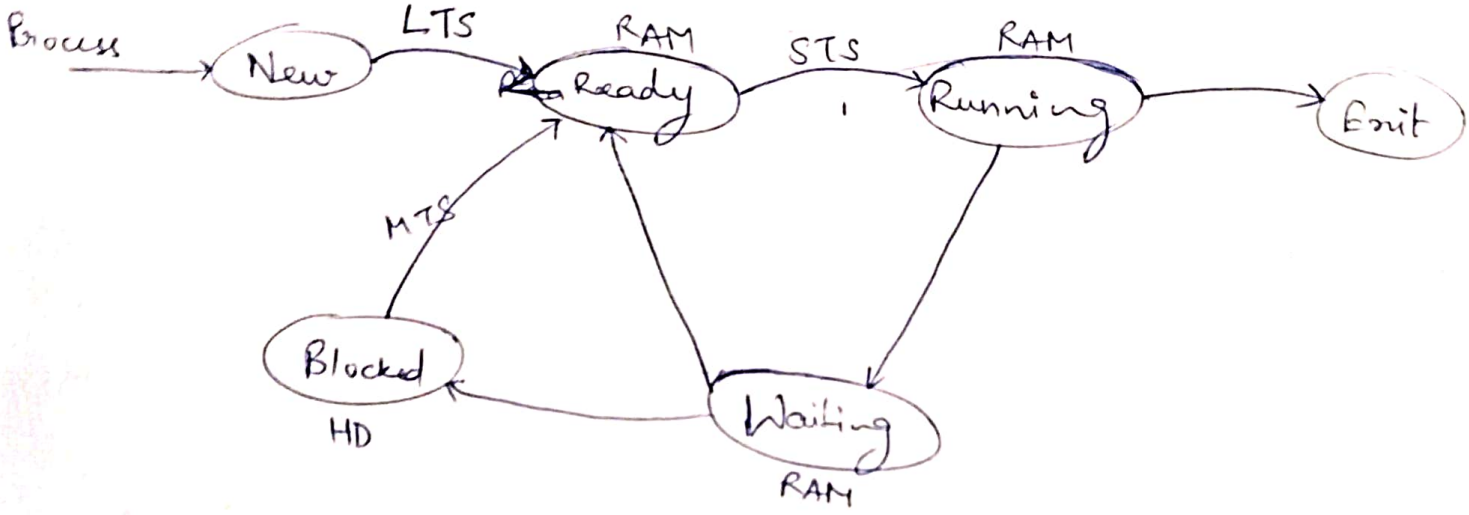
It contains PCBs of the processes that are waiting for that device

Queuing Diagram



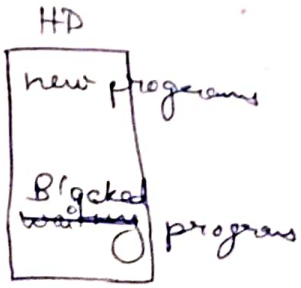
Types of schedulers

1) Long term scheduler (LTS)
2) Medium term (MTS)
3) Short term (STS)



When OS wants to load new program to RAM then LTS will selected by OS.

When OS wants to ~~waiting~~ load ~~RAM~~ new program to RAM as well as waiting state ~~will be~~ then MTS will be selected by OS.



STS will select one of ~~the~~ programs in ready queue for allocating the CPU.

as CPU scheduler.

CPU scheduler → dispatcher → dispatching latency

→ In order to select one of the program from ready queue CPU scheduler is impleted using CPU scheduling algorithms.

1) Non preemptive — CPU is switched to next process only when the current process releases the CPU.
2) preemptive

Scheduling criteria

To evaluate CPU scheduler it is used.

High
1) CPU utilization - % of time for which CPU is in busy state.
2) Throughput - indicates no. of programs completed for unit time.

Majorly used for comparision
3) Turnaround time - TAT = completion time - Arrival time
4) Waiting time - WT = TAT - Burst time

Burst time: total CPU time to execute a process.

Low
5) Response time - RT = Time at which the first output is responed — Arrival time

Different CPU scheduling algorithms

Basic many CPU algorithms
1) First come first serve (FCFS)
2) Shortest Job first (SJF)
3) Priority
4) Round Robin (RR)
5) Multilevel Queue
6) Multilevel feedback Queue.

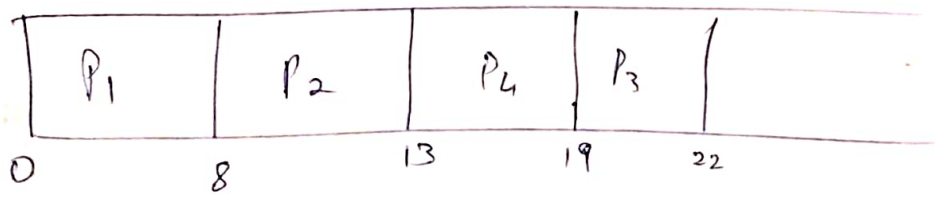
1) First come first serve (FCFS)

Process	Burst time	Arrival time
P_1	8	0
P_2	5	0
P_3	3	0
P_4	6	0

→ In case of FCFS the process is executed in order of entering.

FCFS → Non preemptive CPU scheduling algo.

Gantt chart



We will get 24 orders like this

TAT = 8 − 0 = 8
= 13 − 0 = 13
19 − 0 = 19
22 − 0 = 22

WT
8 − 8 = 0
13 − 5 = 8
19 − 3 = 16
22 − 6 = 16

AWT = (0 + 8 + 16 + 16) / 4

=

Process	Burst time	Arrival time
8	0	8 - 0 = 8
5	1	13 - 1 = 12
3	2	16 - 2 = 14
6	4	22 - 4 = 18

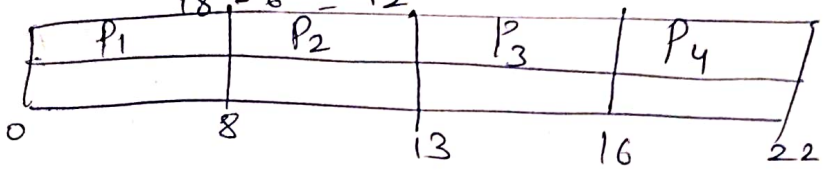
Turnaround time

6 - 8 = 0

12 - 5 = 7

14 - 3 = 11

18 - 6 = 12



Avg waiting time = (0 + 7 + 11 + 12)/4

= 7.5

Advantage

1) Simple and easy to implement

Disadvantage

1) Not suitable for time sharing operating systems
2) Average waiting time is high.

2) Shortest Job first (SJF) scheduling algorithm

SJF has two versions : preemptive

Rest of the CPU scheduling algorithm are from here

2) (SJF) Shortest job first

→ Two versions: 1) Non preemptive
2) preemptive

→ process are executing in burst time
process in least burst time is executed first

* process are executed in increasing order of burst time

→ If two or more burst time are same then FCFS algorithm will followed.

→ If different burst time then SJF algorithm will be followed.

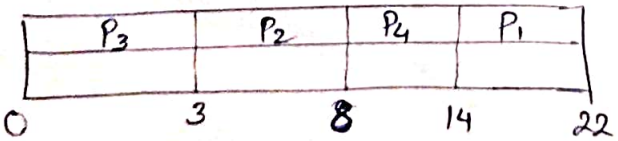
Non preemptive SJF

CPU switches to next process after completion of current process

→ Ex:

process	Burst time	Arrival time
P_1	8	0
P_2	5	0
P_3	3	0
P_4	6	0

Grantt chart



Turnaround time = completion time − Arrival time

22 − 0 = 22
8 − 0 = 8
3 − 0 = 3
14 − 0 = 14

Waiting time

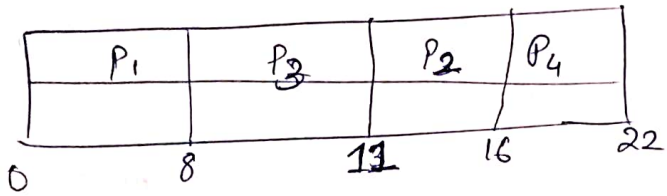
TAT − BT = 22 − 8 = 14
= 8 − 5 = 3
3 − 3 = 0
14 − 6 = 8

Avg waiting time = 14 + 3 + 0 + 8/4 = 6.25

Eg:

Process	Burst time	Arr time	TAT	WT
P_1	8	0		
P_2	5	1		
P_3	3	2		
P_4	6	4		

Gantt Chart



AWT =

Basis on arrival time

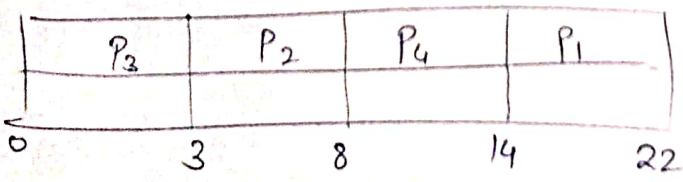
Preemptive SJF

→ If arrival time is 0 for all process then the Average waiting time will be same for preemptive & Non preempt

current process execute until the arrival of next process

process	Burst time	Arr time	TAT	WT
P_1	8			
P_2	5	0		
P_3	3	0		
P_4	6	0		
		0		

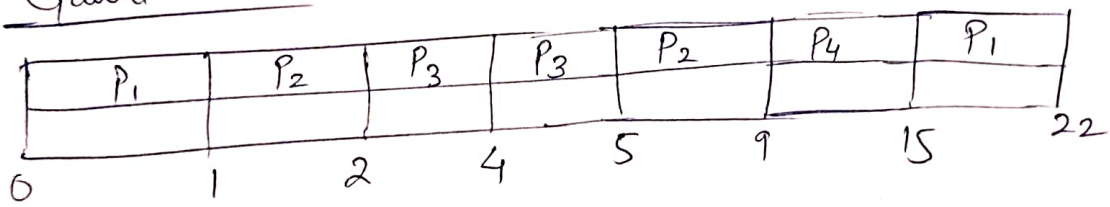
Gantt chart



AWT = 6.25

process	Burst time	Arrival time	TAT	WT
P_1	7	0	22 - 0 = 22	14
P_2	4	1 ←	9 - 1 = 8	3
P_3	10	2 ←	5 - 3 = 2	-1
P_4	0	4 ←	15 - 4 = 11	5

Gantt chart



AWT=

Some processes are repeating more than one time

Disadvantage: It works on Burst time, if Burst time is known then only we can use this algorithm

Algorithm

→ Non preemptive

⟹ preemptive

→ Decides the order of executing processes on the basis of priority scheduling

Priority - numeric value - 1, 2, 3, ...

1 → indicates highest priority.

Least value →

2

:

10 → lowest priority

Working of priority scheduling Algorithm on Non preemptive

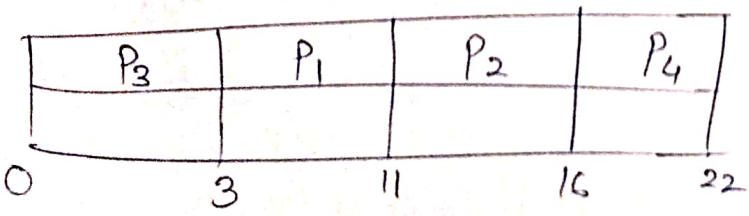
Eg:

Process	Burst time	Arrival time	Priority
× P_1	8	0	2
× P_2	5	0	3
× P_3	3	0	1
× P_4	6	0	4

P_3 is having highest priority

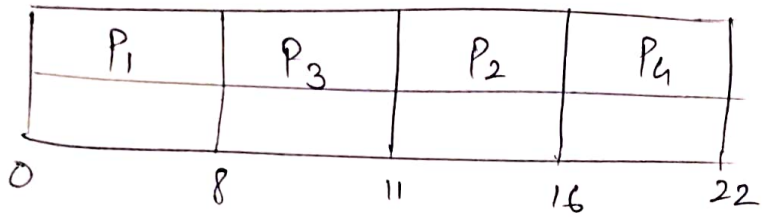
P_4 is having lowest priority

Gantt chart



Process	Burst Time	Arrival Time	Priority
× P_1	8	0	2
P_2	5	1	3
× P_3	3	2	1
P_4	6	4	4

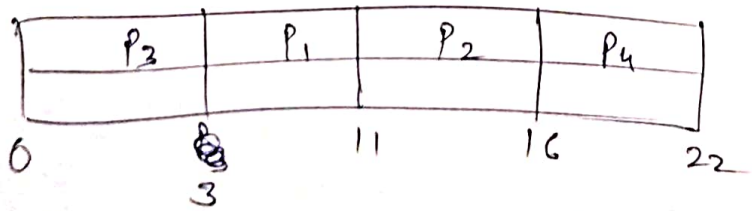
Gantt chart



preemptive priority scheduling Algo

Process	Burst Time	Arr Time	Priority
P_1	8	0	2
P_2	5	0	3
× P_3	3	0	1
P_4	6	0	4

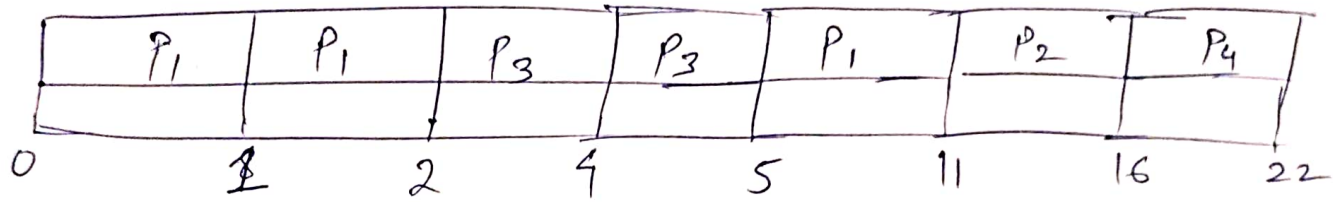
Gantt Chart



When arrival times are 0
Preem = Non preemptive

Process	Burst time	Arrival time	Priority
P_1	8 7 6 0	0	2
P_2	8 0	1	3
P_3	3 x 0	2 ←	1
P_4	6	4 ←	4

Gantt Chart



TAT

11 - 0 = 11

16 - 1 = 15

5 - 2 = 3

22 - 4 = 18

Inter process communication (IPC)

→ In a computer system, number of processes may be executing concurrently.

A process is called → **Independent** if it is not communicating with any other process.

Cooperating if it is communicating with any other process.

Processes must communicate for sharing resources:
Eg: if two or more processes requires data from same file (resource).

Two methods for speeding up inter-process communication:
1) Shared memory
2) Message passing.

Shared memory

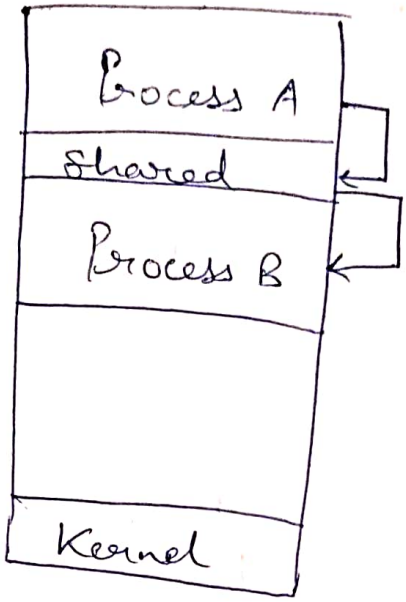
→ A region of memory shared by processes is created.

→ processes can exchange information by reading & writing data to the shared memory.

→ Any process which wants to communicate with the process creating the shared memory must attach the shared memory to its address space.

Generally, OS prevents from accessing the other process memory.

→ This restriction is omitted in the shared memory method.



Kernel → is a computer program at the core of a computer's OS and generally has to complete control over everything in the system.

↳ It is responsible for preventing and mitigating conflicts btween different processes.

Producer - consumer problem

In this problem, there are two processes: 1) Producer
2) Consumer

There is a buffer for storing items.

then → producer process stores items into buffer and consumer process takes items from buffer

Two pointers: (in & out) are maintained for the buffer.

In pointer → points to the next free slot in the buffer.

+

Out pointer → points to the slot from which item can be taken out from the buffer.

* Count → A variable that indicates number of items in the buffer.

→ Now; the Buffer, Sizeofbuffer and count are stored in shared memory as these are accessed by both processes.

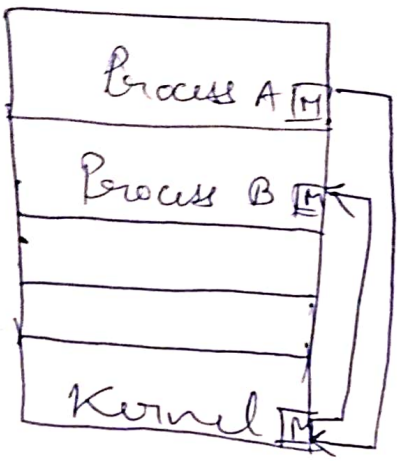
The producer process has to wait until a free slot is available in the buffer.

The consumer process has to wait until an item is available in the buffer.

* Using shared memory method, large amount of data can be exchanged between processes in less time

2) Message passing

with message passing, processes can communicate by exchanging short messages.



Two operations are used for exchanging the messages:

send (message), receive (message)

& communication link

The followings issues need to be considered:

1) Naming
2) Synchronization
3) Buffering

1) Naming → A direct communication link or an indirect communication link is used between the communicating processes.

→ Direct communication – sender process mentions the name of receiver process

similarly, receiver process has to mention the sender is process.

Syntax:

Send (P, message) – sends a message to process P

receive (Q, message) – receives a message from process Q.

Mod3

Process Synchronization (1st Part)

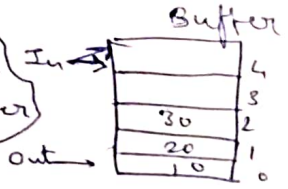
(2nd Part → Deadlock)

→ Process Synchronization

Example: Producer – consumer problem

- There is a Buffer with some number of slots
- One item can be stored in each buffer.
- In and Out are pointers to the buffer.
- Count is a variable which indicates no. of items in buffer
- Buffersize indicates no. of slots in buffer.
- Capacity of a buffer is only one item

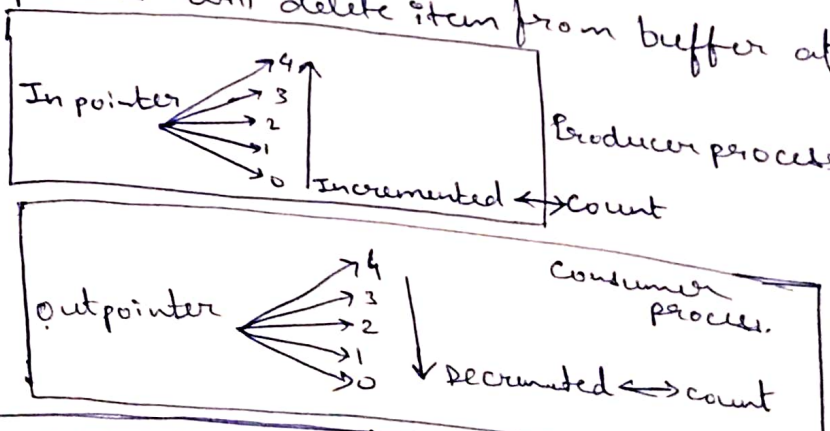
In pointer indicates the position where item is inserted in buffer
Out pointer indicates the position where item is deleted from buffer



Buffer size = 5
count = 3

→ Producer process will insert item into buffer at Inpointer position. ⇒ Inpointer will be incremented at next position.

→ Consumer process will delete item from buffer at outpointer position.



code for Producer process

```
While (true) {
    While (count == buffersize);
        Buffer[in] = item;
        in = (in+1) % buffer-size;
        count = count + 1;
    }
```

code for consumer process

```
While (true) {
    While (count == 0);
        item = buffer[out];
        out = (out+1) % buffer-size;
        count = count - 1;
    }
```

Machine language code (concurrent execution)

Producer process

```
P5: register1 = count;
P6: register1 = register1 + 1;
P7: count = register1;
```

```
P1: While (count == buffersize).
P2: ;
P3: Buffer[in] = item;
P4: in = (in+1) % buffersize;
```

consumer process

```
C1: While (count == 0).
C2: ;
C3: item = Buffer[out];
C4: out = (out+1) % buffersize;
C5: register2 = count;
C6: register2 = register2 - 1;
C7: count = register2;
```

~~P1 P2 P3 P4 P5 P6 C1 C2 C3 C4 C5 C6 P7 C7~~

Serial execution of process (Non preemptive)

→ Process are executed one after another.

→ CPU switch to next position only after completion of the currently running process.

→ N number of processes can be serially executed in N! number of ways.

Example: P1 and P2 are executing serially in two ways → 2! = 2

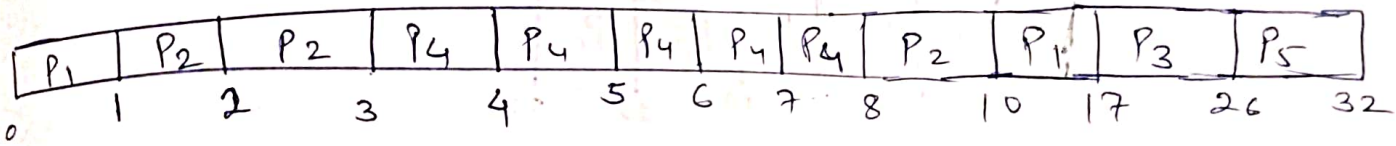
1) P1, P2 or 2) P2, P1

If P1, P2 and P3 are executing serially in three ways → 3! = 6

1) P1, P2, P3 2) P1, P3, P2 3) P2, P1, P3 4) P2, P3, P1 5) P3, P2, P1

6) P3, P1, P2

P_1	7	0	3
P_2	4	1	2
P_3	9	2	4
P_4	6	3	1
P_5		4	5

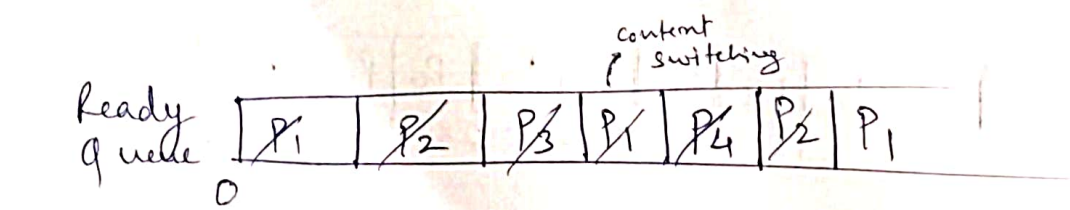


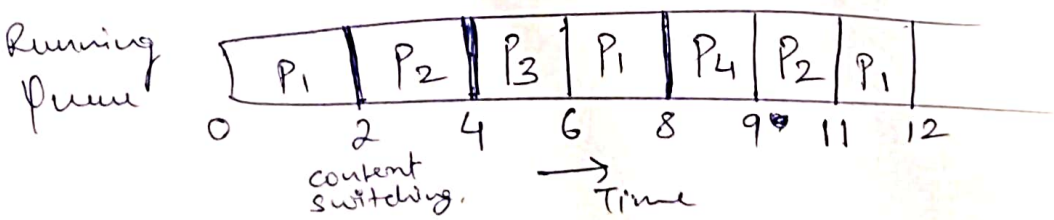
Round Robin.

RR → preemptive scheduling algorithm similar to FCFS.

Time quantum = 2

	BT	AT
P_1	5	0
P_2	4	1
P_3	2	2
P_4	1	4





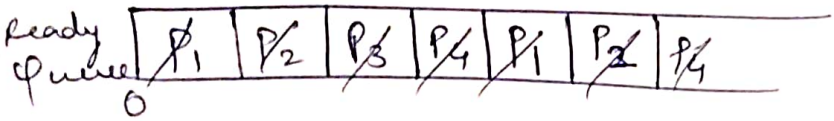
P1, P2, P3 ready

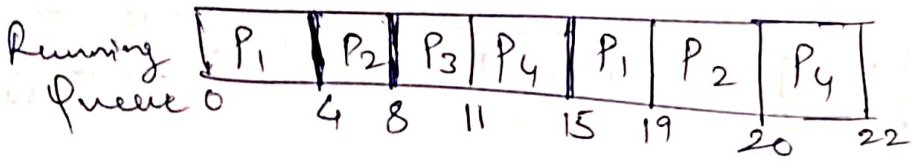
P4 ready

P3 completed.

P_1	8 4 0	0
P_2	5 1 0	1
P_3	3 0	2
P_4	6 2	4

QT = 4

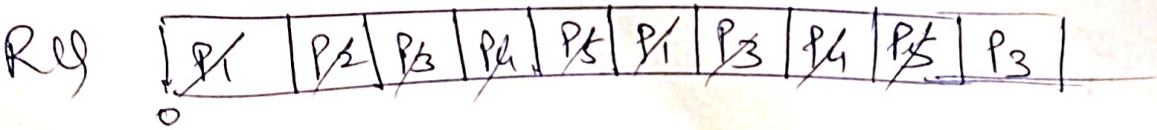


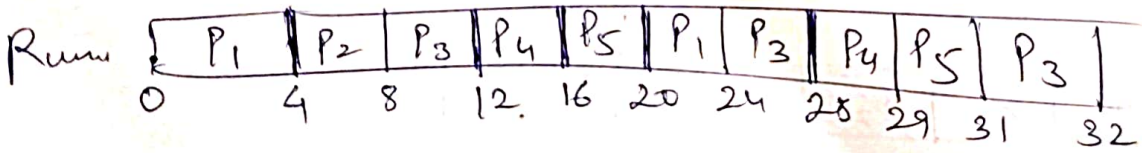


P_1, P_2, P_3, P_4 ready.

P_1	8 4 0	0	3
P_2	4 0	1	2
P_3	9 5 1	2	4
P_4	8 1	3	1
P_5	5 2	4	5

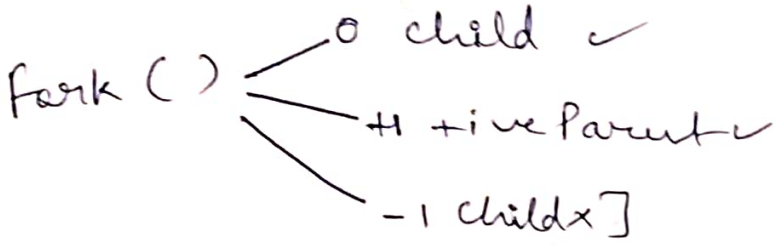
QT = 4





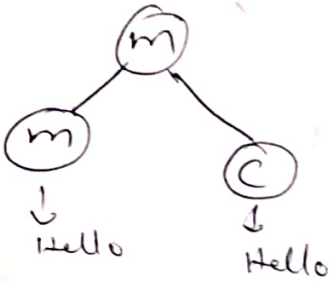
fork system calls

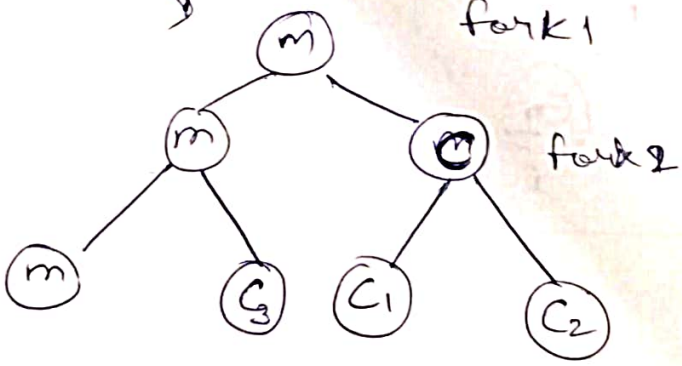
fork ( ) ⟶ parent process has child processes




```
main ( ) {
  fork ( );
  fork ( );
  Printf ("Hello");
  }
```

Output :
Hello
Hello

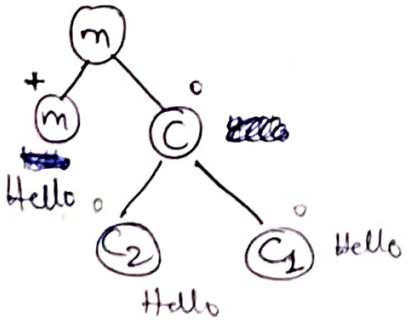




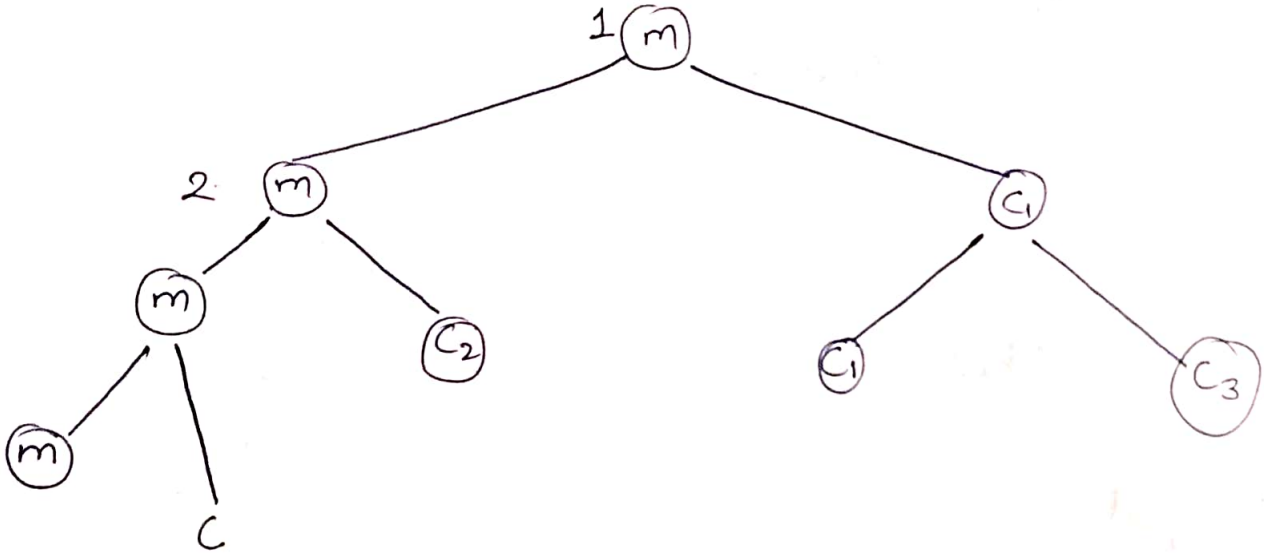
⇒ 4 bar Hello.
3 child 1 parent

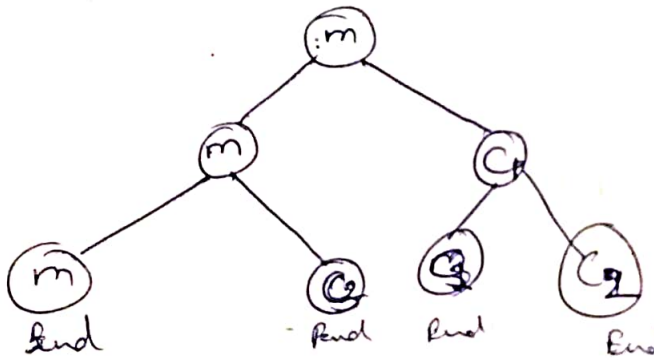
Total Execution = 2^n

Total child = $2^n - 1$



```
fork(); 1
((fork() 2 && fork() 3) || fork() 4);
fork(); 5
```





→ Memory management deals with the allocation and de-allocation of space in main memory for programs or processes.

→ Performance of computer system is high when the CPU is kept busy at all times.

→ A number of processes or programs, is loaded into main memory at a time in order to keep the CPU busy at all times.

→ During the execution of a process, if the process waits for an input/output then the CPU is switched to other process in the main memory.

→ A program or process is a collection of statements or instructions.

→ When the user wants to execute the program then the OS loads the program into main memory.

→ CPU don't know the location of program in the MM.

→ CPU generates the logical address of the statement.

→ The set of logical addresses generated by the CPU during the execution of the program is called as logical address space.

→ Physical address

→ Physical address refers to a location inside the main memory where a statement of the program is loaded.

the set of physical addresses of locations inside main memory where the statements of the program are loaded.

Memory management unit (MMU)

→ MMU maps the logical addresses to their corresponding physical addresses.

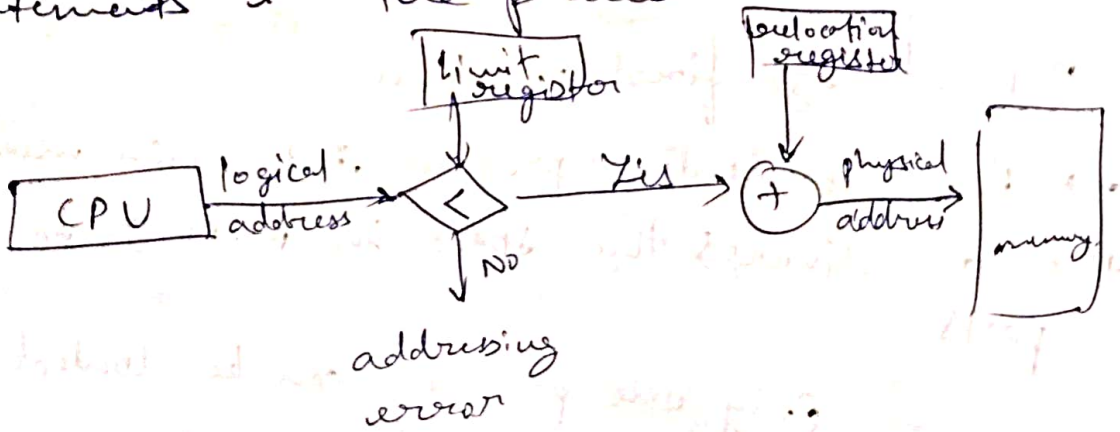
This mapping is also called as relocation.

MMU maintains two registers

1. Base or relocation register
2. Limit register

The relocation register is used to store starting physical address of the process.

The limit register is used to store the no. of statements in the process



Memory management Techniques

→ The OS uses the following techniques to ~~store~~ allocate space for programs in the main memory.

1) Contiguous Memory Allocation
2) Paging
3) Segmentation.

1) Contiguous Memory allocation

→ OS loads the entire program as a single unit into the main memory.

Different approaches used in contiguous memory allocation are:

1) Equal size fixed partition
2) Unequal size fixed partition
3) Dynamic partition

1) Equal size fixed partition

→ Before loading program into main memory, the OS divides the space in M.M into equal size parts.

Only one program can be loaded into each partition of main memory.

Advantage

→ To load a program, the OS can select any free partition of M.M

Disadvantage

1) Wastage of space inside the partition

2) When the size of program to be loaded is greater than the size of partition then it is not possible to load the program into M.M.

2) Unequal size partition technique

Before loading any program into M.M, the OS divides the space in main memory into different size parts.

→ OS has to select a suitable partition of main memory for loading a program.

Dynamic partition technique

OS divides the space in main memory into parts at the time of loading programs into main memory.

Initially, main memory is divided into two parts. OS is loaded into one part and other part is used for loading user programs.

After loading Program say P_1, the space into MM is divided into 3 parts.

After loading program say P_2, the space into MM is divided into 4 parts.

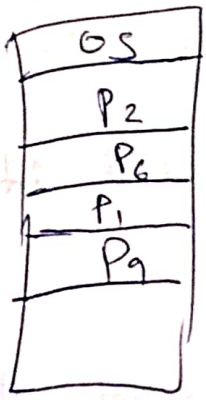
Disadvantage

→ Some space is wasted outside of the partitions. Wastage of space outside the partitions is called 'External Fragmentation'.

To avoid external fragmentation, Compaction technique can be used.

Compaction techniques moves all free spaces in the main memory together.

M.M



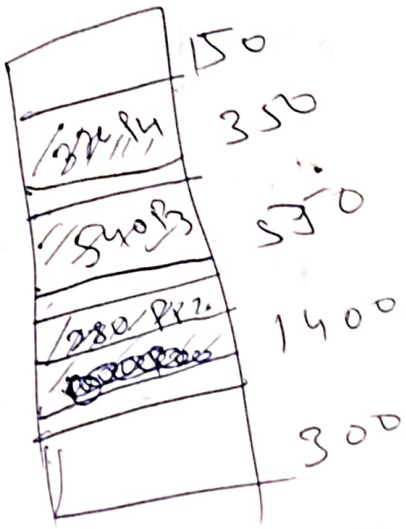
Placement Algorithm

1) first fit
2) Best fit
3) Worst fit

M.M

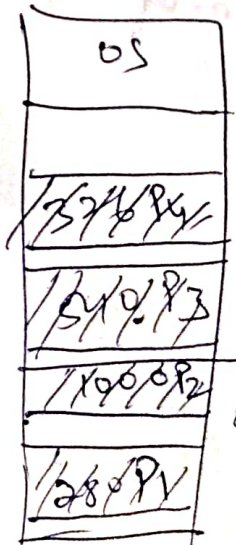
180 + (350 - 280) + (550 - 540) + (1400 - 1000) + 300
= 604 + 400 = 104

150 + (350 - 326) + (550 - 540) + (1400 - 280) + 300
= 1604



150 + (350 - 326) + (550 - 540) + (1400 - 1000) + (300 - 280)

280 KB
1000 KB
540 KB
326 KB.



Paging Technique

→ Before loading any program into main memory the space in main memory is divided into equal size frames.

To load program into main memory, the program is divided into equal size pages.

Size of pages is equal to size of frame

generally the page size / frame size is selected as a power of 2.

→ The pages of program are loaded into free frames of main memory.

→ After loading all pages of program, a page table is created for that program

→ No. of entries in page table of program is equal to no. of pages in the program.

The entries of page table are indexed with page numbers

Q. Logical address space 256 pages with 64 words per page mapped onto a physical memory of 4096 frames

i) How many bits are required in the logical address?

No. of pages = 256 = 2^8 pages

Size of each page = 64 words = 2^6 words

No. of frames in physical memory
= 4096 frame
= 2^{12} frames

Size of logical address
= Size of page no. field + Size of offset field
= 8 bits + 6 bits = 14 bits

Size of physical address = size of frame no. field
+ Size of offset field.
= 12 bits + 6 bits
= 18 bits

Q. No. of frames in main memory

$$= \frac{\text{Size of main memory}}{\text{Size of each frames}} = \frac{2^{27}B}{2^{13}B} = 2^{14} \text{ frames}$$

Size of page = size of frames

No. of bits allocated to offset = No. of bits in page size
= 13 bits

No. of entries in page table = no. of pages in the program

$$= \frac{\text{Size of program}}{\text{size of each page}} = \frac{2^{32}\,B}{2^{13}\,B} = 2^{19}$$ entries

Q.

Size of physical memory or main memory = 128 MB

$= 2^7 \times 2^{10} \times 2^{10}$ B

$= 2^{27}$ B

Size of (logical address) or virtual address = 32 bit

Size of page = 8 KB = $2^3 \times 2^{10}$ B

$= 2^{13}$ B

Size of program = 2^{32} B

i) no. of bits in physical address

= 27 bits

ii) no. of frames in main memory

$$= \frac{\text{Size of main memory}}{\text{Size of each frame}}$$

Size of page = size of frame

$$= \frac{2^{27}\,B}{2^{13}\,B} = 2^{14} \text{ frames}$$

iii) No. of bits allocated to offset

= No. of bits in page size

= 13 bits

iv) no. of entries in page table

$$= \frac{\text{no. of pages in program}}{}$$

$$= \frac{\text{Size of program}}{\text{Size of page}} = \frac{2^{32}\,B}{2^{13}\,B} = 2^{17} \text{ entries}$$

Q.

Size of virtual memory or logical address $= 64 \text{ bit} = 2^{64}\,B$

Size of physical address or main memory $= 60 \text{ bit} = 2^{60}\,B$

Size of page $= 8\,KB = 2^3 \times 2^{10}\,B = 2^{13}\,B$
$=$ Size of frames

Size of each page table entry $= 32\,B = 2^5\,B$

i) Main memory size $= 2^{60}\,B$

ii) no. of frames in M.M $= \frac{\text{Size of M.M}}{\text{Size of frame}}$

$$= \frac{2^{60}\,B}{2^{13}\,B} = 2^{47} \text{ frames}$$

iii) $\frac{\text{Size of page table}}{\text{Size of entries}} = \text{No. of entries in page table}$

$\Rightarrow$ Size of page table $= 2^5 \times \frac{\text{Size of page table}}{\text{Size of entries}}$

$$\frac{\text{Size of page table}}{\text{Size of entries}} = \text{No. of entries in page table}$$

$$\frac{\text{Size of page table}}{\text{Size of entries}} = \text{No. of pages in program}$$

$$= \frac{\text{Size of program (Logical address)}}{\text{size of pages}}$$

$$\text{Size of page table} = \frac{2^{64}\ B}{2^{13}\ B} \times \text{Size of entries}$$

$$= \frac{2^{64}\ B}{2^{13}\ B} \times 2^5\ B$$

$$= 2^{56}\ B$$

iv) no. of bits used for offset = 13 bits

Q.4 Size of LA = 48 bit = 2^{48} B

Size of page = 4KB = $2^2 \times 2^{10}$ B
= 2^{12} B

Size of MM = 256 MB = $2^8 \times 2^{10} \times 2^{10}$
= 2^{28} B

Size of each page table entry = 32 B
= 2^5

ii) No. of frames in M.M

$$= \frac{\text{Size of MM}}{\text{Size of frames}}$$

$$= \frac{2^{28}\,B}{2^{12}\,B} = 2^{16} \text{ frames}$$

iii) $\frac{\text{Size of page table}}{\text{Size of entries}}$ = No. of entries in page table

= No. of page in program

$$= \frac{\text{size of program}}{\text{Size of page}}$$

$$\Rightarrow \text{Size of page table} = \frac{2^{48}\,B}{2^{12}\,B} \times 2^5$$

$$= 2^{41}\,B$$

iv) no. of bits used for offset = 12 bits

Producer process

```
While (true) {
    While (count == buffersize);
        Buffer [in] = item;
        in = (in+1) % buffer-size;
        count = count + 1;
    }
```

Consumer Process

```
While (true) {
    While (count == 0);
        item = Buffer [out];
        out = (out + 1) % Buffer-size;
        Count = count - 1;
    }
```

```
While (true) {
P1  While (count == Buffersize) {
P2          ;
P3  Buffer [in] = item;
P4    in = (in+1) % buffersize;
P5    register1 = count;
P6    register1 = register1 + 1;
P7    count = register1;
}
```

```
While (true) {
P1: While (count == 0)
P2:         ;
P3:   item = Buffer[out];
P4:   out = (out+1) % Buffsi;
P5:   register2 = count;
P6:   register2 = reg2 + 1;
P7:   count = reg2;
}
```

concurrent Execution or parallel Execution of process

→ CPU can switched to next process during execution of the current process.

The no. of concurrent executions possible with n number of processes depends on the number of statements in each process.

Race condition

Race condition is getting different outputs for different concurrent executions of cooperating processes.

Process P_1 :	Process P_2
I_1 : A = A − B;	I_{11} : A = A + 1;
I_2 : B = B + A;	I_{12} : B = B + 1;

Critical section of ~~a process~~ producer process

```
register1 = count;
register1 = register1 + 1;
count = register1;
```

Critical section of consumer process

```
register2 = count;
register2 = register2 - 1;
count = register2;
```

Critical section problem

When two or more cooperating processes are executing concurrently, then at a time only one process should execute the statements in its critical section.

When the execution of cooperating processes is restricted in this way then the correct output is generated even if the cooperating processes are executed concurrently.

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

1) Mutual exclusion
2) Progress
3) Bounded waiting.

1) **Mutual exclusion**

→ No two processes should execute the statements in their critical section at the same time.

2) **Progress** At any time, atleast one process should be in running state or active.

No deadlock should occur.

Bounded waiting

→ All processes should be given equal chance to execute the statements in their critical section.

Peterson's solution

→ It is applicable to two processes only.

Two variables are used
- int turn
- boolean flag[2]

turn - indicates whose turn it is to execute the statements in its critical section.

turn1 - indicates that 1st process (producer process) can execute the statements in its critical section

turn2 - indicates that 2nd process (consumer process) can execute the statements in its critical section.

→ flag indicates whether a process is ready to execute the statements in its critical section.

flag[1] - true - indicates the 1st process is ready to execute the statements in its critical section

flag[2] - true

initially, flag[1] and flag[2] are set to false

Producer Process

```
While (true) {
    While (count == buffersize)
        ;
    Buffer[in] = item;
    in = (in+1) % Buffer-size;

Entry Section
    flag[1] = true
    turn = 2;
    While (flag[2] == true && turn==2)
        ;

Critical Section
    register1 = count;
    register1 = register1+1;
    count = register1;

Exit Section
    flag[1] = false;
}
```

Consumer process

```
While (true)
{
    While (count == 0)
        ;
    item = Buffer[out];
    out = (out+1) % buffersize;
```

```
flag[2] = true;
turn = 1;
While (flag[1] == true && turn == 1)
```

Entry section

```
[                                    ]
```

Critical

```
flag[2] = false;
}
```

Hardware solution

Variable name 'lock' is used

```
boolean lock;

While (true)
{
    Acquire lock;
    Critical section
    Release lock;
}
```

```
boolean Test and Set (boolean lock)
{
        boolean temp = lock;
        lock = true;
        return temp;
}
```

Initial value of lock is false

Producer process

```
While (true)
{
```

```
While(Test and Set(lock))
        ;
```
Busy section

```

```
critical section

```
lock = false;
```
Exit section.

Semaphore solution

Semaphore is an integer value variable

There are two types of semaphore

1) Binary semaphore
2) Counting semaphore

Binary semaphore is used to solve critical section problem

Binary semaphore is also called 'mutex' variables binary semaphore is used to implement mutual exclusion.

Initial value of binary semaphore is 1.

→ A queue is associated with each semaphore variable

Two atomic operations are defined on a semaphore variable:

1) wait( )
2) Signal( )

```
wait (S)
{
        S=S-1;
        if (S<0)
        {
                suspend the execution of the process
                which has invoked the wait( )
                operation
                Inset that process in the queue
                associated with the semaphore
                variable S;
        }
}
```

```
Signal (s)
{
    If (s <= 0)
    {
        remove a process from the queue
        associated with semaphore
        variable S;
        restart the execution of that
        removed process;
    }
}
```

```
wait(s);
Critical section
Signal(s);
```

Problems of Synchronization

1) producer - consumer problem
2) Reader's - Writers Problem

There is a database that can be shared by a number of concurrent processes.

Some processes named "readers" only read data from the database.

Other processes named "writers" write data to the database

At a time, any number of readers can read data from the database without any conflict.

But, when writers is writing data to the database then other writers and readers are not allowed to access the database.

```
Semaphore rw_mutex = 1;
Semaphore mutex = 1;
int read_count = 0;
```

rw_mutex and mutex are binary semaphore
read_count keeps track of how many readers are currently reading from the database.

Writer process code is

```
While (true)
{
    wait (rw_mutex);
    Write data into the database;
    Signal (rw_mutex);
}
```

Reader process code is

```
While (true)
{
    wait (mutex);
    read_count = read_count++;
}
```

code of a reader process is

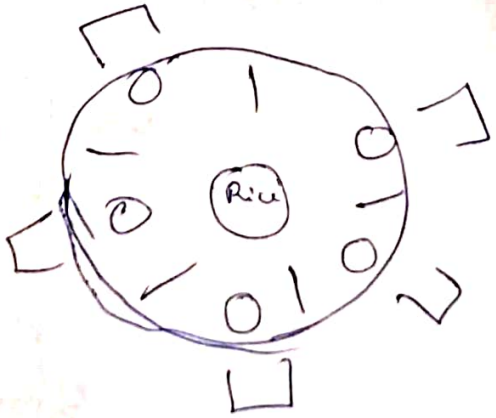
```
While (true)
{
        wait (mutex)
                read_count = readcount +1;
                if (read_count == 1)
                        wait (rw_mutex);
        Signal (mutex);

                Read data from database;

        wait (mutex)
                read_count = readcount -1;
                if (read_count == 0)
                        signal (rw_mutex);
        signal (mutex);
}
```

Dining philosophers problem

There are 5 philosophers.
A philosopher at any time is in either thinking or eating state.



The philosophers share a circular table surrounded by five chairs, each belonging to one philosopher.

In center of the table is a bowl of rice, and table is laid with five single chopsticks

When a philosopher is hungry then the philosopher tries to pick up the two chopsticks that are closest to him.

Semaphore chopstick[5];

where all element of chopsticks are initialized to 1.

The philosoper grabs chopstick by wait() operation on that semaphore.

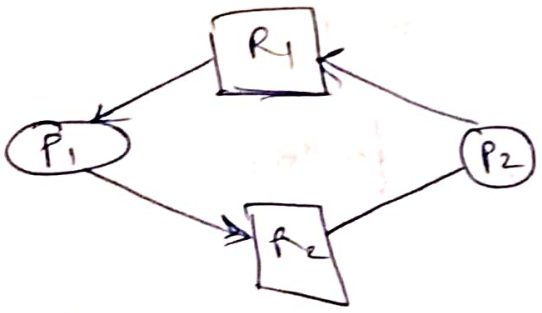
The philosopher releases his chopstick by enacting signal() operation on the appropriate semaphores.

Structure of philosopher i is

```
While (true)
{
    wait (chopstick[i]);
    wait (chopstick[(i+1)%5]);
        Eat for a while;
    Signal (chopstick[i]);
    Signal (chopstic[(i+1)%5]);
        Think for a while;
```

Deadlock

A set of processes is said to be in deadlock state if each process in the set is waiting for an event that can be caused by another process in the set.



Methods for handling deadlocks.

1) Deadlock prevention
2) Deadlock avoidance
3) Deadlock detection
4) Deadlock recovery

Necessary conditions for deadlock
Mutual Exclusion
Hold and wait
No preemption
Circular wait

Mutual exclusion condition indicates that a non-sharable resource should be used by only one process at a time.

Hold and wait
Each process is holding some resources and waiting for other resources.

No preemption
A resource cannot be released from a process before completion of the process.

circular wait

There is a set of processes $\{P_1, P_2, P_3, \ldots P_n\}$

P_1 is waiting for P_2

P_2 is waiting for P_3

P_n is waiting for P_1

Banker's algorithm deadlock avoidance.

Available: vector of size m → no of resources

Max: is matrix of size n x m

↓

no. of ~~resource~~ processes

Allocation: n x m

Need = Max - Allocation

Banker's algorithm has two parts

1) Resource request algorithm
2) safety algorithm
   ↳ whether the system is in safe state or not

if(Needi <= Available)

Bankers Algo

Total $R_1 = 6$, $R_2 = 7$, $R_3 = 12$, $R_4 = 12$

Process	Allocation				Max need				② Available				① Remaining need			
	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4	R_1	R_2	R_3	R_4
P_1	0	0	1	2	0	0	1	2	2	1	0	0	0	0	0	0
P_2	2	0	0	0	2	7	5	0	2	1	1	2	0	7	5	0
P_3	0	0	3	4	6	6	5	6	4	4	6	6	6	6	2	2
P_4	2	3	5	4	4	3	5	6	4	7	9	8	2	0	0	2
P_5	0	3	3	2	0	6	5	2	6	7	9	8	0	3	2	0
	4	6	12	12					6	7	12	12				

Safe

Remaining need =

~~Allocat~~ Max need − ~~Allocation~~

P_1 is safe as its remaining need can be fulfilled

P_1 will be removed.

P_4 is safe as its remaining need can be fullfilled

P_4 will be removed

P_5 will be safe as it remaining need can be fulfilled

P_5 is removed

P_2 will be safe as its remaining need can be fulfilled

P_2 removed

P_3 ——

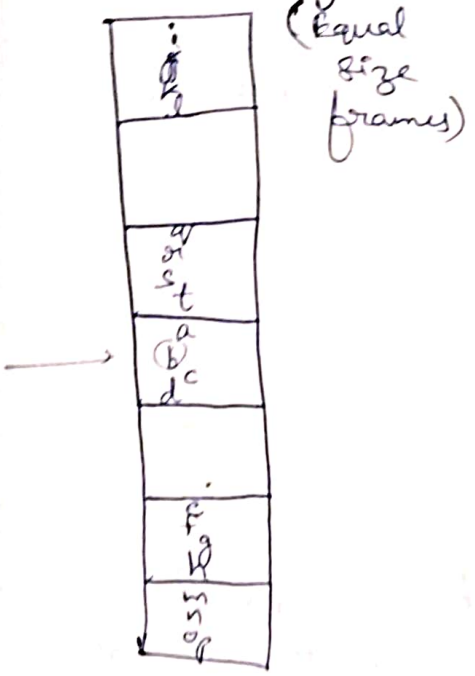
P_3 removed

$P_1 \downarrow P_4 \downarrow P_5 \downarrow P_2 \downarrow P_3$

Translation Lookaside Buffer (TLB)

1) What is TLB
2) How TLB is used in paging technique
3) Why TLB is ''

Main memory

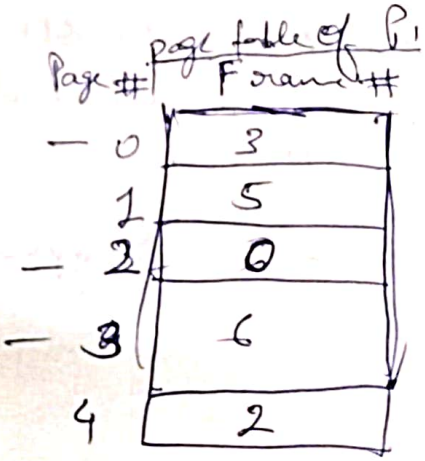


(Equal size frames)

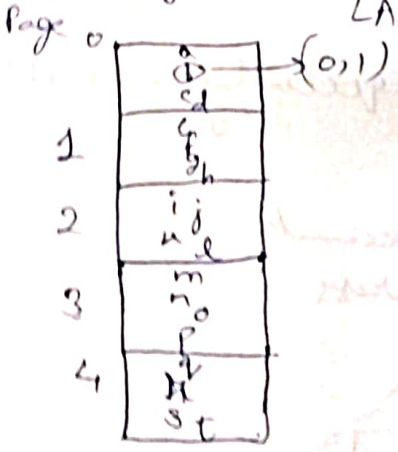
Frame 0
1
2
3
4
5
6

Size of frame = 4 statements
$= (2^2)$

Register



Program P_1



Time required to access data from register = 10 ns
'' '' '' '' '' '' M.M = 100 ns

$$\begin{array}{r} 10 \text{ ns} \\ +\ 100 \text{ ns} \\ \hline 110 \end{array}$$

– Time req to get one statement from register

Time req to access from M.M = 100 nsec
= 200 ns

TLB is used in paging technique to reduce the time of accessing statement from M.M.

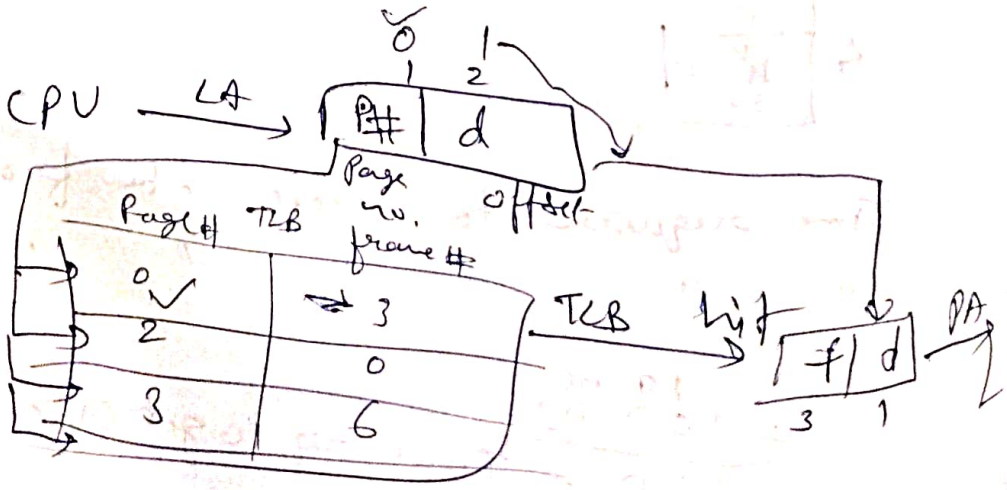
⇒ TLB is a hardware component just like CPU,
↓
contains no. of entries.
⇒ each entry in the TLB has two parts

#Page	TLB # frame
0	3
2	0
3	6

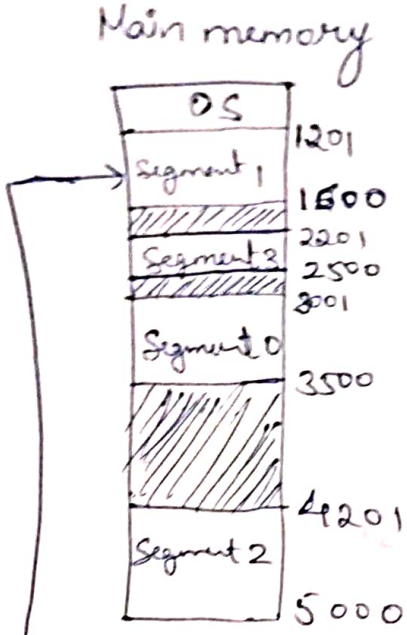
⇒ TLB is used to store information about frequently executed ~~statement~~ pages in the program.

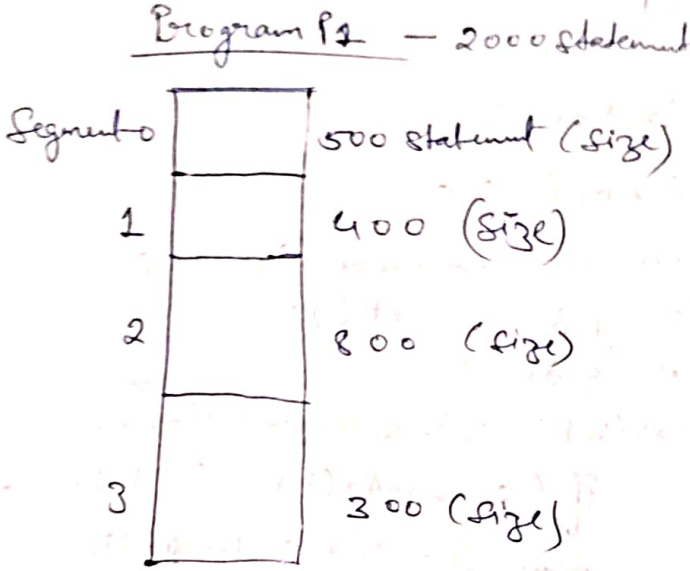
Suppose #Page 0, 2, 3 are frequently executed.

Once CPU executed a statement it generates logical address.



Segmentation technique





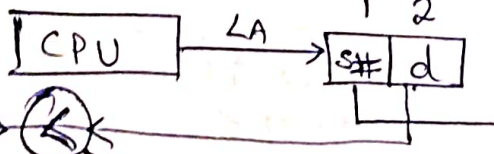
OS loads these statements in the user space of M.M.

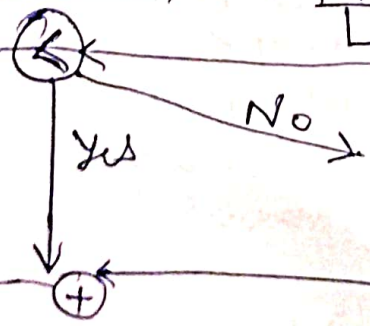
Segment table of P_1

S#	Limit/Length	Base
0	500	3001
1	400	1201
2	800	4201
3	300	2201

0 to 399

3rd statement.
En:450 from statment no. 1





2 + 1201 = 1203

Q. Consider the following Statement Table

S#	Base	Length
0	620	430
1	1800	60
2	170	74
3	2145	321
4	1450	115

Find the physical addresses for the following logical addresses

1) 0, 512 ~~512 < 620 ⇒ 512~~
2) 1, 47
3) 2, 13
4) 3, 185

1) LA → 0, 512 ⇒ Trap to OS (512 > 400)

2) LA → 1, 47 ⇒ (47 < 60) → Yes 1800 + 47 = 1847 PA

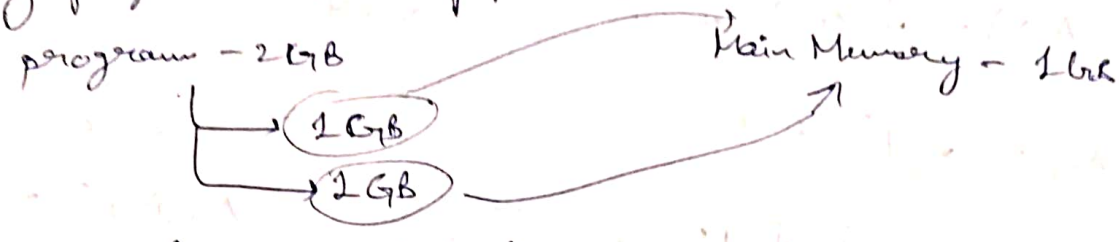
3) LA → 2, 13 ⇒ (13 < 74) → Yes 170 + 13 = 183. PA

4) (3, 185) ⇒ (185 < 321) → Yes 2145 + 185 = 2330 PA
LA

Module-5

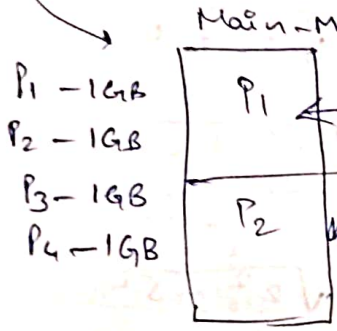
Virtual memory

Dividing programs into no. of parts



→ Advantages of using virtual memory

1) Allows to execute programs whose size is greater than the size of Main memory.
2) Improves the performance of computer system.

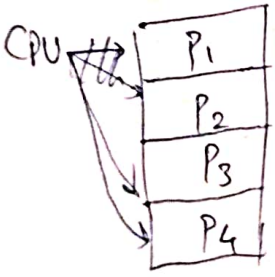


CPU P_1 waiting.

Switches.

executing P_2 req. I/o operation and P_2 is in waiting state

CPU is in Idle state (performance of CS is decreased)



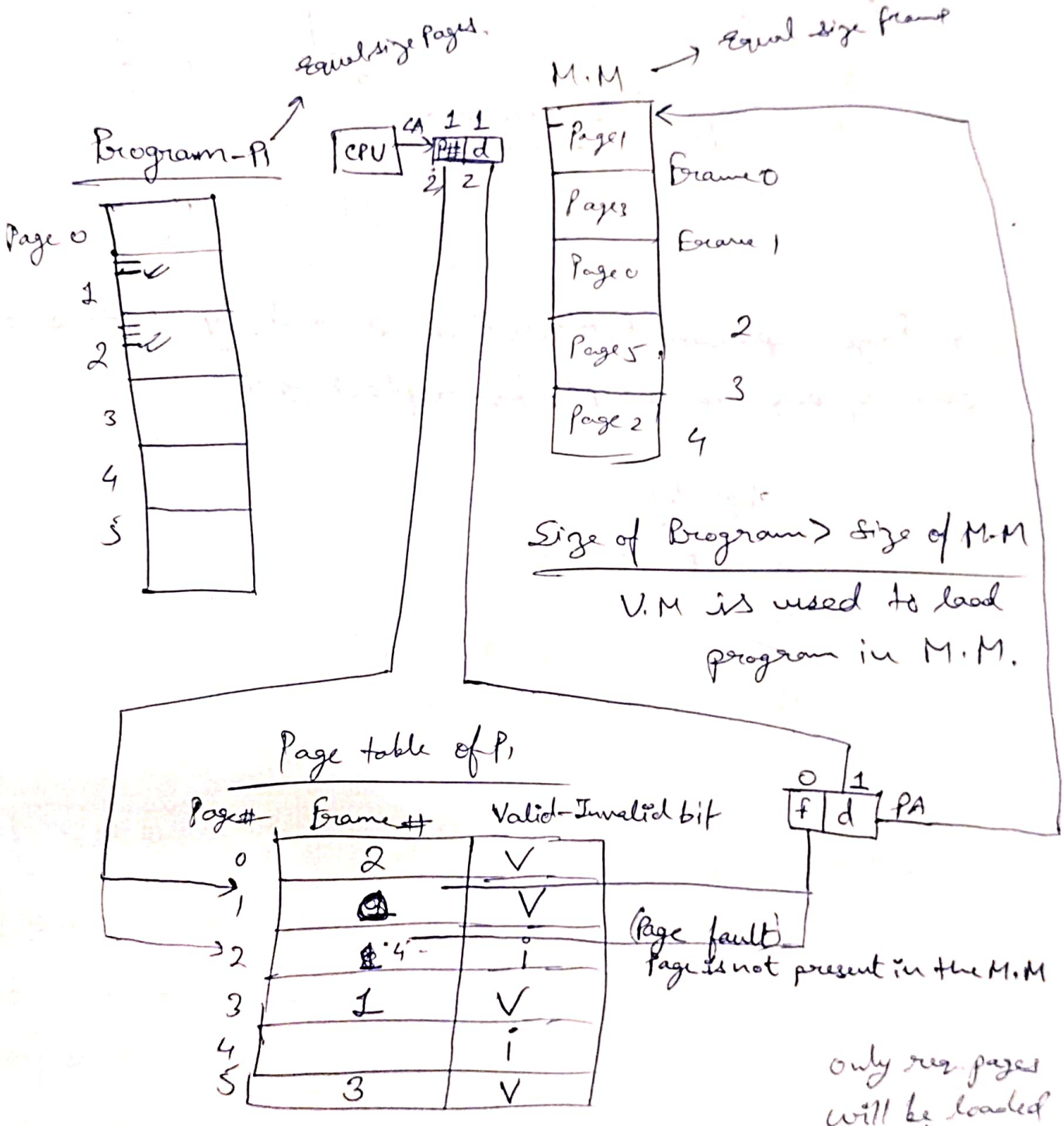
Now, operating system executes P_3 and P_4 after P_1, P_2 is in waiting state.

Virtual memory helps to load more no of programs in main memory to improve the performance of computer system.

3) Virtual memory supports sharing of data/files between the programs

Demand Paging Technique

→ When OS used Virtual memory then Demand paging technique is used to load programs in M.M.



What OS will do when the page fault occurs.

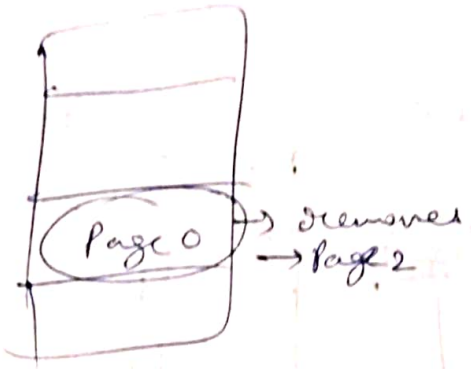
→ A trap is occurred in the OS

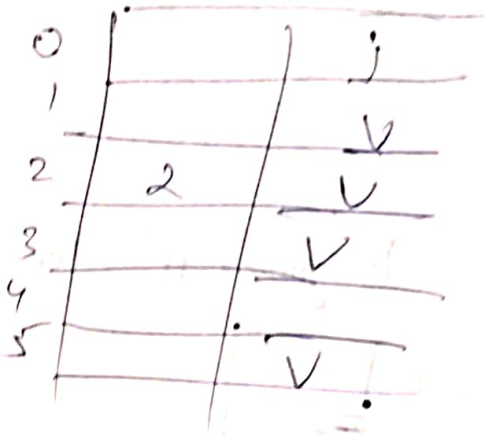
Activities / steps performed by OS to tackle page fault

1) First OS checks any free frame in the M.M.
OS will load the required page into M.M

If there is no free frame in M.M.
OS will replace one of page from main memory.

Ex





→ Page replacement Algorithm is used by OS to select one of page in M.M for replacing it

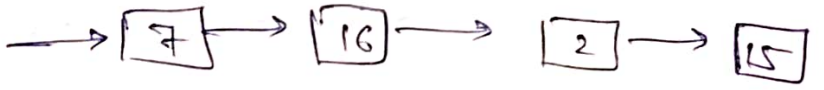
Steps by OS

1) Free frame.
2)
3)
4)

1) Virtual Memory

2) Demand Paging Technique

1) Free frame List



OS updates free frame memory based on previous.

2) performance of Demand paging Technique

Effective Memory Access Time = $(1-P) \times$ Memory Access Time + $P \times$ page fault time.

P → probability of occuring page fault — 0.2

100 pages
- → 80 pages are loaded in M.M
- → 20 pages are not in MM.

Memory Access Time = Time required to access a statement from the M.M

page fault time — The time required to load required page into MM.

P = 0

EMAT = $(1-0) \times$ MAT + $0 \times$ PFT

EMAT = MAT

If P = 0.2

EMAT = $(1-0.2) \times$ MAT + $0.2 \times$ PFT = 0.8 MAT + 0.2 PFT

MAT = 100 ns, PFT = 2000 ns

EMAT = 80 + 400 = 480 ns

p = 0.5, MAT = 100 ns, PFT = 2000 ns

EMAT = 0.5 X 100 + 0.5 X 2000

= 50 + 1000 = 1050 ns.

Page Replacement Algorithms

1) FIFO - First in first out
2) Optimal
3) Least Recently Used (LRU)

Input

1) No. of pages in the program
2) No. of frames in M.M
3) Order of execution of pages in the program Reference string.

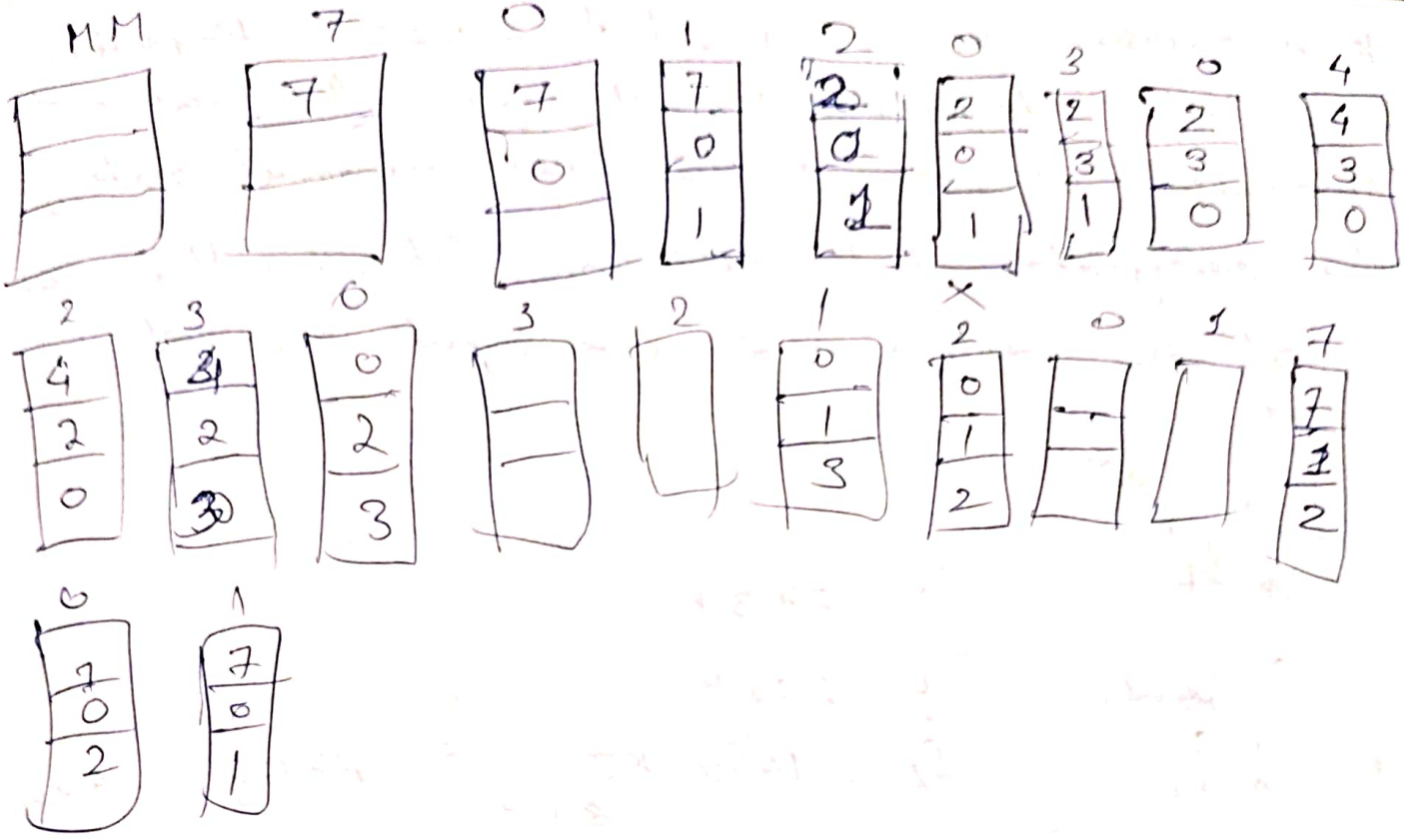
FIFO Page replacement Algo

→ Replaces the page that was loaded for the first time into M.M

Ex1: No. of pages in the program = 8 (0 to 7)

No. of frams in the MM = (3)

Reference String: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



No. of page fault = 15

No of frame = 4

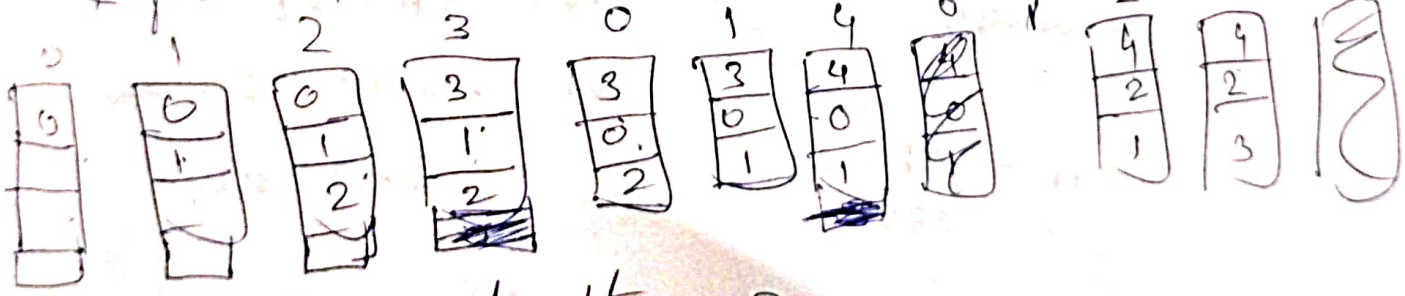
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1

No of page fault = 10

Ex: No. of pages in program = 5 (0 to 4)

No. of frams in the MM = 3

Reference string : 0, 1, 2, 3, 0, 1, 4, 0, 1, 2, 3, 4



No. of page fault = 9

No. of frame = 4

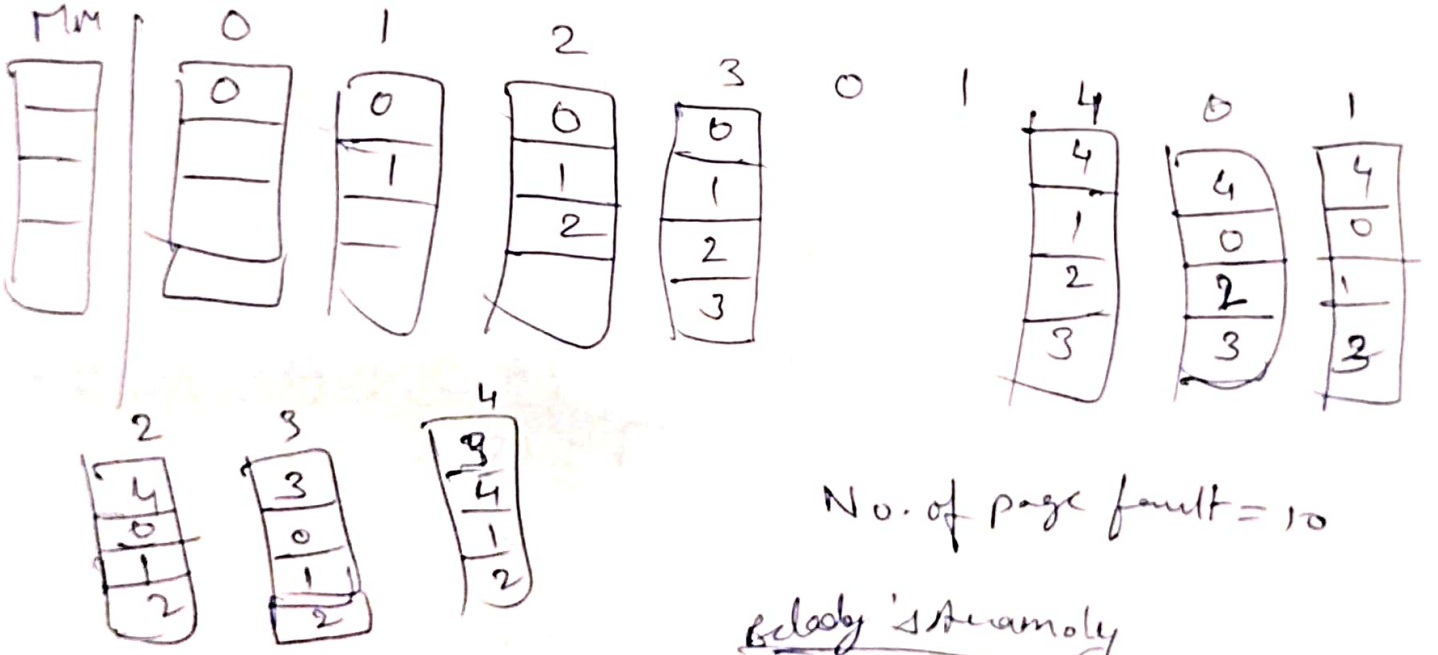
0, 1, 2, 3, 0, 1, 4, 0, 1, 2, 3, 4

Page Replacement Algorithms

→ used by OS to select page in Main memory

1) First in first out (FIFO)
2) Optimal
3) Least recently used (LRU)

1) No. of pages in the program
2) No. of frames in the M.M
3) Order of execution of pages of the program.
   Reference String.



No. of page fault = 10

Belady's Anamoly

Replaces the page that will not be used for the longest period of time.

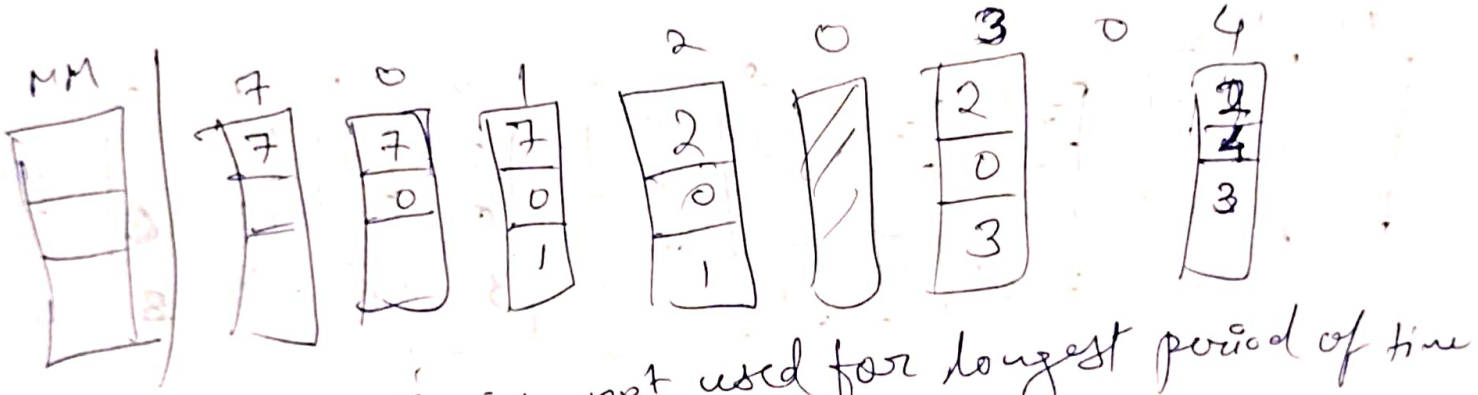
Optimal Algorithm

Ex:1 No. of pages in the program = 8 (0 to 7)

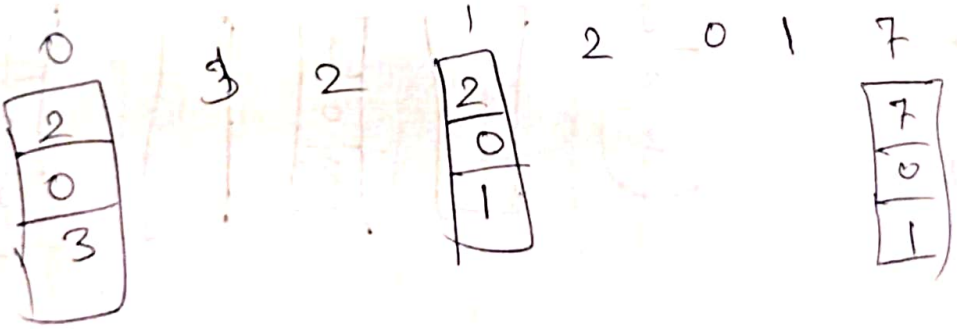
No. of frames in the M.M = 3

Reference String = 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2,

0, 1, 7, 0, 1



7 is not used for longest period of time

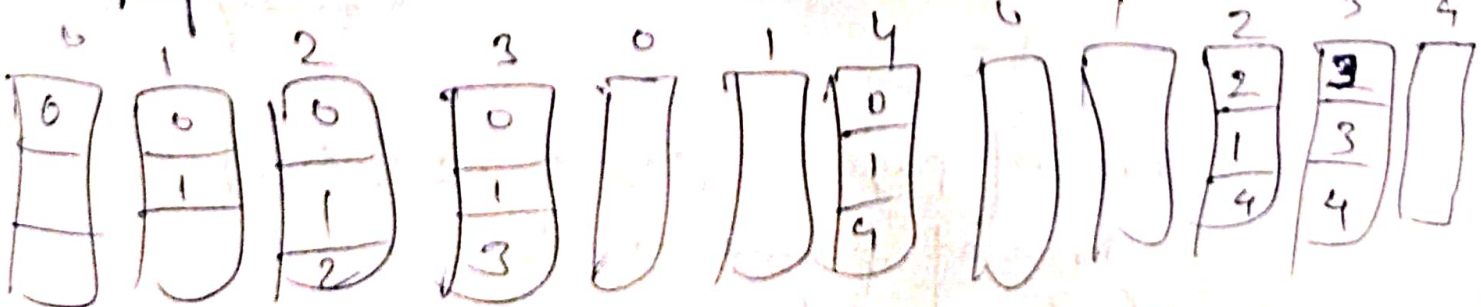


No. of page fault = 9

No. of pages in the program = 5 (0 to 4)

No. of frames in the M.M = 3

Reference String = 0, 1, 2, 3, 0, 1, 4, 0, 1, 2, 3, 4



No. of page fault = 7

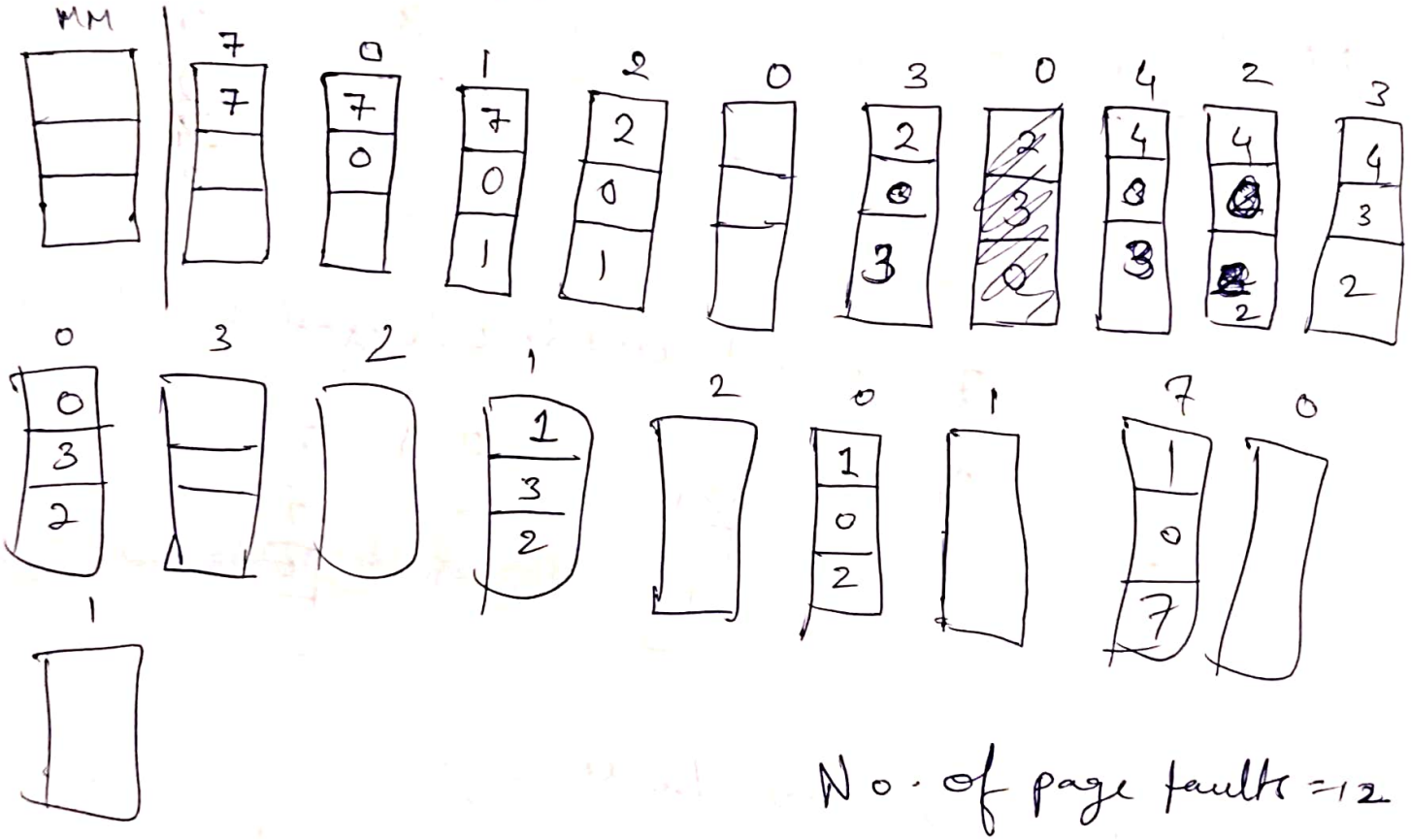
Least Recently used (LRU) page replacement Algorithm

→ Replaces the page that has not been used for the longest period of time.

Ex1 : No. of pages in the program = 8 (0 to 7)

No. of frames in the Main memory = 3

Reference string : 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



No. of page faults = 12

FIFO = 15

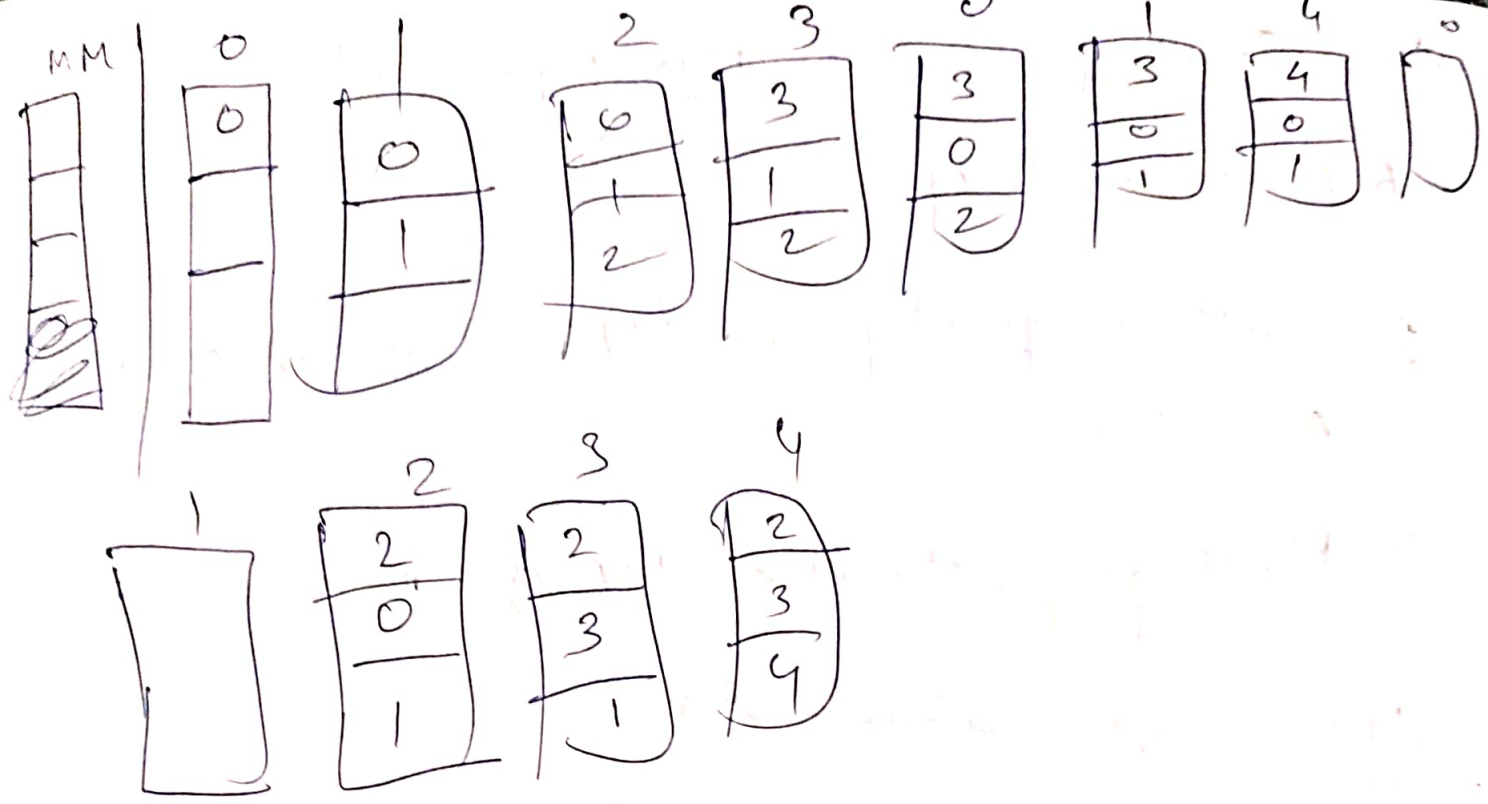
optimal = 9

LRU = 12 (advantage is < FIFO)

No. of page in program = 5 (0 to 4)

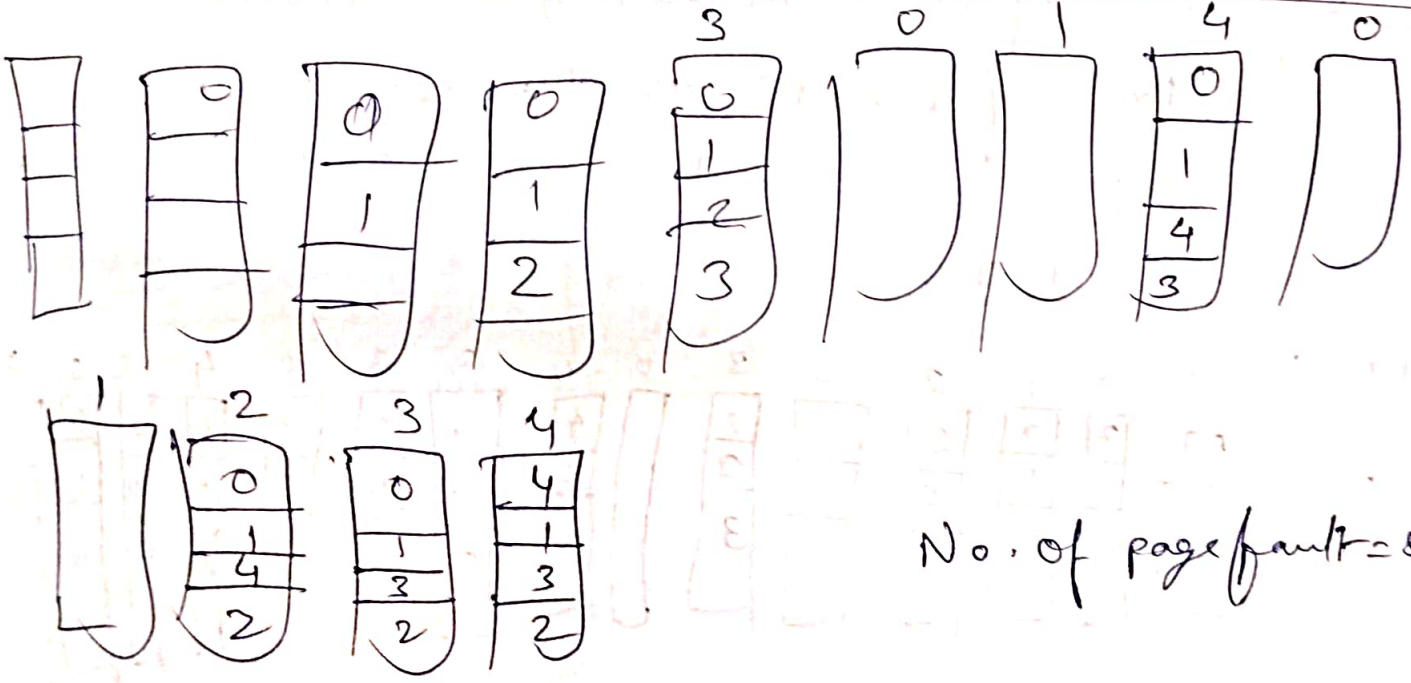
No. of frames in MM = 3

Reference string: 0, 1, 2, 3, 0, 1, 4, 0, 1, 2, 3, 4



No of page fault = 10

~~FIFO = 10~~



No. of page fault = 8

Adv

1) Less no. of page dfault compared to FIFO

2) No. Chance of occuring Belady's Anamoly

Counting - Based page Replacement

Reference count of pages

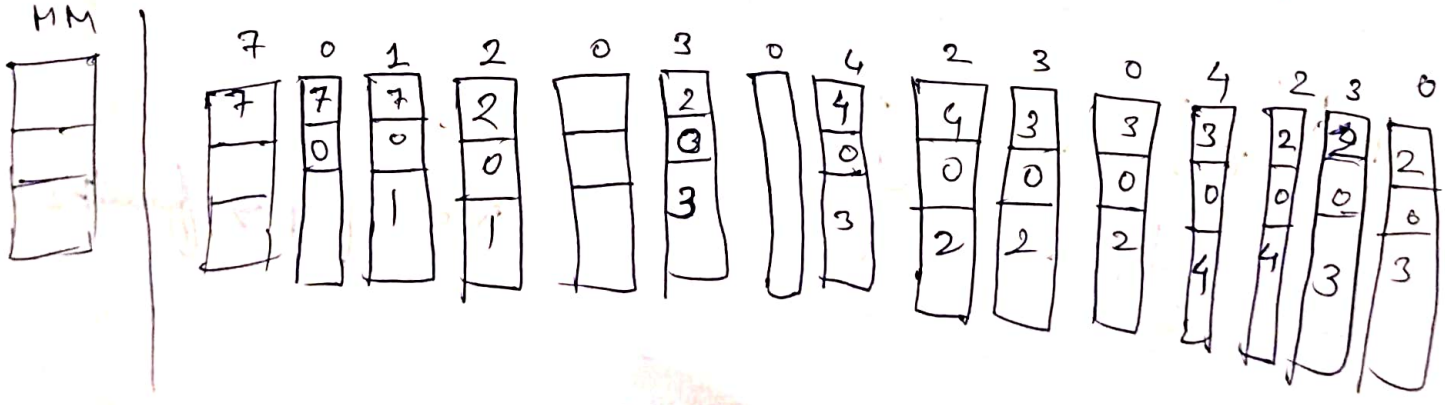
1) Least frequently used (LFU) page replacement Algorithm.

2) Most frequently used (MFU) "

LFU page replacement Algo

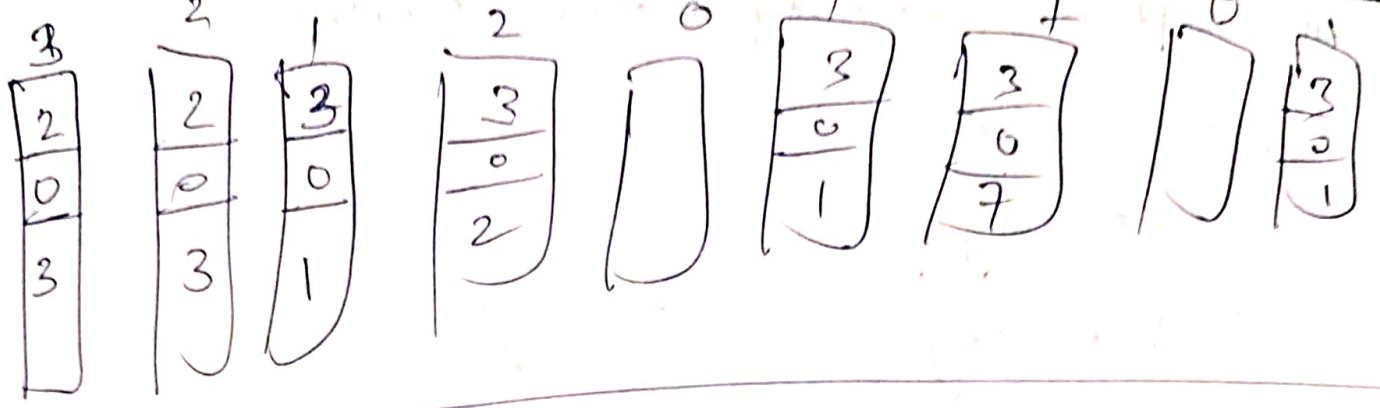
Replaces the page that has least reference count
FIFO

Ex:1 No. of pages in the program = 8 (0 to 7)
No. of frames in the Main memory = 3
Reference string : 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



page#	Reference count#
0	1 2 3 4 5 6
1	1 1 1 1 1
2	1 1 1 1 2 1 1
3	1 1 1 1 2
4	1 1
7	1 1 1

FIFO queue: 7, 0, 1, 2, 3, 4, 2, 3, 4, 2, 3,

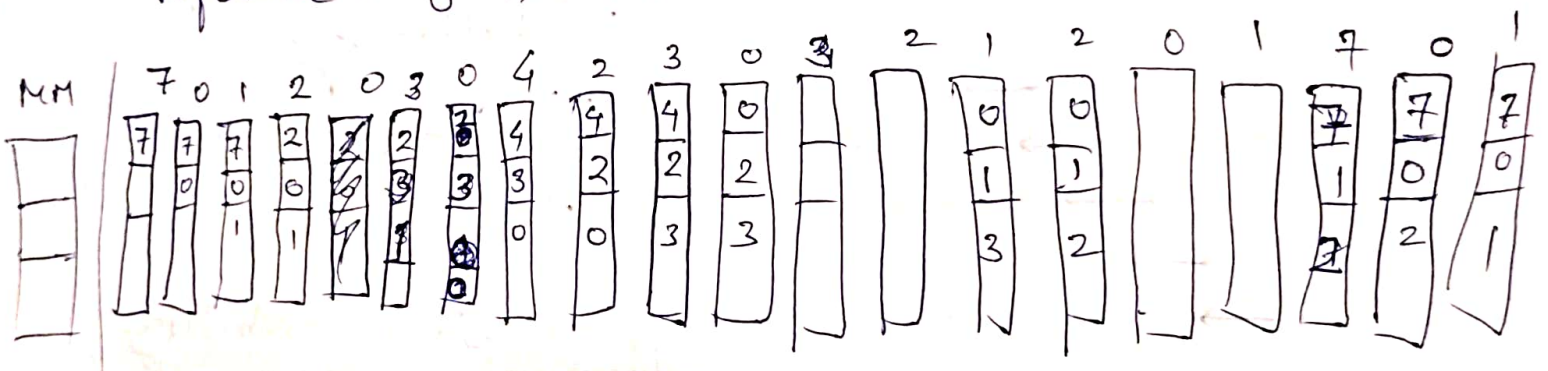


(MFU) Page replacement algo

→ Replaces the page that has highest reference count.

FIFO

ex1: No. of pages in the program = 8 (0 to 7)
No. of frams in MM = 3
Reference string = 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1



page#	Reference count
0	2
1	2
2	2
3	2
4	0
7	1

FIFO queue:
7, 0, 1, 2, 3, 0, 4, 2, 3, 0, 1, 2, 7, 0,

No. of page fault = 15