Date - 08.09.23 Ch-1

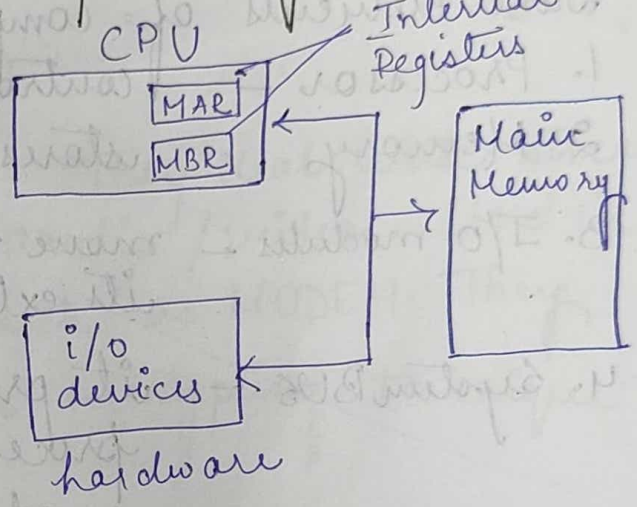
OS → interface b/w user and computer hardware

First OS - MS DOS

Windows - GUI, ease of convenience, time saving, user friendly, multiuser support

* Four basic elements of a computer system:-

1. CPU
2. I/O devices
3. Memory
4. System BUS



⊙ Execution of an instruction :-

Instruction → Fetch → Execute

Memory Address Register (MAR) - holds the address in memory of the next instruction to be executed.

Memory Buffer Register (MBR) - a two-way register that holds data fetched from memory (and ready for the CPU to process) or data waiting to be stored in memory.

Date - 12.09.23

Q. What is an operating system?

Ans → It exploits the hardware resources of one or more processors to provide a set of services to system user. It acts as an interface b/w user application & hardware

Basic elements of computer :-

1. Processor - controls the operation of computer & perform data processing
2. Memory - stores the data & program
3. I/O modules - move the data between computer & its external environment
4. System BUS - it provides for communication b/w processors, main memory & i/o modules.

Two main registers which are present in processor are :-

① MAR - specifies the address in memory for the next read/write

② MBR - contains the data to be written in memory or recieves the data read from memory

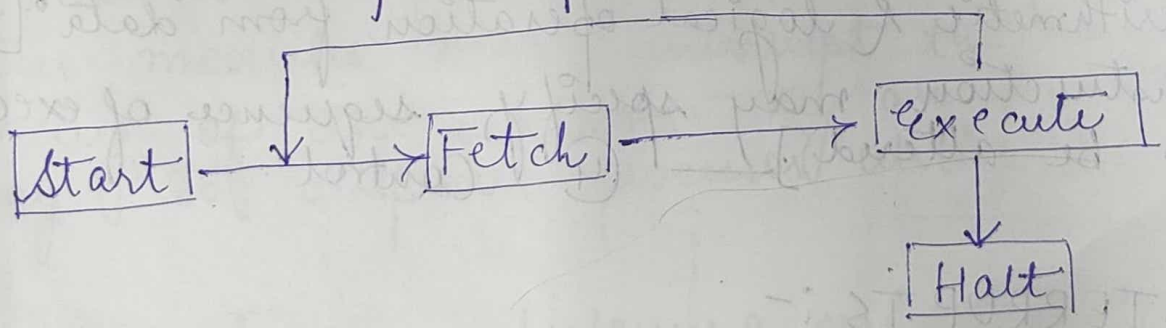
* GPU uses SIMD (Single Instruction Multiple Data)
DSP (for syncing).

* Microprocessor - It is the fastest general purpose processor when a processor is present on a single chip. However, each chip may contain multiple processors

* GPU - They provide efficient computation on arrays of data using SIMD technique. Used for general numerical processing.

* Digital Signal Processor (DSP) - They deal with stream-ing signals such as audio / videos. Used in embedded system devices such as MODEM. They provide security through encryption technique.

* SOC (System on chip) - To satisfy the requirement of hand-held devices, the microprocessor are giving way to SOC. Components such as DSP, GPU, Cortex & main memory in addition to memory are on a single chip.



* Instruction Execution -

1. A program consists a set of instructions
2. Instruction processing consists two steps - fetch & execute

3. The processing required for a single instruction is called Instruction cycle. In IC, the PC holds the address of the next instruction to be fetched.

4. The processor always increments the PC after each instruction ~~step~~ fetch so it will fetch the next instruction in sequence.

5. The fetched instruction is loaded in IR (Instruction Register) it contains the bits that specifies the action processor has to take.

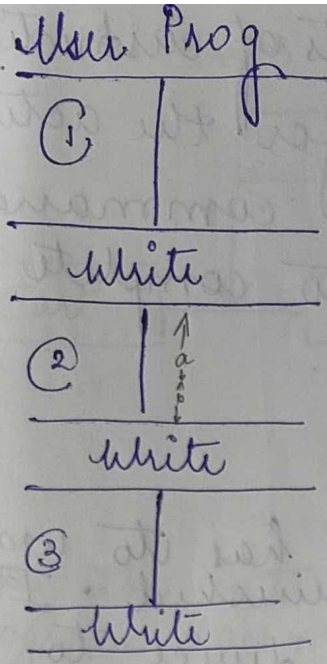
6. There are 4 types of action :-

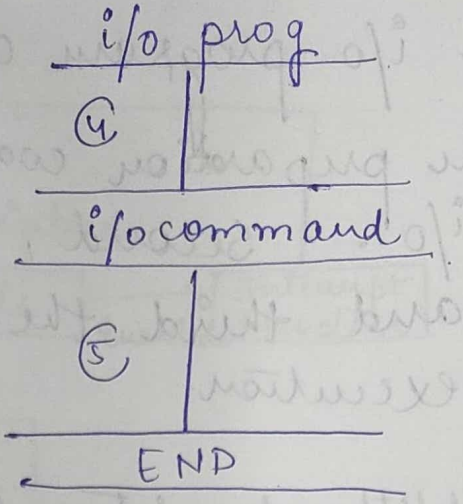
① Processor - Memory → data may be transfer-red from processor to memory or vice-versa.

② Processor - i/o → Data may be transferred to or from any peripheral device.

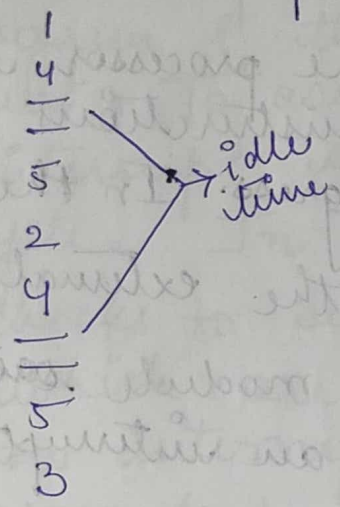
③ Data Processing - The processor may perform arithmetic & logical operation from data. [Any instruction may specify sequence of execution to be altered.] ④ Control.

* INTERRUPTS :-





Without interrupt



With interrupt

1
4
2a
5
2b
4
3a
5
3b

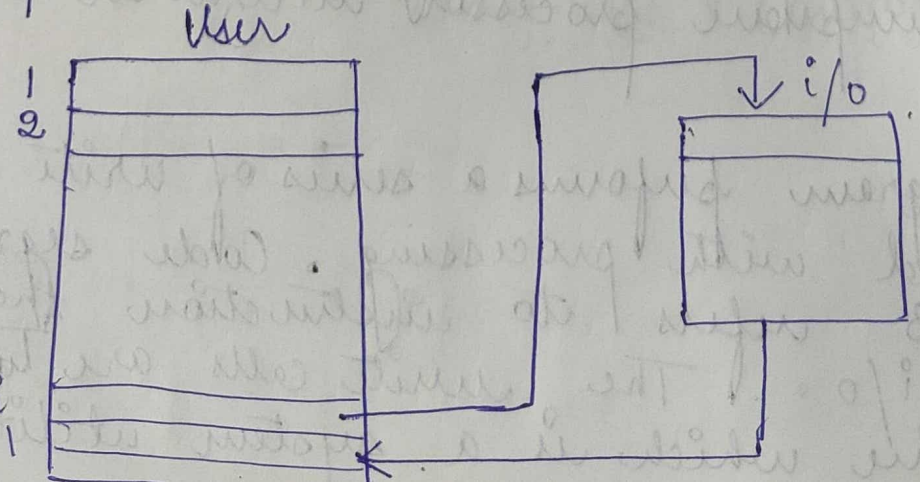
t mechanism is provided by all computers by which other modules may interrupt the normal sequencing of the processors. They are provided as a way to improve processor utilization.

The user program performs a series of write calls inter-leaved with processing. Code segments 1, 2 and 3 refers to instruction that do not need i/o. The write calls are to an i/o routine which is a system utility.

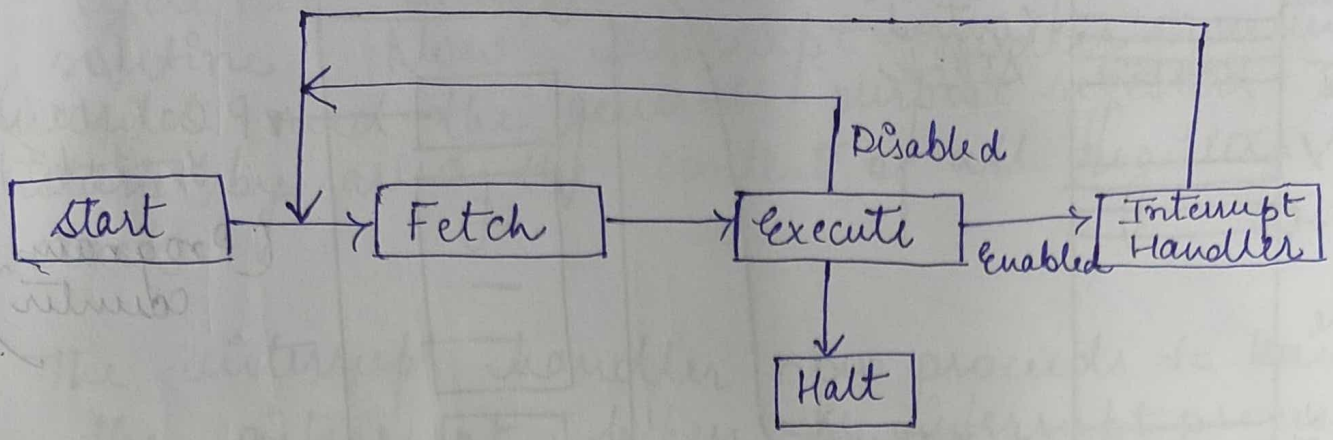
The i/o program consists of 3 sets of instructions:
1st, The preparation code to prepare for the actual i/o. Second, the actual i/o command and third the instructions to complete the execution.

⊙ Without interrupts the processor has to wait until the i/o routine is finished. It takes a long time for the external device to finish process until then the time of processor is wasted.
But, with use of interrupts the processor is engaged in executing other instructions while an i/o operation is in progress. In the example we can see, when the external device completes its processing, the i/o module ~~[illegible]~~ for that external device sends an interrupt request signal to the processor.

The I/O program that is invoked consists only of preparation code & actual i/o command. In this way, processor works concurrently & (with the i/o program) improves its utilization

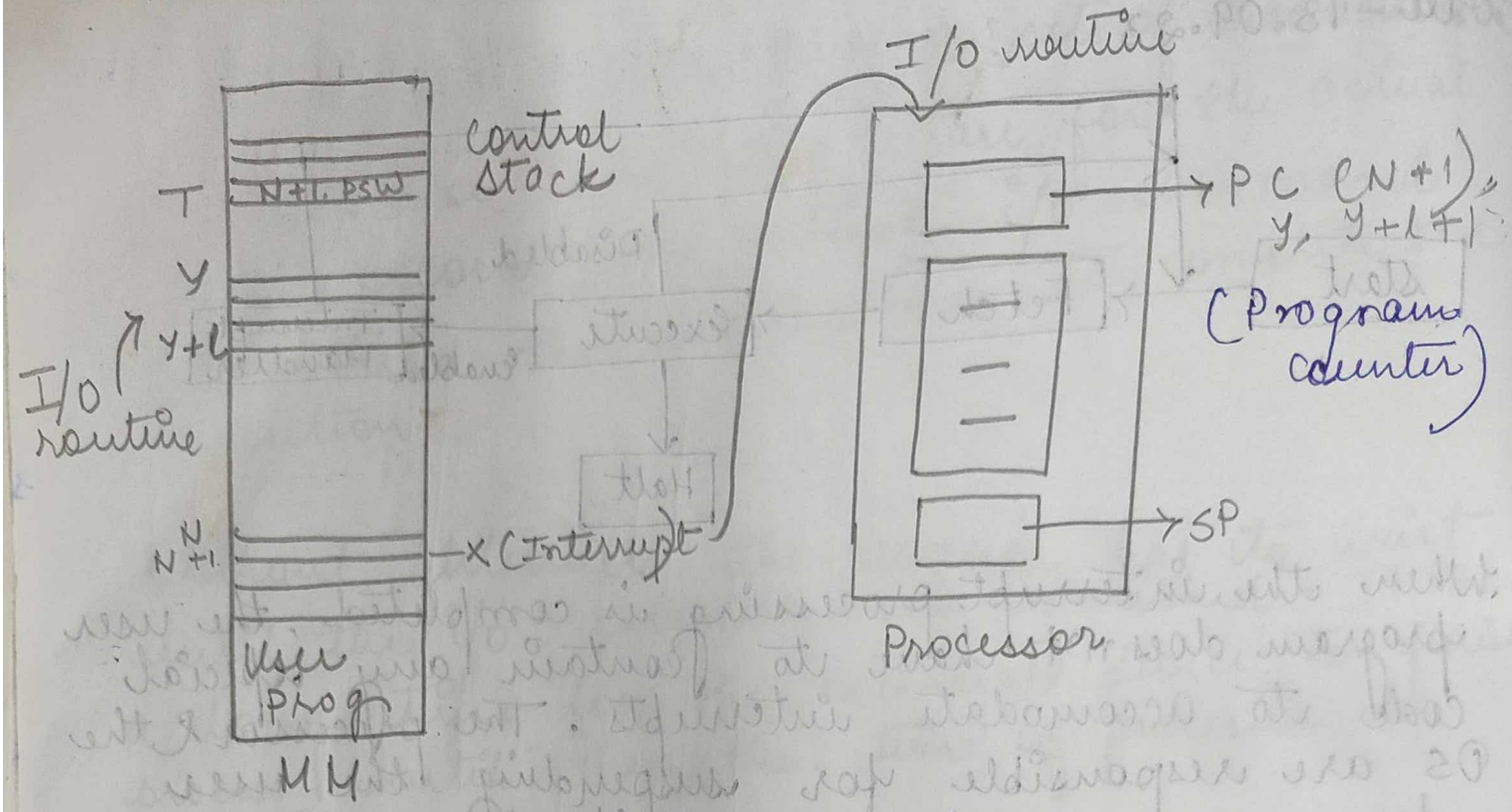


Date - 13.09.23



When the interrupt processing is completed, the user program does not have to contain any special code to accomodate interrupts. The processor & the OS are responsible for suspending the user program & then resuming it from the same point. To accomodate the interrupts, an interrupt stage is added to the instruction cycle. In the interrupt stage, the processor checks to see if any interrupts have occured which are indicated by the interrupt signal.

- If no interrupts are pending, the processor proceeds to the next step & fetches the next instruction of the current program.
- If an interrupt is pending, the processor suspends the execution of current program & executes the interrupt handler routine. This routine determines the nature of interrupt & performs whatever actions to be taken.



Refer flowchart from book (Pg 41)

* <u>Interrupt Processing</u> :-

1) The device issues an interrupt signal to the processor

2) The processor finishes the execution of the current instruction before responding to the interrupt

3) The processor tests for a pending interrupt request determines there is one & sends an acknowledge signal to the device that has issued the interrupt

4) Now processor needs to prepare to transfer control to the interrupt routine. It saves the information needed to resume the current program i.e PSW (Program status word) and the location of next instruction to be executed. The processor then loads the program counter with the entry location of the interrupt handling routine that will handle the interrupt

5) Once the PC has been loaded, next instruction cycle is started from interrupt handler routine. Now, interrupt handler routine would need the general purpose registers so its starts by saving the content of all registers to the stack.

6) The interrupt handler now proceeds to process the interrupt. When the interrupt processing is complete, the saved values are taken back from the control stack. The final act is to restore the values of PSW and PC values from the stack.

* Multiple Interrupts :-

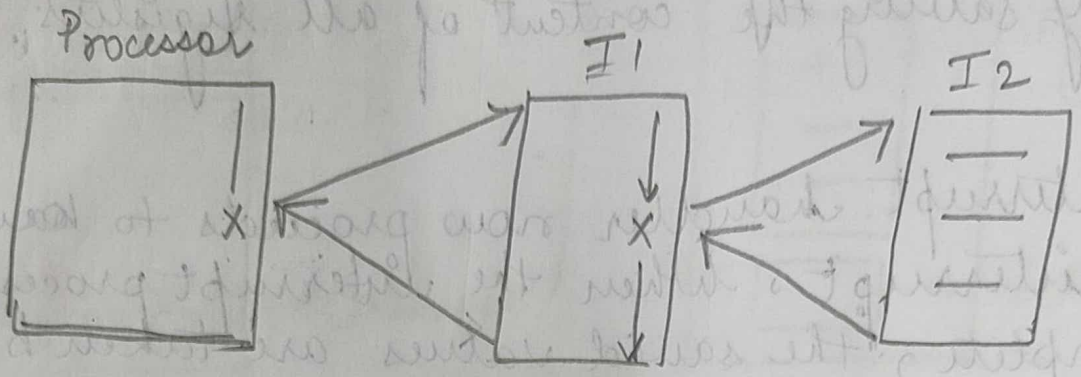
→ One or more interrupts can occur while an interrupt is being processed. So, there are two approaches that can be taken in account to deal with multiple interrupts. The first one, is to disable the interrupt is processed. & disabled interrupt means that the processor will ignore any new interrupt signal. If an interrupt comes during this time. It will remain pending & will be checked by processor after the processor has enabled interrupts. Thus, if an interrupt occurs when a user program is executing, the interrupt is disabled immediately.

The drawbacks of disabled interrupt :-

It does not take into account relative priority & time critical needs.

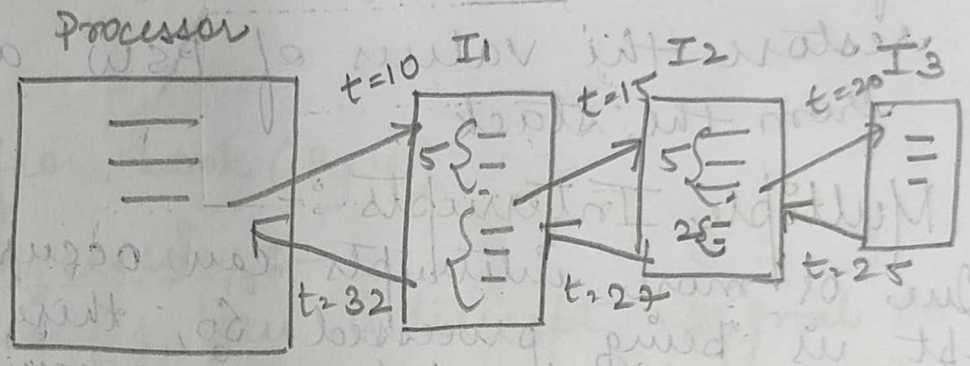
forex- If any interrupt comes from communication line, its priority is higher & needs to be serviced immediately. But, as the interrupts are disabled, it has to wait for a very

long time until the previous interrupt has executed its execution.



Priority

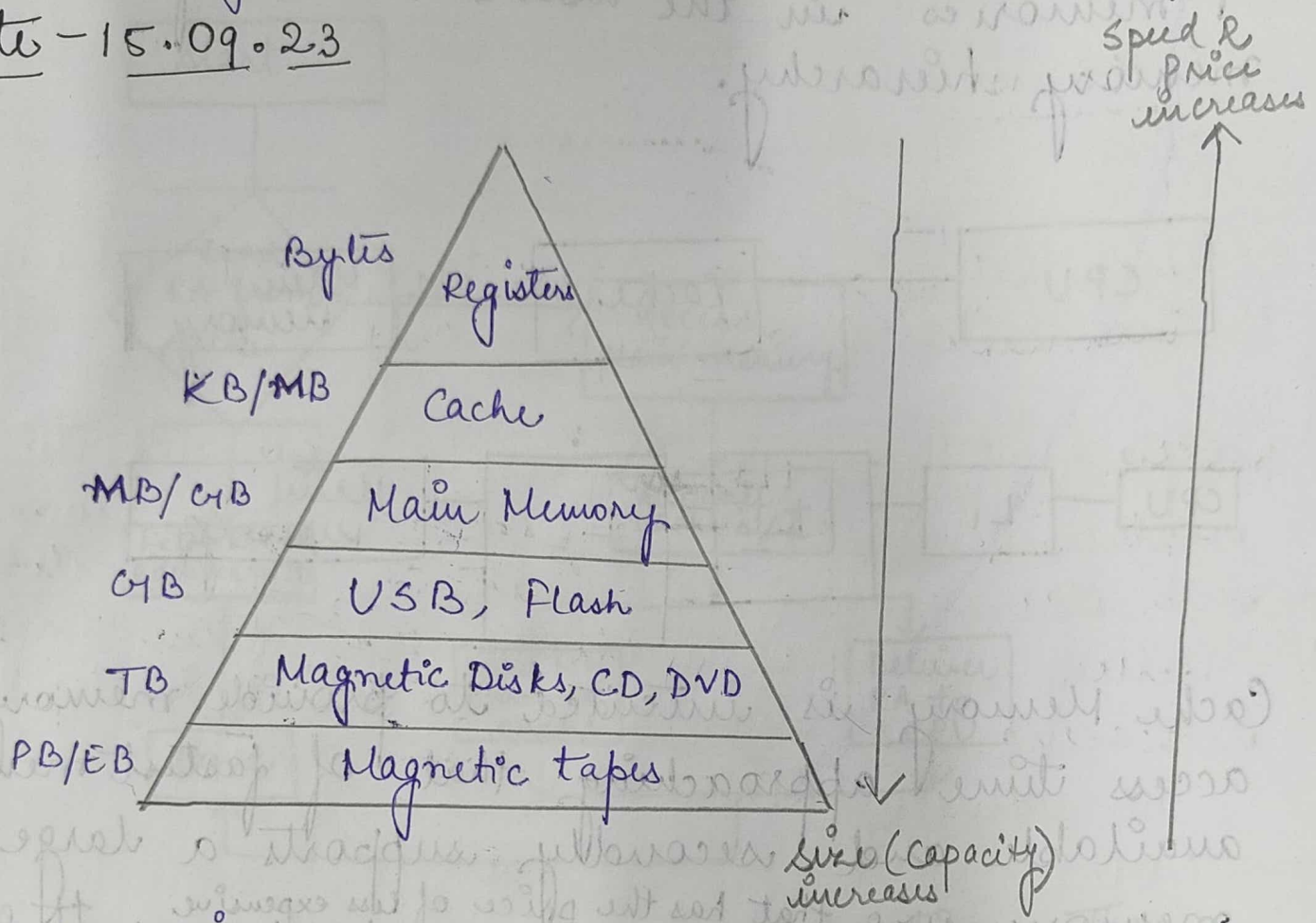
	t=0, 40s
2	t=10, 10s
4	t=15, 7s
5	t=20, 5s



→ Suppose there are 3 i/o devices, a printer, a disk and a communication line, with the priorities 2, 4 and 5. A user program begins at t=0, at t=10, a printer interrupt occurs, the user information is stored on control stack and the execution continues at the printer interrupt service routine. While this routine is executing, another interrupt from communication line arise at t=15. As the communication line has higher priority, the interrupt request of this i/o device is serviced. At t=20, an interrupt with much higher priority arives, so the processor services the interrupt request of this I/O device. When all the interrupts are handled, the previous processes are restored which is the execution of printer interrupt. After the interrupt routine

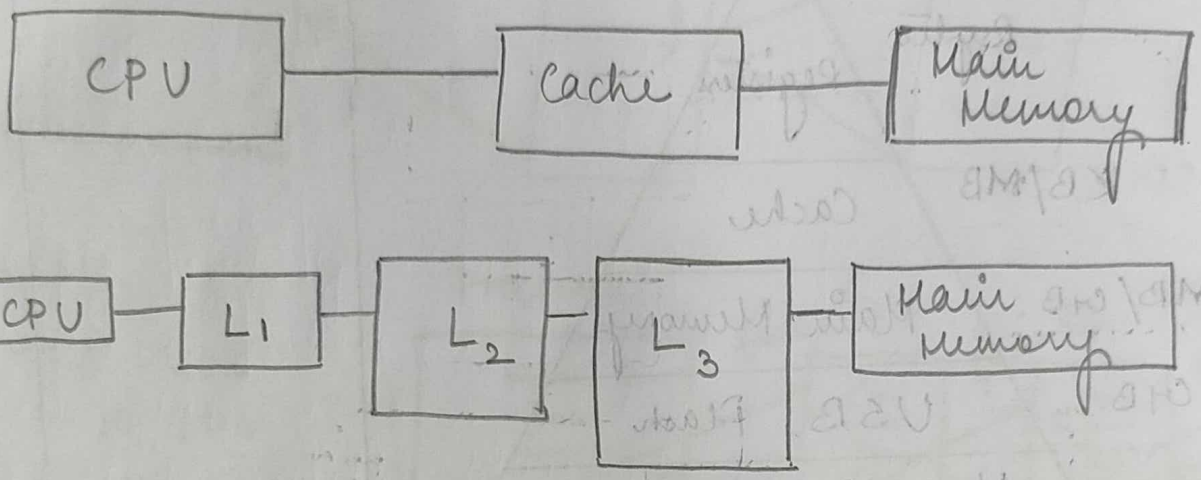
of printer is completed, the execution of the user program is resumes.

Date - 15.09.23



⊙ The memory in computer system is divided in the form of levels. The smaller, more expensive, faster memories are supplemented by larger cheaper & slower memory. The smallest fastest & most expensive type of memory consists of registers, skipping to two levels, main memory is the principal internal memory of the computer. The cache is usually visible to the programmer for staging the movement of data b/w main memory & registers. These 3 types of memory are volatile. External non volatile memory is referred to as secondary memory which includes USB & flash memory. These store program & data files. The magnetic disks

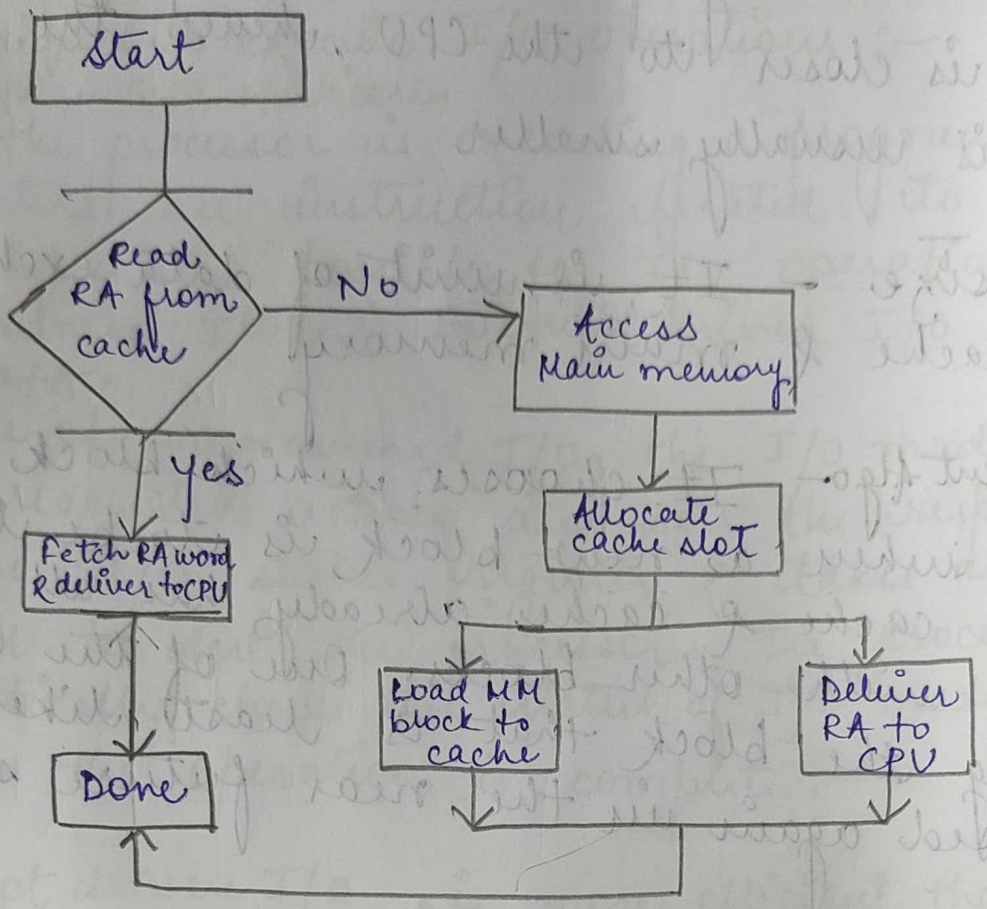
and magnetic tapes are more ...
of the data & are slower than the other memories in the above levels of the memory hierarchy.



Cache Memory is intended to provide memory access time approaching that of fastest memories available and secondly supports a large memory size that has the price of less expensive type of memory. It contains a copy of the portion of the main memory. When the processor attempts to read a byte or a word of memory, a check is made to determine if the word or byte is present in cache. If it is there, the byte/word is delivered to the CPU. If not, a block of main memory consisting of fixed words is read into the cache, then the byte/word is delivered to the CPU because of the concept of locality of reference. When a block of data is fetched into the cache to satisfy a single memory reference. It is likely the near future

memory references will be to other bytes of the block

RA - read address



The figure illustrates the read operation. The processor generates the address RA of a word to be read. If the word is contained in cache it is delivered to the processor otherwise the block containing that word is loaded into the cache & the word is delivered to the processor. Now, there are design issues that must be addressed in dealing with virtual memory & cache memory.

1. cache size
2. block size
3. replacement algo.
4. mapping func
5. write policy
6. no. of cache levels

} design issues

1. Cache Size – According to hierarchy, small caches have significant imp act on performance. Cache is closer to the CPU, hence the size of cache is usually smaller.

2. Block size – It is unit of data exchanged between cache & main memory

3. Replacement Algo – It chooses which block to replace when a new block is to be loaded into the cache & cache already has all slots filled with other blocks. One of the way is replacing the block that is least likely to be used again in the near future such as LRU

4. Mapping func. – When a new block of data is read into the cache, the mapping func^n determines which cache location the block will occupy

5. Write policy – If the contains of a block in cache are altered, then it is necessary to write it back to the main memory before replacing it. One of the way is writing can occur any time when the block is updated. Another way writing occurs only when the block is replaced

6. No. of cache levels – Multiple levels of cache

provide fast access time to the CPU & enhance the performance.

* Techniques to execute I/O operations :-
H/W - SMP, Multicore

When the processor is executing a program and encounters an instruction related to I/O, three techniques are possible for I/O operations. 1) Programmed I/O, 2) Interrupt driven I/O and 3) DMA.

1) In case of programmed I/O, the I/O module performs the requested action and sets the respective bits in the I/O status register & takes no further action to alert the processor. The processor periodically checks the status of I/O module until it finds the operation is complete.

2) Interrupt driven I/O, is more efficient than programm-ed I/O and still requires active intervention of the processor to transfer data b/w memory & I/O module. Its drawback is the processor is busy in managing I/O Transfer & its utilization in running current program decreases. So the solution to this problem is DMA (Direct Memory Access).

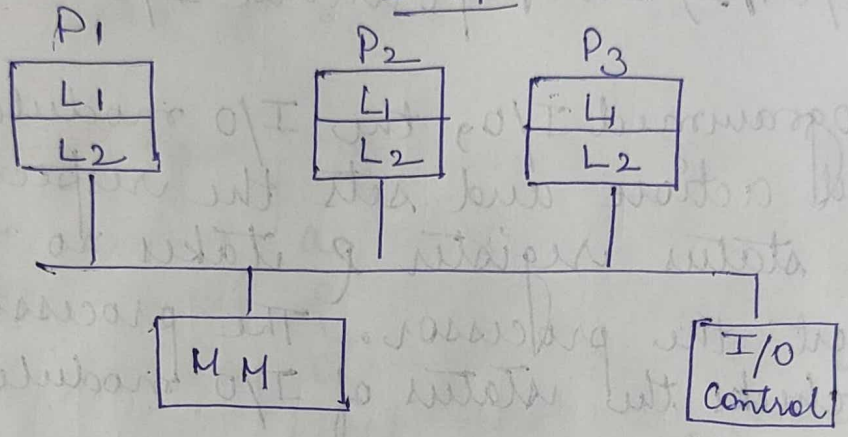
3) When large volumes of data are to be moved, DMA is more efficient technique, the DMA func^n can be performed by a seperate module or it can be combined with I/O module. The processor issues a command to the DMA module by sending these 4 information - 1) whether a read/write is requested, 2) the address of I/O device

involved, 3) the starting location in memory to read data from & write data to and 4) the no. of words to be read or written.

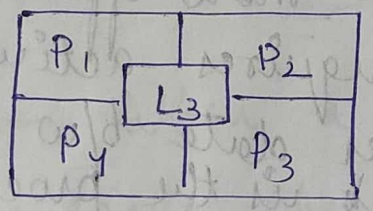
The processor then continues with other work.

* Date - 20.09.23

SMP



Multicore processors



Ch-2 (Operating System Overview)

Goals :-

1.) Convienience
2.) efficiency
3.) ability to evolve

Q. How Operating System acts as a user interface?

Ans → Program development

1. Provides services of editors & debugger
2. For the execution of program, instruction & data must be loaded. I/O devices & files must be initialized & other resources must be prepared is handled by OS.
3. Access to I/O devices
4. Controlled Access to files

→ In a system with multiple users the OS may provide protection management to control access to files.

5. Error detection & response

→ OS must provide a response that clears the error position with the least impact on the running program.

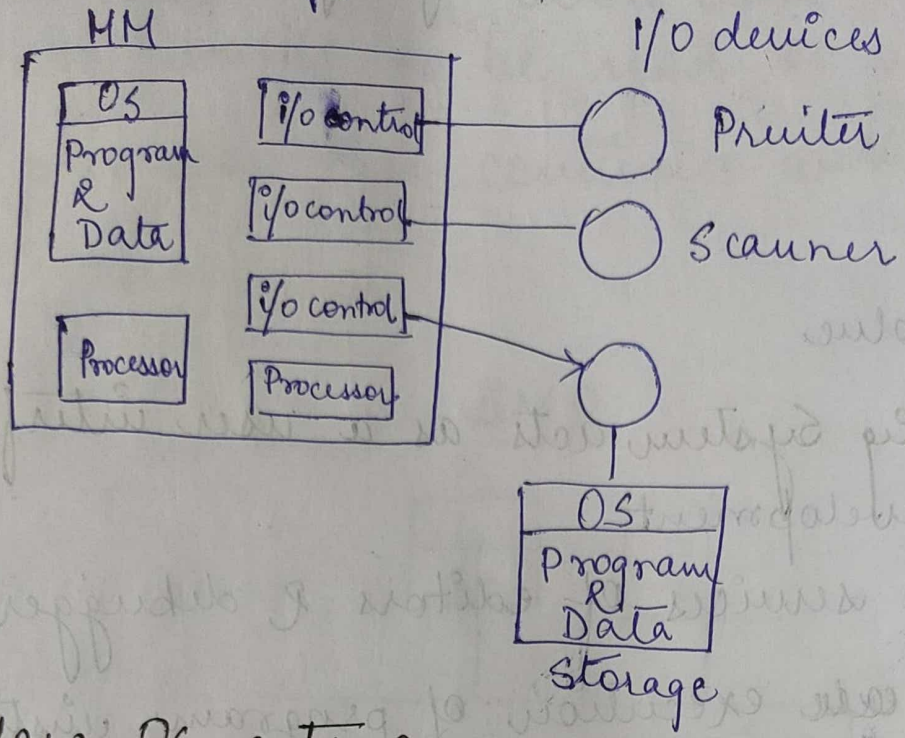
6. ISA

→ The OS has access to addnl. machine level language instructions that manage system resources

7. API

→ It gives a program access to the hardware resources & services available in a system

through different service calls.



Q How OS acts as a resource manager?

Ans → A portion of OS is in the memory which includes the Kernel which contains frequently used functions in the OS. The remainder of main memory contains user & utility programs & data. The OS & memory management hardware in the processor jointly controls the allocation of main memory. The OS decides when an I/O device can be used by a program in execution & controls access to & use of files. The processor itself is a resource & OS determines how much processor time is to be given for the execution of any program.

* Types Of OS –

1. Batch OS

In batch OS, the user has no direct access to the processor instead the programs are submitted on cards or tapes to the computer operator

who batches the jobs sequentially & places the batch on a input device. Each program is constructed to branch back to the computer when it completes processing the monitor automatically loads the next program. The programs are executed in system mode or Kernel mode.

2. Multiprogrammed OS.

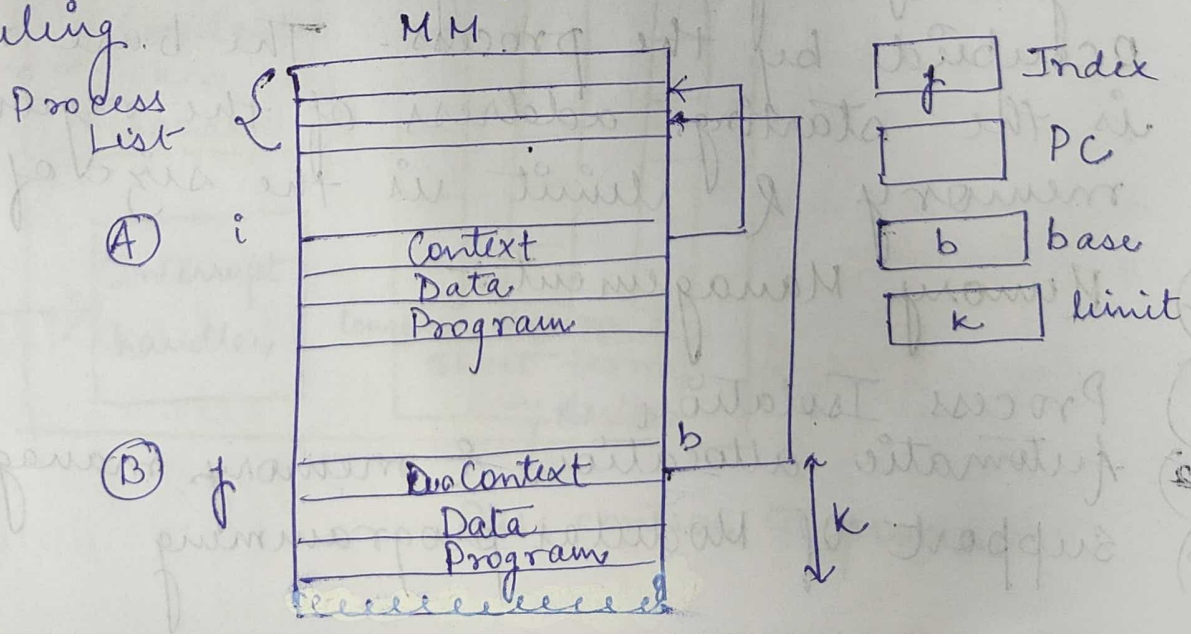
More than one processes are present in RAM. This is the non-primitive approach. The goal is to minimize CPU idle time.

3. Multitasking OS.

Similar to multiprogramming but primitive approach. The goal is to minimize response time.

* Major achievements of any OS:-

1) Processes
2) Memory management
3) Information & Security Measures
4) Scheduling



1) Process

A program in execution or the instance of a program currently running on the computer.

In the diagram two processes A and B exist in portions of main memory. Each process is recorded in a process list build & maintained by the OS.

There are 3 components of a process:-

1) Executable program
2) Associated data
3) Execution context of the program.

The process list contains one entry for each process which includes a pointer to the location of a block of memory which contains the process.

The process index register contains the indexes into the process list of the process currently controlling the processor. The base & limit registers define the region in memory occupied by the process. The base register is the starting address of the region of the memory & limit is the size of region

2) Memory Management:-

1) Process Isolation
2) Automatic allocation & memory management
3) Support of Modular programming

4) Long term storage
5) Protection & Access Control.

A_1		B_5
B_1	A_2	B_3
	A_3	
		B_2
A_4	A_5	

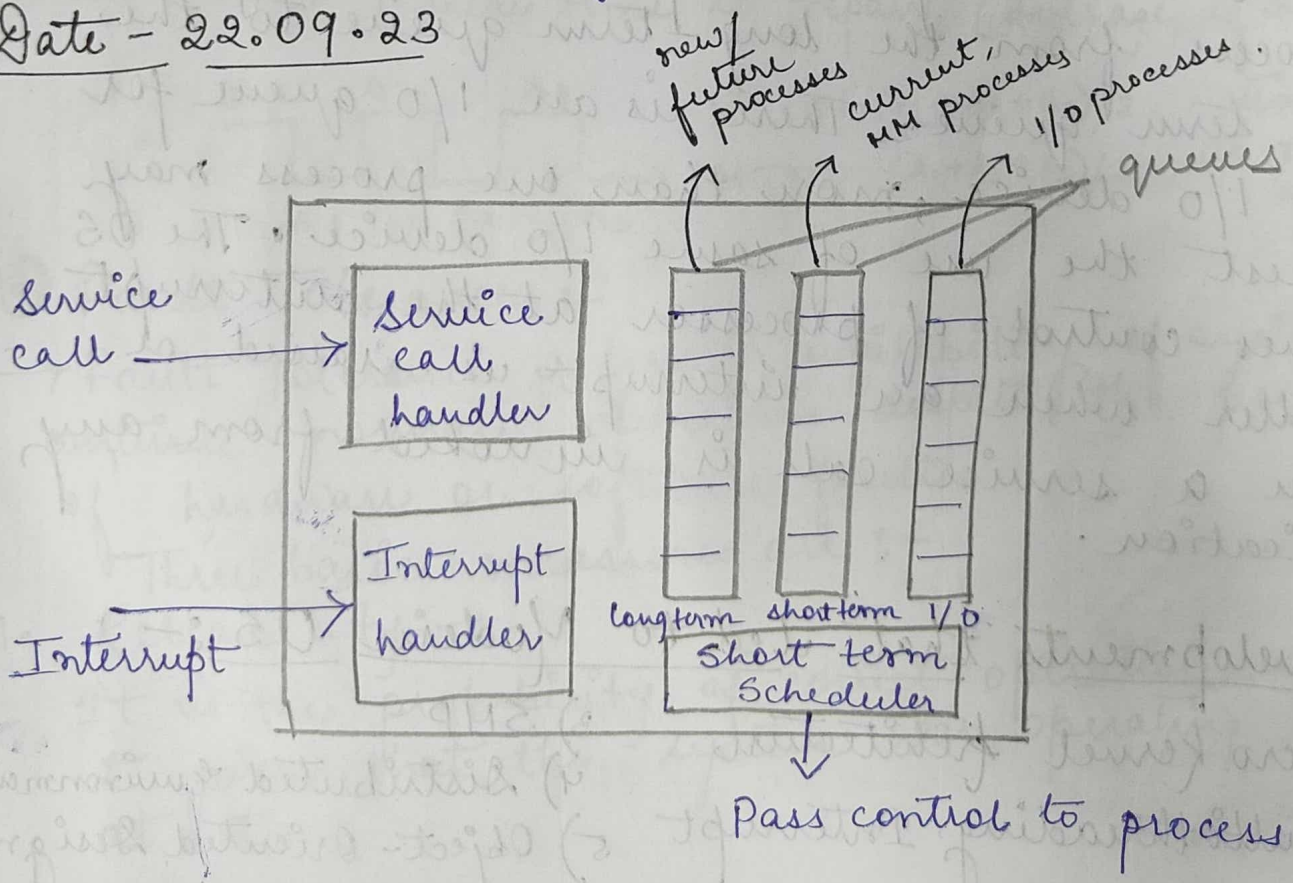
Main memory

A	B
1	1
2	2
3	3
4	4
5	5

Secondary memory.

⊙ Pages & Frames :-
Processes the price of fixed size blocks called pages which are stored in secondary memory. MM consists of no. of frames each equal to the size of a page for the program to execute some or all pages should be present in MM.

Date - 22.09.23



3. Information and Security Measures :-

(i) Availability -
(ii) Confidentiality
(iii) Data Integrity
(iv) Authentication

4. Scheduling and Resource Management :-

→ The OS maintains a no. of queues which contains a list of processes waiting for some resource. The short-term queue consists of processes that are in main memory & are ready to run as soon as processor is made available. The long term queue is a list of new jobs waiting to use the processor. The OS adds the jobs to the system by a transferring a process from the long-term queue to the short term queue. There is an I/O queue for each I/O device, more than one process may request the use of same I/O device. The OS recieves control of processor at the interrupt handler when an interrupt is raised or when a service call is invoked from any application.

* Developments that led to Modern OS :-

1) Micro kernel architecture
2) Multithreading Interrupt
3) SMP
4) Distributed Environment
5) Object-Oriented Design

#. Features Of SMP :-

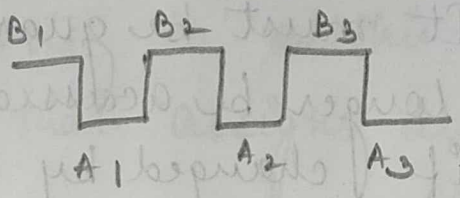
1) Performance
2) Incremental Growth
3) Scaling
4) Availability even after one processor shuts down

#. Faults :-

→ Fault is the hardware or software state resulting from component failure, operator error, design error, data structure error or physical interferences from the environment. They are of 2 types :-
(i) Permanent and (ii) Temporary

⊙ Fault Tolerance :-

→ Reliability
→ MTTF (Mean time to failure) (Average of uptime) when system is functioning correctly
→ MTTR (Mean time to repair) (Average of downtime) time taken to repair the fault



$MTTF = (B_1 + B_2 + B_3)/3$

$MTTR = (A_1 + A_2 + A_3)/3$

⊙ Fault Tolerance :-

→ Fault Tolerance refers to availability of system to continue normal operations despite the presence of hardware or software faults.
Three basic measures are :-

1. Reliability :-
It is the probability of correct operation upto time t given that the system was operating correctly at t = 0

2. MTTF:-
It is the average time available of the system when it was executing correctly.

3. MTTR:-
The average time it takes to repair or replace a faulty element. The time during which system is not available is downtime & when system is available is uptime.

#. Design considerations for Multiprocessors:-

→ A multiprocessor OS must provide all the functionality of multiprogramming system. Some of the design considerations are:-

1. Simultaneous concurrent processes or threads.
2. Scheduling
3. Synchronization
4. Memory Management - It must be guaranteed that old data can no longer be accessed & needs to be updated if changed by any process.
5. Reliability and Fault tolerance.

H/W - Multicore OS Considerations.

⊙ OS Design considerations for multiprocessors & multicore

Issues - scale of potential parallelism
- efficiently harness the multicore processing

→ Parallelism with Applications – GCD (Grand Central dispatch)
→ Virtual Machine Approach

Date - 26.09.23

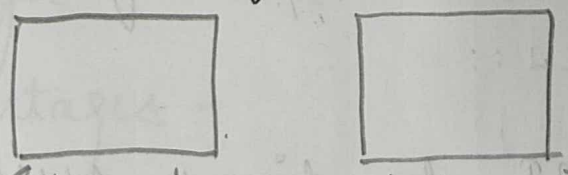
* Multicore OS Considerations :-

1. Parallelism :-
Many applications can be subdivided into multiple tasks that can be executed in parallel. It is the duty of developer to how to split the application work into into independent executable task. GCD is a multicore support capability in which the OS maps, tasks onto the threads.

2. Virtual Machine Approach :-
One or more cores are dedicated to a particular process. Then processor devotes all its efforts to ~~all~~ this process. It is the duty of OS, that assigns an application, a processor or some memory and the program itself using metadata generated by the compiler

* Windows Architecture :-

Windows 1983
Windows NT 1995



Kernel

1. Execution Services
2. Microkernel
3. HAL (Hardware Abstraction Layer)
4. Device drivers
5. Windows Graphic Interface

User

1. Special processes
2. Environ. subsystem → Environment subsystem
3. User Application

(Kernel mode)

1. Executive Services –
It makes the low level of Kernel mode. There are different modules such as I/O manager, cache manager, object manager, security manager, power manager, virtual memory manager & process manager.

2. Microkernel –
It is the heart of the kernel and it is the central element of new technology. Its primary task is to schedule all active system threads while maintaining a procedure, operating environment at its highest performance level.

3. HAL –
It maps between generate hardware commands and responses and those unique to a specific ~~special~~ platform

4. Device Drivers :-
These include h/w device drivers that translates user I/O function calls into specific hardware device I/O request.

5. GUI :-
Implements the GUI functions such as user interface controls

(User mode)

1. Special processes :-

They manage the system such as session manager, authentication subsystem, service manager and log in process.

2. Environment Subsystem :-

It includes the subsystem process that deals with responsiveness of applications

3. User Application :-

executable (exe) and (dll)s that provide the functionality users run to make use of system.

* Client / Server Mode :-

A client which can be the application program or another server program request a service by sending a message. The message is routed through executive to the appropriate server, here server is a subsystem of processes that provides services. The server performs the requested operation & returns the result & status information by means of another message which is routed through the executive back to the client.

Advantages -

1. It simplifies the executive.
2. Improves reliability
3. Provides a suitable base for distributive computing

* Windows Objects :-

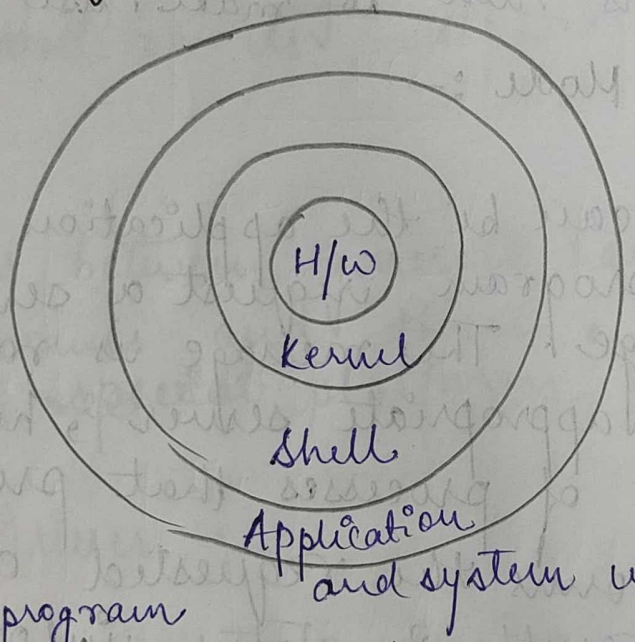
1. Encapsulation
   The only way to access the data in an object is by invoking one of object services.

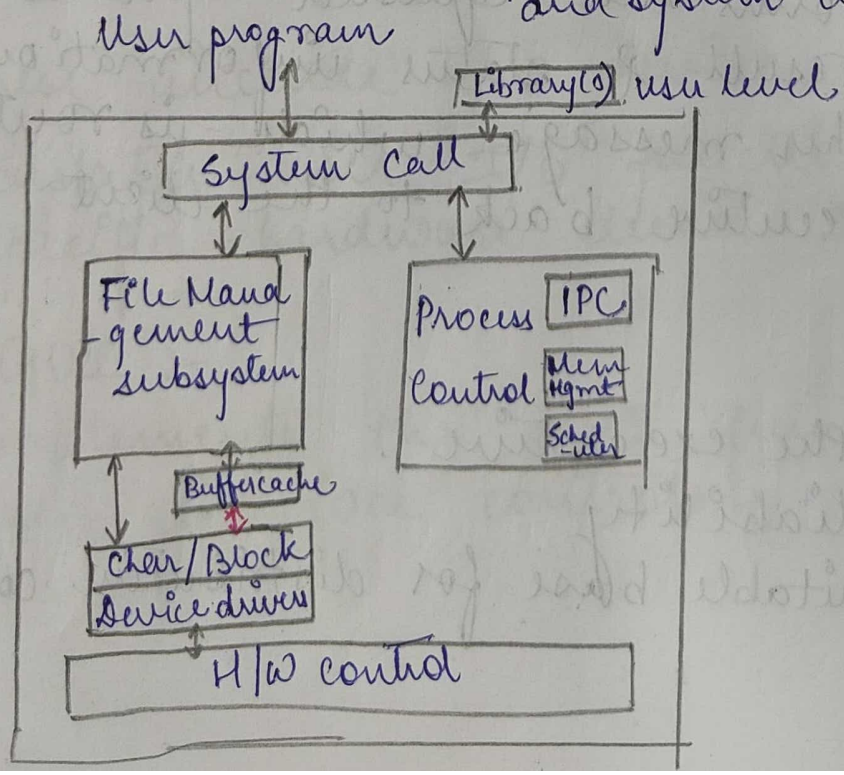
2. Inheritance

3. Object Class and Instance (Abstraction)

4. Polymorphism.

* UNIX Architecture :-





→ User programs can invoke OS services either directly or through library programs. The system call interface is the boundary with the user & allows software to gain access to specific kernel functions. System has two main parts — (1) Process control and (2) File management and I/O.

→ The process control subsystem is responsible for memory management, IPC and scheduling and dispatching of processes. The file system exchanges data between memory & external devices either as ~~process~~ stream of characters or in blocks. For this variety of device drivers are used.

⊙ Modern UNIX system -

SVR4
BSD
Solaris 11

Linux

→ Linux follows modular structure. The modules can be loaded & unloaded on demand. They have two main characteristics:- (1) Dynamic linking (2) Stackable Module.

* Components of Linux :-

1. Signals - The kernel uses signals to call into a process
2. System call - It is the means by which a process requests a specific kernel service

3. Processes and Scheduler
4. Virtual Memory

Date - 27/09/23 (Process & Process Control Block)

Identifiers
State
Priority
Memory Pointers
Context Data
I/O status info
Accounting info
PC

fig - PCB

The process can be uniquely characterised by a number of elements :-

(i) Identifiers - unique id associated with the process

(ii) The state of the process

(iii) Priority - priority level relative to other processes

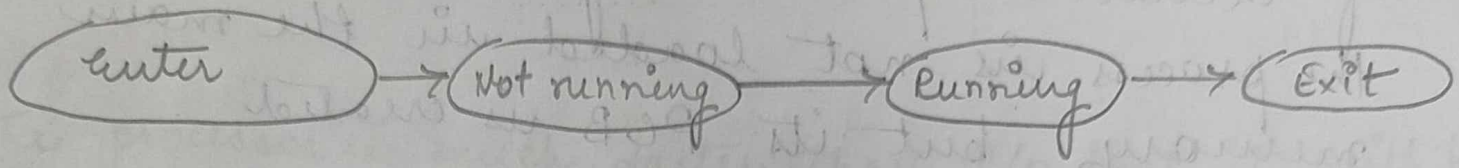
(iv) Memory Pointers - pointers to the program code & data associated

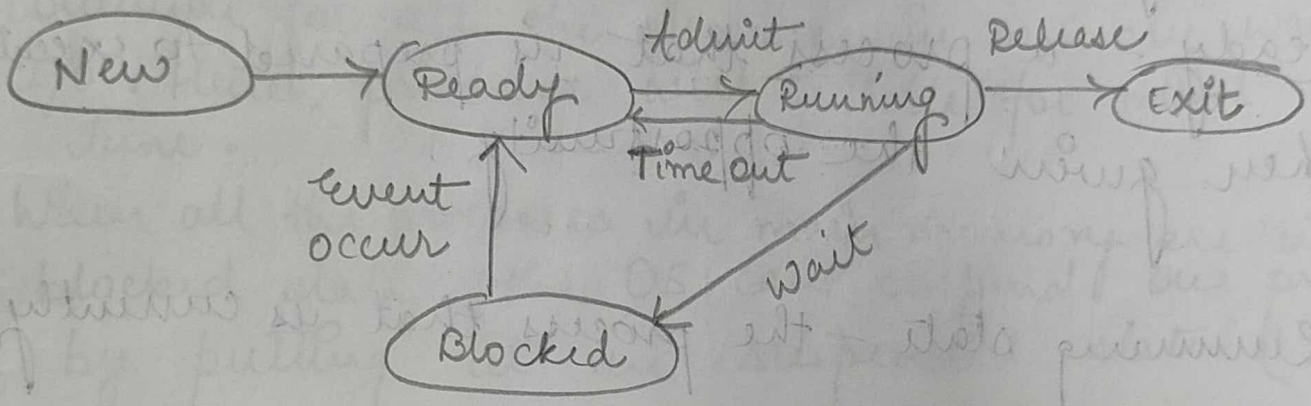
(v) Context data - Data that are present in registers while the process is executing

(vi) I/O status info - includes outstanding I/O request, I/O devices assigned to process, a list of files in use by the process.

(vii) Accounting Info - amount of processor time, clock time or time limits for this process

(viii) PC - the address of next instruction





1) 2 state process Model - (Diag 1)

→ When OS creates a new process, it creates a process control block for the process & enters that process into the system in the not running state. The process exits and is waiting for its turn to execute.
From time to time, the currently running process will be interrupted & the scheduler will select some other process to run. The process in not running state is moved to running state when it gets the turn

→ All the new processes and the process that are interrupted are transfered to the queue of not running state. So, we split the not running state to ready & block state

Diag2 (5 state process Model) :-

i) New - a process that has just been created but has not been admitted into the pool of executable processes by the OS. It means the process is not loaded in the main memory but its PCB is created.

ii) Ready - a process that is prepared to execute when given the opportunity.

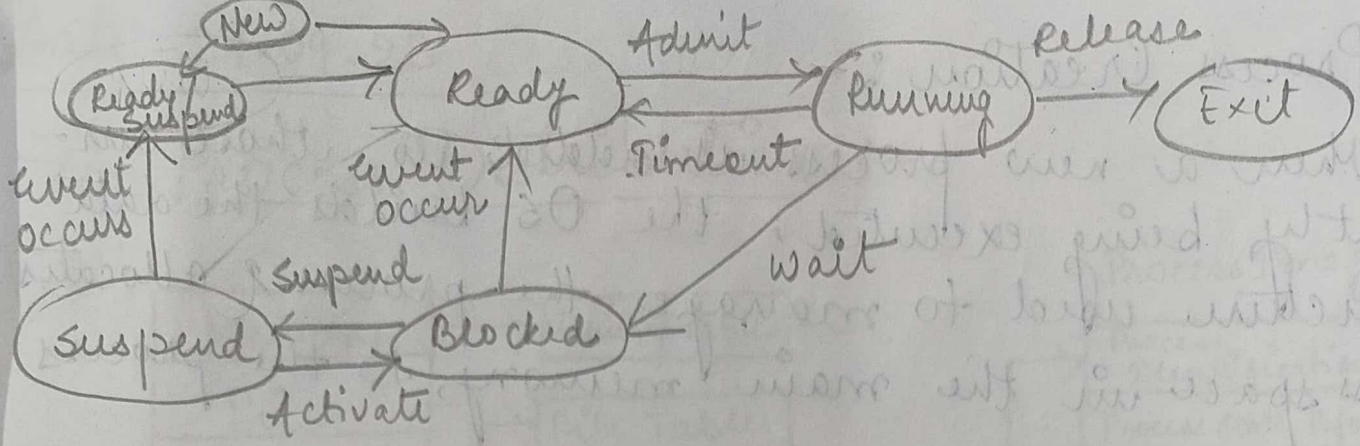
iii) Running state - the process that is currently being executed.

iv) Blocked - a process that can't execute until some event occurs such as completion of I/O

v) Exit - the process that has completed the execution & has been released by the processor.

* Key point :-

→ While a process is in new state, information regarding the process is maintained in the control tables in main memory. However the process itself is not in main memory



The processor is much faster than I/O that it will be common for all the processes in memory waiting for I/O. Hence, processor will be idle for most of the time.

When all the processes in main memory are in blocked state, the OS can suspend one process by putting it in suspended state & transfering it to secondary memory.

i) Ready – the process is in main memory & ready for execution

ii) Blocked – the process is in the main memory & awaiting an event.

iii) Suspend – the process is in secondary memory and awaiting an event

iv) Ready Suspend – the process is in secondary memory but is available for execution as soon as it is loaded in the main memory

Process Creation :-

→ When a new process is added to those currently being executed, the OS builds the data structure used to manage the process & allocates address space in the main memory to the process

⊙ Reasons -

1) New application & new batch job created
2) Created by OS to provide a service
3) Process spawning - a currently managed process can create a new process
4) Interactive log-on - a user in terminal log-on(s) to the system.

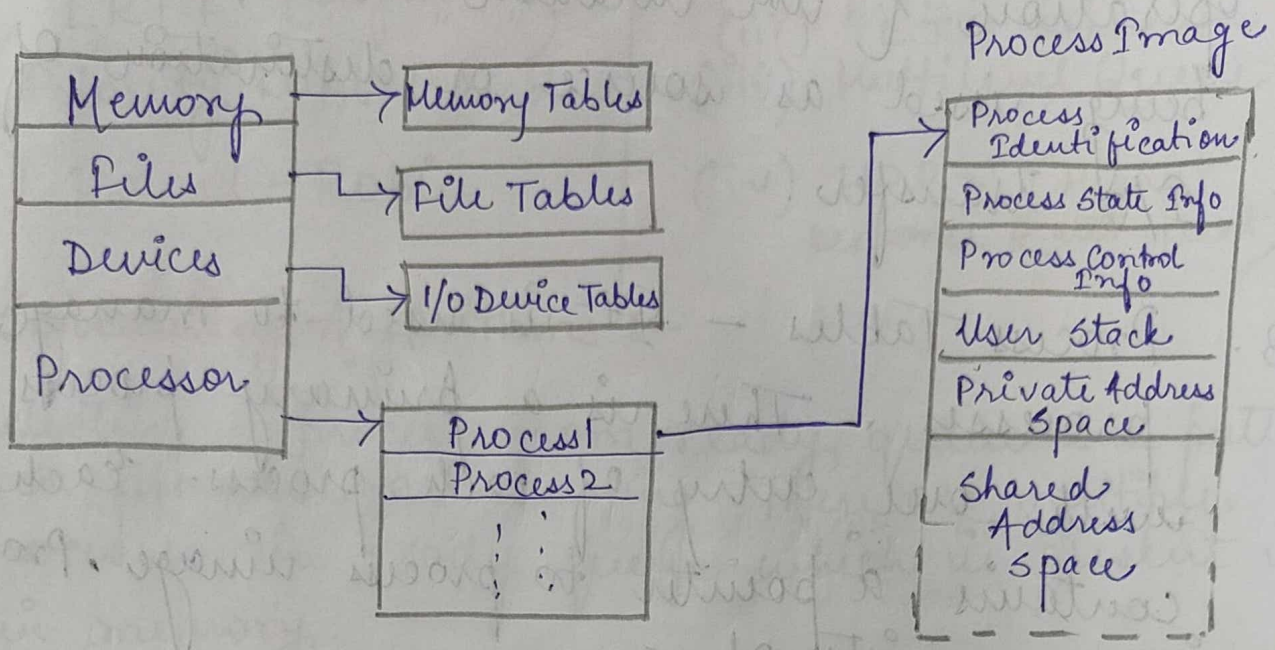
Process Termination :-

⊙ Reasons -

1) Normal completion
2) Requires more memory than the system can provide
3) Bounds Violation - the process tries to access a memory location that is not allowed to access
4) Arithmetic Error
5) I/O Failure
6) Invalid Instruction - the process attempts to execute a non existing instruction.
7) Parent process termination
8) Time limit exceeded

Date - 29.09.23

Process Distribution :-



If OS wants to manage processes and resources it must have information about the current status of each process and resource. The OS manages this information in four categories :-

1. Memory tables - These are used to keep track of both main & secondary memory. The memory table includes the following information -
   i. The allocation of MM to processes
   ii. The allocation of secondary memory to processes
   iii. Any information needed to manage virtual memory.
   iv. Any protection attributes of blocks of main memory.

2. File Tables - It provides information about the existence of files, their location on secondary memory & their current status. I/O tables are used to manage the I/O

devices and channels of computer system. OS needs to know the status of I/O operation & the location in main memory being used as source or destination of the I/O transfer.

3. Process Tables – It is used to manage processes. There is a primary process table with one entry of each process. Each entry contains a pointer to process image. Process image consists of –

i. Process identification – unique process identifier

ii. Processor state info – it consists of contents of processor register which contain the information of a running process in the register

iii. Process Control Information – it consists of different flags such as resume flag, interrupt enabled flags & others to control the process. It also consists of private address space & shared address space.

* Scheduling :–

Premptive	Non-Premptive
(i) SRTF (Shortest Remaining Time First)	(i) FCFS (First come first serve)
(ii) LRTF (Longest " ")	(ii) SJF (Shortest Job first)
(iii) Priority Level Based	(iii) LJF (Longest " ")
(iv) Round-Robin	(iv) Multilevel Queue
	(v) HRRN (Highest Response Ratio Next)

Scheduling Algorithms :-

→ Selecting a process from ready queue and putting it on processor for running. There are multiple processes in ready queue which is present in main memory.

⟹ Times :-

1. Arrival Time (static time when process arrives in ready queue)
2. Burst Time (time duration taken by the process in processor)
3. Completion Time (total time to execute)
4. Turn-around Time = (Completion Time - Arrival Time).
5. Waiting Time = (TAT - Burst Time)
6. Response Time = (time when process gets the CPU for the first time) - Arrival Time

① Arrival Time

The time at which the process enters the ready queue or state

② Burst Time

Time required by a process to get executed on CPU (Time duration)

③ Completion Time

Time at which process completes execution

④ Turn-around Time

Completion Time − Arrival Time

⑤ Waiting Time

Turn-around time − Burst Time

⑥ Response Time

The first time when processor gets the CPU − Arrival time

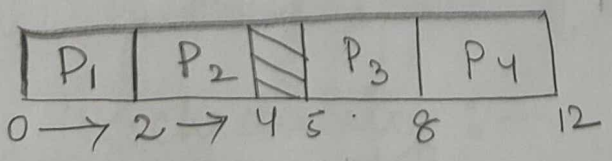
Date - 03.10.23

* FCFS (First Come First Serve)

PNo	AT	BT	CT	TAT	WT	RT
P_1	0	2	2	2	0	0
P_2	1	2	4	3	1	1
P_3	5	3	8	3	0	0
P_4	6	4	12	6	2	2

equal in case of non-preemptive

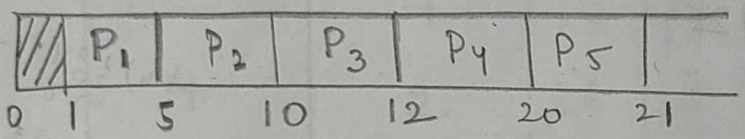
* Gantt Chart :-



Avg. TAT = $\frac{14}{4} = 3.5$

Q. FCFS

PNo	AT	BT	CT	TAT	WT	RT
P_1	1	4	5	4	0	0
P_2	3	5	10	7	2	2
P_3	4	2	12	8	6	6
P_4	7	8	20	13	5	5
P_5	8	1	21	13	12	12

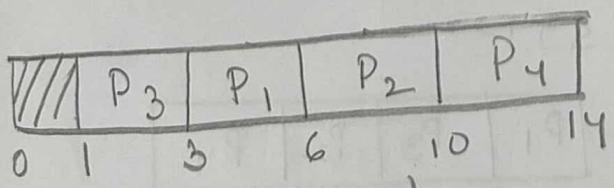


$$\text{Avg TAT} = \frac{45}{5} = 9$$

$$\text{Avg WT} = \frac{25}{5} = 5$$

Q. SJF (Shortest BT is taken first)

PNo	AT	BT	CT	TAT	WT	RT
P_1	1	3	6	5	2	2
P_2	2	4	10	8	4	4
P_3	1	2	3	2	0	0
P_4	4	4	14	10	6	6



$\{P_1, P_3\}\{P_1, P_2\}\{P_2, P_4\}$

BT same, now AT is checked.

$$\text{Avg TAT} = \frac{25}{4} = 6.25$$

$$\text{Avg WT} = \frac{12}{4} = 3$$

SJF

PNo	AT	BT	CT	TAT	WT	RT
P_1	2	3	5	3	0	0
P_2	0	4	12	12	8	8
P_3	0	2	2	2	0	0
P_4	3	3	8	5	2	1

P_3	P_1	P_4	P_2

0 2 5 8 12

P_2, P_3 P_2, P_1 P_2, P_4

4) 22 (5.5
20
20
0

4) 9 (2.2
8
10
8

4) 27 (6.7
24
30
28

$$\text{Avg TAT} = \frac{22}{4} = 5.5$$

$$\text{Avg WT} = \frac{10}{4} = \frac{5}{2} = 2.5$$

$$\text{Avg CT} = \frac{27}{4} = 6.75$$

PNo	AT	BT	CT FCFS	CT SJF	TAT FCFS	TAT SJF	WT FCFS	WT SJF
P_1	0	3	3	3	3	3	0	0
P_2	1	5	8	10	7	9	2	4
P_3	3	2	10	5	7	2	5	0
P_4	9	4	14	14	5	5	1	1

FCFS

P_1	P_2	P_3	P_4

0 3 8 10 14

SJF

P_1	P_3	P_2	P_4

0 3 5 10 14

Avg TAT (FCFS) $= \frac{22}{4}$

Avg WT $= \frac{8}{4} = 2$

Avg TAT (SJF) $= \frac{19}{4}$

Avg WT $= \frac{5}{4}$

H/W

Q FCFS and SJF

PNo	AT	BT	CT		TAT		WT		RT
			FCFS	SJF	FCFS	SJF	FCFS	SJF	
P_1	3	1	14	7	11	4	10	3	Same as Waiting Time
P_2	1	4	10	16	9	15	5	11	
P_3	4	2	16	9	12	5	10	3	
P_4	0	6	6	6	6	6	0	0	
P_5	2	3	13	12	11	10	8	7	

FCFS :-

P_4	P_2	P_5	P_1	P_3

0 6 10 13 14 16

Avg TAT $= \frac{11+9+12+6+11}{5}$

$= 49/5 = 9.8$

Avg WT $= \frac{10+5+10+8+0}{5}$

$= 33/5 = 6.6$

SJF :-

P_4	P_1	P_3	P_5	P_2

0 6 7 9 12 16

Avg TAT $= \frac{4+15+5+6+10}{5}$

$= \frac{40}{5} = 8$

Avg WT $= \frac{3+14+3+0+7}{5}$

$= 24/5 = 4.8$

Date - 04.10.23

* SRTF :-

PNo	AT	BT	CT	TAT	WT	RT
P_1	0	~~5~~ ~~4~~ 3	9	9	4	0
P_2	1	~~3~~ ~~2~~ 0	4	3	0	0
P_3	2	~~4~~ ~~6~~ 0	13	11	7	7
P_4	4	~~1~~ 0	5	1	0	0

P_1	P_2	P_2	P_4	P_1	P_3

0 1 2 4 5 9 13

→ It is a preemtive SJF that will preempt the currently executing process while the non-preemtive SJF will allow currently running process to complete CPU time or CPU burst.

Q:

PNo	AT	BT	CT	TAT	WT	RT
P_1	0	~~9~~ 8	13	13	4	0
P_2	1	~~4~~ ~~3~~ 0	5	4	0	1
P_3	2	9	22	20	11	13

P_1	P_2	P_2	P_1	P_3

0 1 2 5 13 22

Avg TAT $= \frac{13+4+20}{3}$

$= 37/3$

Avg WT $= 15/3 = 5$

Avg RT $= 14/3$

H/W - SRTF :- (and Priority)

Priority	PNo	AT	BT	CT		TAT		WT		RT	
				SRTF	Priority	SRTF	Priority	SRTF	Prio-rity	SRTF	Prio-rity
4	P_1	1	5	17	6	6	5	1	0	0	0
2	P_2	2	6	20	15	18	13	12	7	12	7
2	P_3	2	4	14	19	12	17	8	13	8	13
1	P_4	4	1	5	20	1	16	0	15	0	15
3	P_5	6	3	10	9	4	3	1	0	1	0

SRTF:

P_1	P_1	P_1	P_4	P_1	P_1	P_5	P_3	P_2

0 1 2 3 4 5 6 7 10 14 20

Avg TAT = 41/5 , Avg WT = 22/5

Avg RT = 21/5 = 7

Priority :-

P_1	P_5	P_2	P_3	P_4

0 1 6 9 15 19 20

Avg TAT = 54/5 Avg WT = 35/5 = 7

Avg RT = 35/5 = 7

* Priority Scheduling

Assumption - Higher no. → Higher priority

Priority	PNo	AT	BT	CT	TAT	WT	RT
2	P_1	0	~~4~~ 3	15	15	11	0
3	P_2	1	~~3~~ ~~2~~ 0	12	11	8	0
4	P_3	2	~~1~~ 0	3	1	0	0
5	P_4	3	~~5~~ ~~4~~ 0	8	5	0	0
5	P_5	4	~~2~~ 0	10	6	4	4

P_1	P_2	P_3	P_4	P_4	P_5	P_2	P_1

0 1 2 3 4 8 10 12 15

→ In this technique, a priority is associated to the process and CPU is allocated to the process with highest priority. equal priority processes are scheduled in FCFS.

Q.

Priority	PNo	AT	BT	CT	TAT	WT	RT
1	P_1	0	~~4~~ 3	16	16	12	0
4	P_2	1	5 ~~4~~ ~~3~~ ~~2~~ ~~1~~ 0	9	8	3	0
2	P_3	3	1	13	10	9	9
3	P_4	4	3	12	8	5	5
5	P_5	5	~~3~~ ~~2~~ ~~1~~ 0	8	3	0	0

P_1	P_2	P_2	P_2	P_2	P_5	P_5	P_2	P_4	P_3	P_1

0 1 2 3 4 5 6 8 9 12 13 16

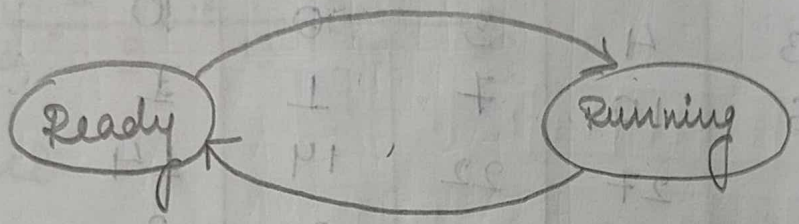
Avg TAT = 45/5 = 9

Avg WT = 29/5

Date - 06/10/23

Round - Robin :-

PID	AT	BT	CT	TAT	WT	RT
P_1	0	~~5~~ 3 1	12	12	7	0
P_2	1	~~4~~ 2	11	10	6	1
P_3	2	~~2~~ 0	6	4	2	2
P_4	4	~~1~~ 0	9	5	4	4



time quantum = 2

Ready

P_1	P_2	P_3	P_1	P_4	P_2	P_1

0

Gantt Chart

P_1	P_2	P_3	P_1	P_4	P_2	P_1

0 2 4 6 8 9 11 12

PID	AT	BT	CT	TAT	WT	RT
P_1	0	~~5~~ ~~3~~ ~~1~~ 0	13	13	8	0
P_2	1	~~3~~ 1	12	11	8	1
P_3	2	~~1~~ 0	5	3	2	2
P_4	3	~~2~~ 0	9	6	4	4
P_5	4	~~3~~ ~~1~~ 0	14	10	7	5

Avg TAT = 43/5
= 8.6

Ready

P_1	P_2	P_3	P_1	P_4	P_5	P_2	P_1	P_5

0

Gantt Chart

P_1	P_2	P_3	P_1	P_4	P_5	P_2	P_1	P_5

0 2 4 5 7 9 11 12 13 14

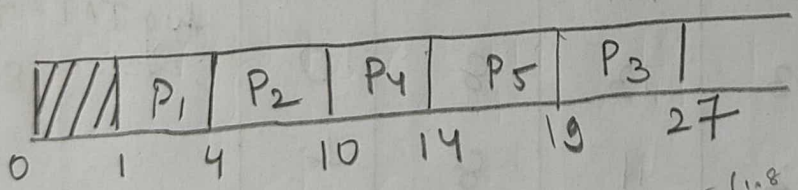
→ A time quantum 'q' is maintained and it is the criteria for choosing the next process. Its mode is preemitive so that after the time quantum the ~~context~~ process will be context switched.

* HRRN :- (Highest Response Ratio Next)

mode - non-pre-emtive

PID	AT	BT	CT	TAT	WT	RT
P_1	1	3	4	3	0	0
P_2	3	6	10	7	1	1
P_3	5	8	27	22	14	14
P_4	7	4	14	7	3	3
P_5	8	5	19	11	6	6

$$\text{Response ratio} = \frac{WT + BT}{BT}$$



t(at 10) $P_3 - \frac{5+8}{8} = 1.6$

$P_4 - \frac{3+4}{4} = 1.7$

$P_5 - \frac{2+5}{5} = \frac{7}{5} = 1.4$

t(at 14)

$P_3 - \frac{9+8}{8} = 2.12$

$P_5 - \frac{6+5}{5} = 2.2$

PID	AT	BT	CT	TAT	WT = RT
P_1	0	3	3	3	
P_2	2	6	9	7	
P_3	4	4	13	9	
P_4	6	5	20	14	
P_5	8	2	15	7	

Avg TAT = 40/5 = 8

P_1	P_2	P_3	P_5	P_4

0 3 9 13 15 20

~~At t=8~~

At t = 9

$P_3 = \frac{5+4}{4} = \frac{9}{4} = 2.25$

$P_4 - \frac{3+5}{5} = \frac{8}{5} = 1.6$

$P_5 - \frac{1+2}{2} = \frac{3}{2} = 1.5$

At t = 13

$P_4 - \frac{7+5}{5} = \frac{12}{5} = 2.4$

$P_5 - \frac{5+2}{2} = \frac{7}{2} = 3.5$

→ HRRN (Highest Response Ratio Next) –

In this algorithm we consider the response ratio as $\frac{WT+BT}{BT}$. Using SJF algorithm we face the problem of starvation, i.e the process with longer time do not get their chance to get executed on the CPU. Hence, in HRNN we consider WT as well as BT. Its mode is non-preemptive in nature

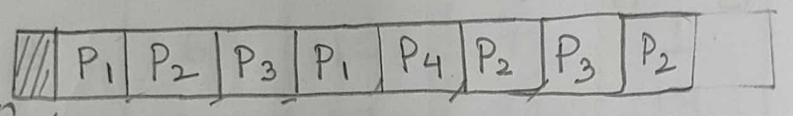
H/W

Q Round Robin (q=3) & HRNN

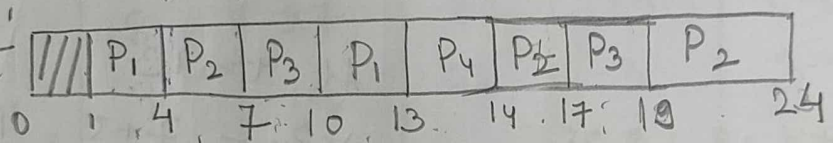
PID	AT	BT	CT	TAT	WT	RT
P_1	1	~~6~~ 30	13	12	6	0
P_2	2	11 ~~8~~ 5	24	22	11	2
P_3	4	~~5~~ 2	19	15	10	3
P_4	7	1	14	7	6	6

Round Robin :-

Ready



Gantt chart

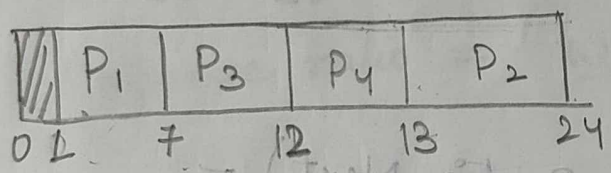


Avg TAT = 14

Avg WT = 8.25

Avg RT = 2.875

HRRN :-



$t = 7$ } $P_2 \Rightarrow \frac{5+11}{11} = \frac{16}{11} = 1.45$

$P_3 \Rightarrow \frac{3+5}{5} = \frac{8}{5} = 1.60$

$P_4 \Rightarrow \frac{0+1}{1} = 1$

$t = 11$ } $P_2 = \frac{9+11}{11} = \frac{20}{11} = 1.8$

$P_4 = \frac{4+1}{1} = 5$

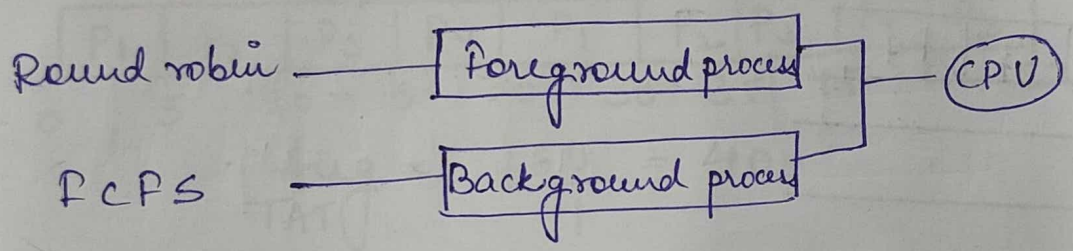
PId	AT	BT	CT	TAT	WT = RT
P_1	1	6	7	6	0
P_2	2	11	24	22	11
P_3	4	5	12	8	3
P_4	7	1	13	6	5

Avg TAT = 10.5 Avg WT = 4.75

* Date - 10.10.23

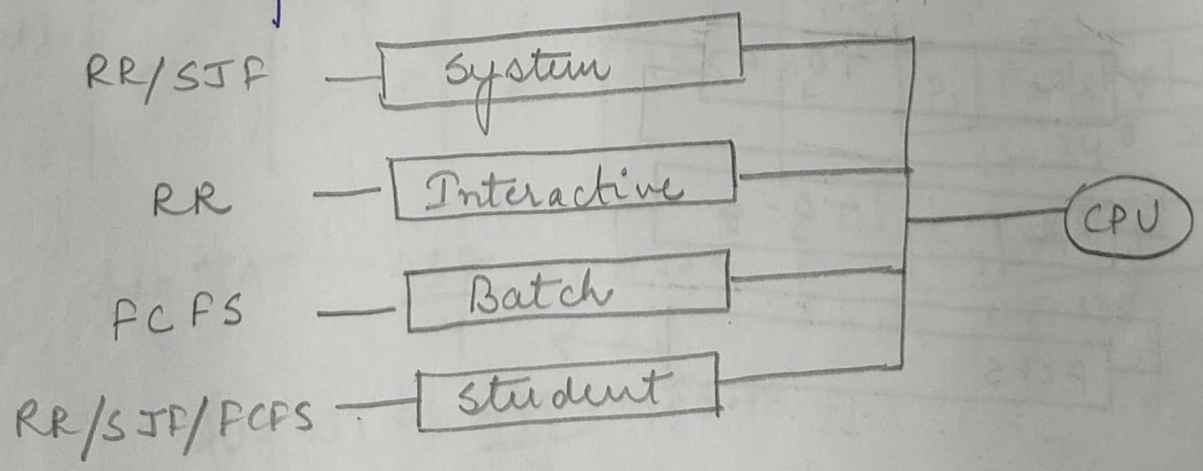
* Multilevel Queue Scheduling :-

1. Foreground — Interactive process
2. Background — Batch process



Priority wise processes -

1. System process
2. Interactive process
3. Batch process
4. Student process



→ Only after the completion of all processes from top level queue, the bottom level ready queue would be executed.

q = 2 time units

P No.	Q No	AT	BT	CT	TAT	WT
P_1	1	0	4	6	6	2
P_2	1	0	3	7	7	4
P_3	2	0	8	20	20	12
P_4	1	10	5	15	5	0

RR

q_1 | P_1 | P_2 | P_1 | P_2 | P_4

FCFS

q_2 | P_3 |

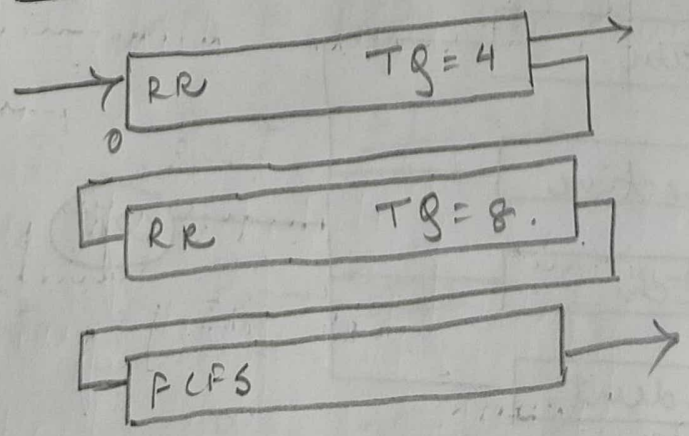
Gantt chart

P_1	P_2	P_1	P_2	P_3	P_4	P_3

0 2 4 6 7 10 15 20

$$\text{Avg TAT} = \frac{38}{4} = 9.5$$

* Multilevel Feedback Queue :-



PNo.	AT	BT	CT	TAT
P_1	0	~~15~~ ~~10~~	30	30
P_2	1	~~10~~ ~~5~~	35	34
P_3	1	~~20~~ ~~15~~ 5	52	51
P_4	2	~~7~~ ~~2~~	47	45

BT = ~~40~~ ~~38~~ ~~31~~ ~~19~~ ~~20~~

TQ = 2

Incremented by 5

TQ = 2 → TQ = 7 → TQ = 12 → TQ = 17 → TQ = 22

5 queues are used.

P_1	P_2	P_3	P_4		RR = 5

P_1	P_2	P_3	P_4		RR = 10

P_3		FCFS

P_1	P_2	P_3	P_4	P_1	P_2	P_3	P_4	P_3

0 5 10 15 20 30 35 45 47 52

$$\text{Avg TAT} = \frac{160}{4} = 40$$

PNO	AT	BT	CT	TAT	WT
P_1	0	~~53~~ ~~36~~ 11	136	136	83
P_2	0	~~17~~ ~~0~~	34	34	17
P_3	0	~~68~~ ~~51~~ 26	162	162	94
P_4	0	~~24~~ ~~7~~	125	125	101

P_1	P_2	P_3	P_4		RR = 17

P_1	P_3	P_4		RR = 25

P_1	P_3		FCFS

P_1	P_2	P_3	P_4	P_1	P_3

17 34 51 68 93 118

P_4	P_1	P_3	

118 125 136 162

$$\text{Avg TAT} = \frac{457}{4} = 114.25$$

$$\text{Avg WT} = \frac{295}{4} = 73.75$$

Date- 11.10.23

* Traditional Unix Scheduling :-

1) Kernel Non - Interruptable Mode
2) Kernel Interruptable Mode
3) User Mode (can only decrease the priority of a user process using nice function)

Priority → (Base Priority of Process) + f (CPU time of process) + $\underbrace{\text{nice}}_{\text{added only in user mode}}$

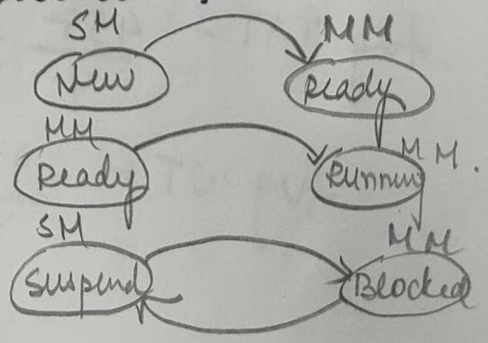
* Priority (for fair scheduling in user mode)

= Base priority of process + f (CPU time of process) + nice + f (CPU time of process group)

→ • The UNIX Scheduler employs Multilevel feedback using Round-Robin within each of the priority queues. Priorities are based on process time and its execution history. The priority of each process is recomputed once per second at which time a new scheduling decision is made.

⊙ Types Of Scheduling & Scheduler :-

1) Long term Scheduling (Scheduler - Job scheduler)
2) Short term Scheduling (Scheduler - CPU scheduler)
3) Medium term Scheduling (Scheduler - Swapper)



→ The aim of processor scheduling is to assign processes to be executed by the processor in a way that meets ~~obje~~ different system objectives such as response time, throughput & processor efficiency.

→ Long term scheduling is performed when a new processes are created. It is determined by long term scheduler which programs are admitted to the system for processing. The scheduler must decide when the processor can take one or more process. It controls degree of multiprogramming.

no. of processes ready queue can accomodate

→ The short term scheduling makes the decision to transfer processes from ready state to running state. The short term scheduler also known as dispatcher executes most frequently & makes the decision which process to next.

→ The swapping in decision b/w suspend or block states is based on managing the degree of multiprogramming. This is the duty of medium term scheduler.

Algo	FCFS	SJF	SRTF	Priority	Roundrobin
Criteria	AT	BT	BT	Priority	Time quantum
Mode	Non preemtive	Non-preemtive	Preemtive	Preemtive	Preemtive
Advantage	① Easy to implement ② Simplest	① Minimizes RT ② Improved WT	① Benefit if we predict next process	① Priority gets chance ② Starvation of low priority process	① Every process gets chances ② No starvation ③ No convoy effect
Disadvantage	① Convoy effect	① Difficult to predict next process ② Starvation	① Complex ② Prediction difficulties		① Not suitable for less TQ ② Overhead of context switching

Date-13.10.23

* Process Control

- Modes of Execution
  1) User Mode
  2) System Mode / Kernel
- Process Creation
  1. Process Id
  2. Allocate some space
  3. PCB Initialization
  4. Setting appropriate links
- Process Switching
- Mode Switching
  1) Interrupt occurs
  2) User call to privileged instructions
  - Interrupts
  - Traps

1. Modes of Execution:-

→ Processor execution normally works in two modes :-

① User mode - The less privileged mode often referred as user mode as user program typically executed in this mode.

② System mode - The more privileged mode is Kernel or system mode. Important and privileged processes/instructions are run in this mode.

There is a bit in PSW that indicates the mode of execution. The bit is changed in response to certain events. When a user makes a call to OS service, and when an interrupt triggers, execution of an OS routine, the mode is set to Kernel mode and upon return from the service to the user process, the mode is set to the user mode.

2. Process Creation :-

1. Assign unique process Id to the process.
2. OS must know how much space is needed for the private user address space & the user stack. These values can be assigned by default based on the type of process, they can set based on user request or by a parent process.
   If a process is spawned by a process, then space is allocated by parent process.
3. All the entries are 0 except PC (Program Counter) and System Stack Pointers. The priority may be set by default to the lowest unless a request is made for higher priority.
4. If the OS maintains each scheduling queue then the new process must be put in new/ready state.

3. Process Switching :-

→ Two kinds of system interrupts are present which may need process switching

1) Clock Interrupt - When the time slice of any process or time quantum expires, this process must be switched to the ready state

2) I/O Interrupts - If the I/O action contains an event for which processes are waiting then after the completion of I/O, the OS moves the processes from block state to the ready state

3) Supervisor Call - OS may be activated by a supervisor call by a program being executed.

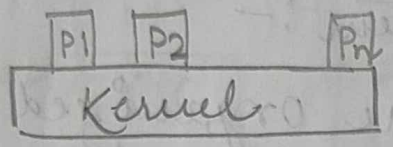
For example -

A user process is running & the instruction requests an I/O operation. This call results in a transform to a routine to the part of OS system part. The use of the system call may place the user process in the block state.

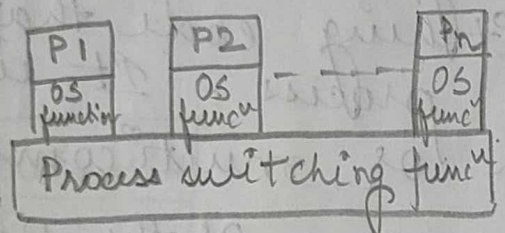
→ With a trap, OS determines if the error or exception we can be handled or not. If it can't be handled, the process is moved to the exit state and if it can be handled, OS invokes some recovery procedures.

* Execution Of OS :-

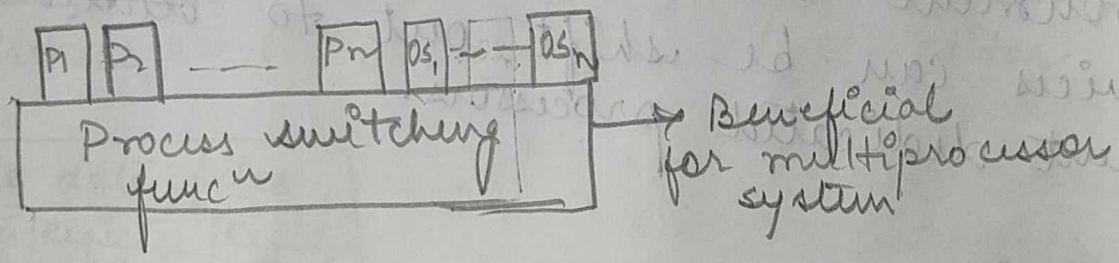
1)



2)



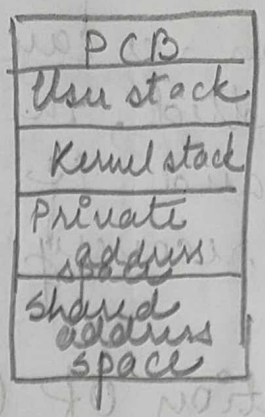
3)



1) Separated Kernel / Non process kernel :-
The kernel is executed outside of any process. The process is considered to apply only to user programs and OS code is executed as a separate entity operating in ~~se~~ privileged mode.

2) Execution within user process :-
All OS softwares are executed in the context of user process. OS is a collection of routines, the user calls to perform different functions. A separate kernel stack

is used to manage calls while the process is in kernel mode.



3) Process based Operating System :-

The major kernel functions are organized as separate processes. There may be a small amount of process switching code that is executed outside of any process. It is useful in multiprocessor or multicomputer environment in which some of the OS services can be shift out to ~~dedicated~~ different dedicated processors.

~~Date~~ - ~~17.10.23~~

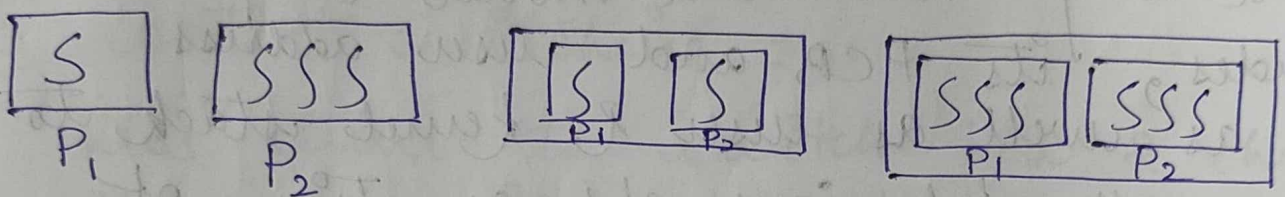
Date - 17.10.23

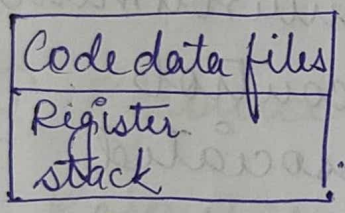
Ch-4 (Threads)

process - program in execution

threads - process is divided into multiple/ single thread

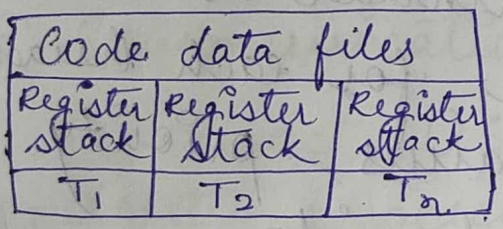
- basic unit of CPU utilization





Process

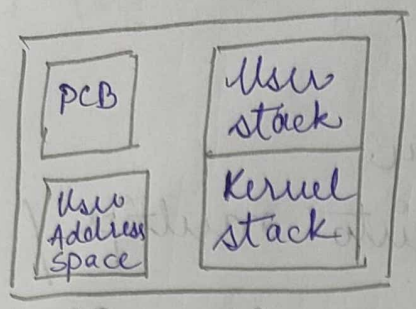
for example (of threads)
- web browser, text editor



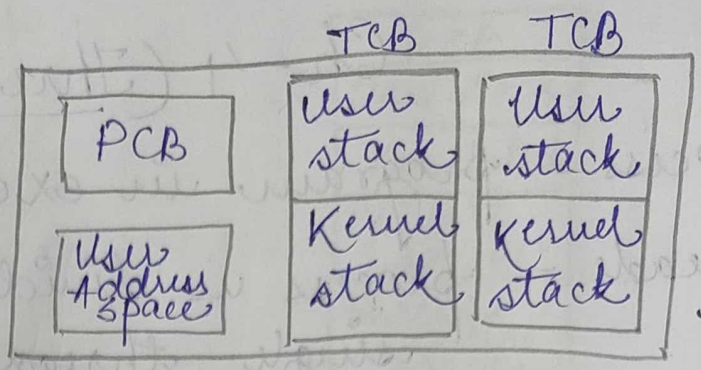
Threads.

→ A thread is the basic unit of CPU utilization. We can have single or multiple threads in a system. Multithreading refers to the ability of an OS to support multiple concurrent paths of execution within a single process. In a multithreaded environment, the thread share code data & files of the process & maintain

their own register & stack.



Single thread



Multi-thread

→ In a single threaded model, the process includes, its PCB and user address space as well as user & kernel stack to manage the behaviour of execution of processes. In a multi-threaded environment there is still a single process control block and the user address space associated with the process, but now there are separate stacks for each thread as well as separate control block for each thread that contains register values, priority & other thread related info.

Advantage Of Threads :-

1. Creation is easy
2. Termination is easy
3. Easy context switching
4. Resource management is better
5. Communication improves

①
It takes less time to create a new thread in an existing process rather than creating a brand new process.

② Termination

Less time to terminate a thread than a process.

③ Context Switching :-
Less time to switch b/w 2 threads than to switch b/w process.

④ Resource Management :-
Resource management is better

⑤ Communication :-
In most OS, communication b/w independent processes, requires the intervention of the kernel to provide protection & the mechanisms needed for communication. As threads in a process share memory & files, they can communicate without invoking the kernel

* States Of a Thread :-

① Ready/Running :-

As like process, the key stage for a thread are running/ready.

② Block :-

When a thread needs to wait for an event, it will block and the processor will run the next ready thread.

③ Unblock -

When the event for which, thread is blocked occurs, the thread is unblocked & send to the ready state.

④ Finish :-

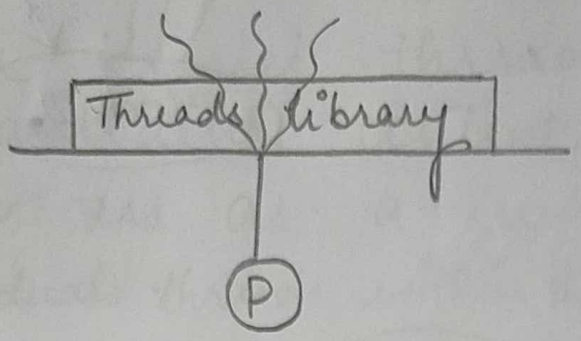
When a thread completes, the stack & registers are re-allocated.

⑤ Spawning :-

A thread within a process may spawn another thread, providing an instruction pointer & arguments for the new thread.

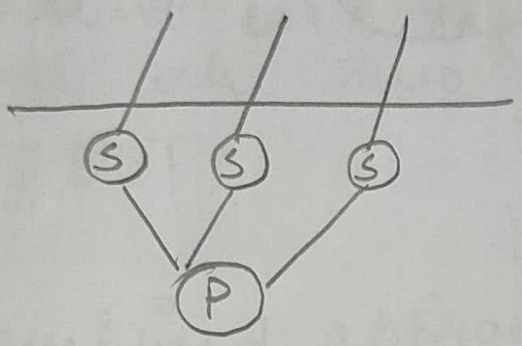
* Types Of Thread :-

① User level thread [created by user, not recognized by OS
② Kernel level thread [created by OS, recognized by OS



User space

Kernel space

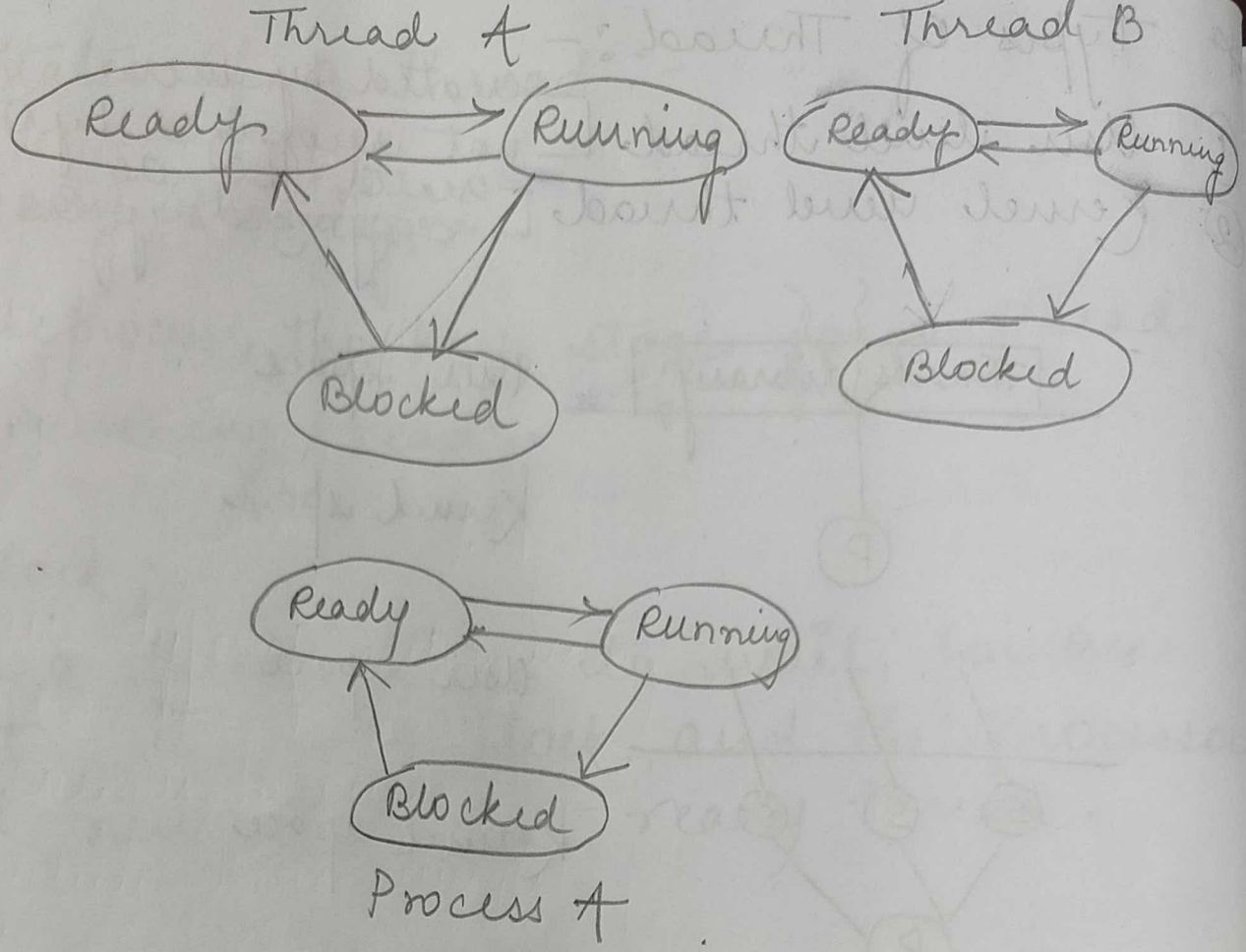


User space

Kernel space

→ In user level thread, all the thread management work is done by the application & kernel is not aware of the existence of threads. Any application can be programmed to be multithreaded by using a thread library which is package of routines for ULT management.

The thread library contains the code for creating & destroying threads, for passing messages & data b/w threads, for scheduling thread execution & by saving & restoring thread context.



⊙ Advantages of ULT :-

① Scheduling can be application specific.

② Thread switching does not require kernel mode priviledges because all of the thread management are within the user space of a single process

③ ULT can be run on any OS, no changes are to be made to the kernel

⊙ Disadvantages :-

① When system calls are blocking in a typical OS, so blocking a process could block all its threads.

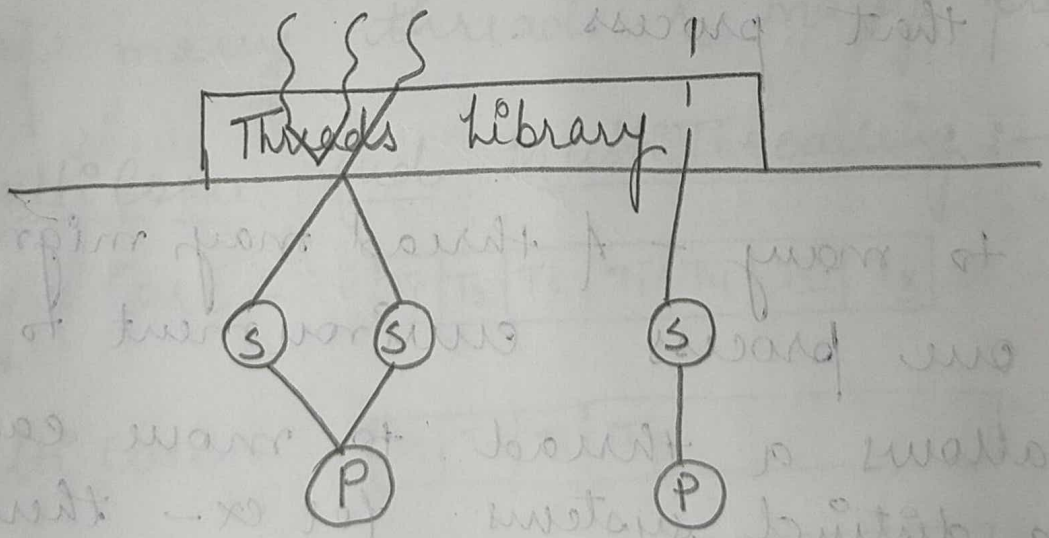
② In a multithreaded application, it can't take advantage of multiprocessing as a kernel assigns one process to the processor at a time.

* Kernel Level Thread :-

→ There is no thread management code in the application level, simply an API to the kernel thread facility. The kernel maintains context information for the process as a whole and for the individuals threads within the process and scheduling is done on the thread basis. This over-comes the two 2 drawbacks faced by the ULT.

* Combined approach -

The multiple ULTs from a single application can be mapped onto some level of KLTs



Date - 18.10.23

⓪ Relationship b/w threads & processes :-

① 1:1
② M:1
③ M:N
④ 1:M — One thread / Many process

→ Four types of relationship holds for the process :-

(i) One is to one → each thread of execution is a unique process with its own address space & resources

(ii) Many to one → A process defines an address space & dynamic resource ownership. Multiple threads can be created & executed within that process.

(iii) One to many - A thread may migrate from one process environment to other. This allows a thread to move easily among distinct systems. for ex - there is a main program that makes use of an I/O subprogram. The main program can spawn a new process to handle the I/O. So, there are 3 ways to implement

this :-

(i) The entire program can be implemented as a single process but we have memory constraint here.

(ii) The main program & I/O subprogram can be implemented as two sub-processes. But, there is a overhead of creating a sub-ordinate process & context switching.

(iii) So, the solution for these problem is, we treat the main program and I/O subprogram as a single activity & implement it as a single thread. One address space can be created for main program and one for the I/O subprogram, so, the threads can be moved b/w these 2 address spaces as execution proceeds.

(iv) Many to many - It combines attributes of many threads & many processes.

* Multi Core and Multi Threading :-

Single Core - | T_1 | T_2 | T_3 | T_1 | T_4 | T_2 | T_3 |

Multi Core - | T_1 | T_3 | T_1 | T_3 |

| T_2 | T_4 | T_2 | T_4 |

→ Multicore & multithreaded programming provides a mechanism for more efficient use of multiple cores. On a single core system, the execution of threads will be interleaved over time. On a system with multiple cores, concurrency means the threads can run in parallel. However, this posses few challenges:-

(i) Dividing activities:-

The application must be examined to find areas that can be divided into seperate concurrent task.

(ii) Balance:-

It must ensure the task perform equal work of equal value.

(iii) Data Splitting:-

As the applications are divided into tasks, the data accessed & manipulated must be divided to run on seperate cores.

(iv) Data Dependency

The execution of task must be synchronized to accomodate data dependency.

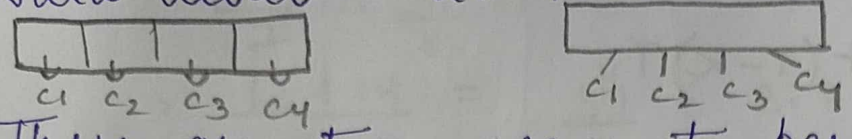
(v) Testing, Debugging & Concurrency.

When a program is running parallely on

multiple cores there are different execution paths. Testing & Debugging such concurrent program is difficult.

* Data And Task level Parallelism:-

1. Data level 2. Task level



→ There are two ways to parallelise the work load of a process:-

(1) Data Level Parallelism / Fine-grained Threading

It focuses on distributing subsets of same data accross multiple computing cores & performing the same operation on each core. It is also known as Fine-grained Threading.

(2) Task Level Parallelism

It involves distributing not data but tasks accross multiple computing cores. For example could be different threads performing different operations on a single array of elements. It is also known as Coarse-grained Threading.

⊙ SJF

To predict time:-

① Simple Averaging

	P_1	P_2	P_3	P_4
BT	1	4	3	2

$P_5 = 10/4$

② Exponential Averaging

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n$$

τ_{n+1} - predicted time of next process

α - scaling factor

t_n - original BT

$T_1 = 10$

$\alpha = 0.5$

P_1	P_2	P_3	P_4
6	4	6	4

$T(5) = ?$

$T(5) = \alpha t_4 + (1-\alpha) T_4$

$T_2 = \alpha t_1 + (1-\alpha) T_1$

$= \frac{1}{2} \times 6 + \frac{1}{2} \times 10$

$= 8$

$T_3 = \alpha t_2 + (1-\alpha) T_2$

$= \frac{1}{2} \times 4 + \frac{1}{2} \times 8$

$= 2 + 4$

$= 6$

$T_4 = \alpha t_3 + (1-\alpha) T_3 = \frac{1}{2} \times 6 + \frac{1}{2} \times 6$

$= 6$

$T_5 = \frac{1}{2} \times 4 + \frac{1}{2} \times 6 = 5$

Q

P_1	P_2	P_3	P_4
5	2	6	3

$T(5) = ?$ $\alpha = 1/2$ $T(1) = 5$

$T(2) = 1/2 \times 5 + 1/2 \times 5$

$= \frac{5}{2}$

$T(3) = 1/2 \times 2 + 1/2 \times 5$

$= 1 + \frac{5}{2} = 7/2$

$$T(4) = 1/2 \times 6 + 1/2 \times 7/2$$
$$= 3 + \frac{7}{4} = 19/4$$

$$T(5) = 1/2 \times 3 + \frac{1}{2} \times \frac{19}{4}$$
$$= \frac{3}{2} + \frac{19}{8}$$
$$= \frac{12+19}{8} = \frac{31}{8} = 3.875$$

Date - 25.10.23

⊙ Process Synchronization :-

① Serial - one after the another

② Parallely - running at the same time

↓ shareable

memory, resources, variables, code.

a = 10

{

1. R(a)
2. a - 1
3. W(a)

}

	Serial		Parallel	
	P_1	P_2	P_1	P_2
R(a)	9			9
a - 1		8	9	

B = 2

$P_1()$ $P_2()$

{ {

1) C = B - 1 3) D = 2 * B

2) B = 2 * C 4) B = D - 1

} }

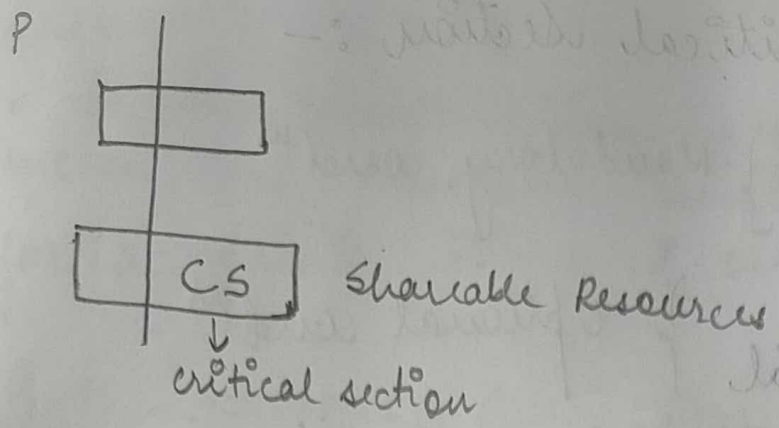
Case 1 1234	3	Case 3 1342	2	Case 5 1324	3
Case 2 3412	4	Case 4 13142	2	Case 6 3124	3

shared = 5

P_1	P_2
{	{
int shared x;	int shared x;
x++;	x++;
sleep();	sleep();
W(x);	W(x);
}	}
≈ 6	≈ 6

* Race Condition :-

→ race cond^n occurs when multiple processes or threads read and write data items so that the final result depends on the order of execution of instructions in the multiple processes.



* Critical Section :-
Shared resources are accessed by different processes at a time, the space where or the region where the processes share this shareable resource, that region is known as critical section.

* Concerns of an OS -
1. The OS should keep track of different processes which is done with the help of PCB.
2. The OS must allocate & de-allocate various resources for each active process. Resources can be memory, IO devices, files & processor time.
3. The OS must protect the data & physical resources of each process against unintended interferences by other processes.
4. The functioning of a process must be independent of the speed its execution is carried out relative to the speed of another concurrent process.

* Process Interaction :-
1. Process unaware of each other. → competition
2. Process indirectly aware of each other → co-operation by sharing
3. Process directly aware of each other → co-operation by communication

Deadlock is possible in all

* Criteria to access Critical Section :-

① Mutual Exclusion } Mandatory condns
② Progress
③ Bounded Wait. } Optional condns.
④ Architecture Neutral

The criteria to access critical section is -

① Mutual Exclusion :-

- only one process at a time is allowed in critical section,
- Among all processes that have critical sections for the same resource / shared object
- A process that halts in its non-critical section must do so without interfering with other processes.
- It must not be possible for a process requiring access to critical section to be delayed indefinitely.
- When no process in critical section, any process that requests entry to its critical section must be permitted to enter without delay.

② Progress :-

- only those processes should be allowed to enter those who willingly wants to enter critical section
- No assumptions are to be made about relative process, speeds, no. of processes.

③ Bounded Wait:-

- It is the maxm time a process can wait. A process can remain inside the critical section for a finite time & only.

* Date - 27.10.23 (Hardware Solutions)

P_0

```
main()
{
    x, y
    [Non-critical section]
    entry code
    [Critical section]
    count ++
    exit code
}
```

P_1

```
main()
{
    a, b
    [Non-critical section]
    [Critical section]
    count ++
}
```

reserves critical section for a program

frees the critical section

→ *) Independent variables need not be added in the critical section. If any program wants to enter critical section it will not access directly. It will execute some entry section code. At one time only one process can execute, entry code & enter critical section. Other processes will stop at the entry indicating mutual exclusion.

① Lock Variable —

lock = 0 X 0 P_0 P_1

lock != 0

```
{
1. while (lock == 1); ] entry
2. lock = 1
3. CS
```

} fails mutual exclusion

4. lock = 0 ; exit

}

⊙ Turn Variable -

P_0

```
{
while (turn != 0);
CS
turn = 1
}
```

P_1

```
{
while (turn != 1);
CS
turn = 0;
}
```

turn = 0 / 1

→ Mutual exclusion holds but progress doesn't ho

→ This solution guarantees mutual exclusion bu has few drawbacks :-

1. It does not follow progress as processes strictl alternate in their use of critical section.
2. The choice of process has been considered whether it wants to enter critical section or not. Therefore P_0 can't enter even though the critical section is empty.

⊙ Flag Variables :-

P_0

```
{ flag(0) = T
while( flag(1) = T);
CS
flag(0) = F
}
```

P_1

```
{
flag(1) = T
while( flag(0) = T);
CS
flag(1) = F
}
```

	0	1
flag	F	F

→ Mutual exclusion
→ Progress ✗

Lab Section

Date - 11.09.23

* Commands in Unix (CLI) -

1. who
2. who am i
3. date
4. cal
5. pwd - print working directory
6. man
7. mkdir
8. cd → cd / (parent directory)
   cd ~
   cd .. (previous directory)
9. rmdir → -help
   - v
   - version - output version info & exit
   - ignore - fail - on - non-empty
10. cat (view a single file) <filename>
    rm - -v - r
11. mv
12. touch (creates a file in ~~an~~ directory)
13. cp
14. ls
15. rm
16. cat
17. wc
18. sort
19. head (prints the 1st 10 lines of a file)
20. tail (last 10 lines)

rmdir - - version :-
output version information & exit

rmdir - v :-
output a diagnostic for every directory processed

rmdir - - ignore - fail - on - non - empty
ignore each failure that is solely because a directory is non-empty

ls R ls -v

↓ displays the list of files in a directory

↓ displays the list of files in a directory natural sort (version) of numbers

Date - 18.09.23

Directory Access Mode

→ read (4)
→ write (2)
→ execute (1)

} Permission values

1) Owner 2) Group 3) Others

Changing permission :- chmod

chmod [ugoa][+, -, =][r w x]

-R (recursive)

chown → change ownership → chown user2 file

chgrp → change group.

-c num → Prints the last 'num' bytes from the file specified

cmp → compares 2 files.

diff → outputs the difference b/w two files

a: add
c: change
d: delete

num1 a num2
↓ ↓
line no. of 1st file — line no. of 2nd file

-e → \b (remove spaces)
\c (suppress trailing new line)
\n (creates a new line)
\t (creates horizontal tab spaces)
\r (carriage return with backspace interpreter)
\v (vertical tab spaces)

* File Operations :-

find (for searching purposes) → only for files & directories

grep (searches a file with specific pattern)
↓
not just files & directories but also content inside it.

1) ls, chmod, tail, cmp, echo, history, find, grep.

Date - 09/10/23.

* Types of Shells
  - Borne shell
  - C shell
  - Korn shell
  - Bash Shell.

.bash , .sh → extension for excutable scripts.

#!/bin/sh

→ comment

" " → string /input

' ' → output (for variable reference)

` ` → command.

$ → variable

Variables < System (in capital letters) / User defined

expr → for mathematical computation.

continued theory ...

→ This solution guarantees mutual exclusion but creates another problem. If both processes set their flag to True before either has executed the while statement then each process will think that the other has entered CS causing deadlock

⊙ Using both variable - (Peterson's Solution)
(2 process solⁿ)

P_0

```
{
flag(0) = T
turn = 1
while (turn ==1 && flag(1) = T);
CS
flag(0) = F
}
```

P_1

```
{
flag(1) = T
turn = 0
while (turn ==0
&& flag(0) == T);
CS
flag(1) = F
}
```

→ Mutual exclusion ✓
→ Progress ✓

turn = 0 1

flag | F T F |

→ The mutual exclusion is preserved in this solution. This solves the problem of progress as —

(i) P_1 has no interest in the CS as it will set flag(1) = F

(ii) P_1 can use CS repeatedly, this case is not possible as P_1 gives opportunity to P_0 by setting the turn to 0 before entering CS.

Hence we can say Peterson's solution satisfies our mandatory criteria i.e mutual exclusion & progress.

Date - 31.10.23 (Hardware Solutions)

* 1. Test & set :-

```
{
  1) while (lock ==1); ] entry
  2) lock = 1;
     CS
     lock = 0; ] exit
}
```

```
1) while (test & set (& lock));
2)       CS
3)       lock = false;
```

```
boolean test & set (boolean * target)
{
    boolean r = * target;
    * target = true;
    return r;
}
```

Among hardware solutions one of the way to ensure mutual exclusion is disabling interrupts. But this solution is not feasible in the multiprocessor environment as disabling interrupts can be time consuming since the message are passed to all the processes

Many moder computer system therefore provide many special hardware instructions that allow us either to test & modify the content of a word or to swap the contents of two words atomically.

→ In this solution, we combine line 1 & 2 from lock variable solu^n so that no pre-emption can occur and mutual exclusion is ensured. This solution satisfies mutual exclusion & progress but fails on bounded wait.

2. Compare and Swap:-

```
{
while(compare & swap(&lock,0,1) != 0);
        CS;
        lock = 0;
}
int compare & swap (int *value, int expected, int
                    new_value)

{   int temp = *value;
    if (temp == 0) — expected
    *value = 1; → new_value
    return temp;
}
```

ME ✓
Progress ✓
BW ✗

→ This solution satisfies mutual exclusion property as we take a shared variable that is initialized to 0. The only process that can enter in the critical section is the one that finds lock = 0. All other processes attempting to enter the CS goes into a busy waiting mode. When a process leaves its critical section, it resets lock to 0 and at this point one of the waiting processes is granted access to CS. This process also satisfies progress but fails on bounded wait.

* 3. Exchange :-

```
do {
    key = true;
    while (key == true) {
        swap (&lock, &key);
    }
    CS
    lock = false;
}

swap (boolean *a, boolean *b)
{
    temp = *a
    *a = *b
    *b = temp;
}
```

loop

ME ✓
Progress ✓
BW ✗

→ In this solution, mutual exclusion is satisfied as a shared variable lock is initialized with 0. It excludes all the other processes from CS by setting lock to 1. Here, we have use the swap function which exchanges the value of key & lock when the process is in CS.

Progress is also satisfied but this solution also fails on bounded wait

* Demerits of h/w solns –

1. Busy waiting is employed. While a process is waiting for an access to CS, it continues to consume processor time

2. Starvation is possible. When a process leaves a CS and more than one processes are waiting. The second process could indefinitely denied access

3. Deadlock – Suppose P_1 executes some special instructions and enters the CS. P_1 is then interrupted by P_2 which is a higher priority process. If P_2 now attempts to enter CS, it will denied access because of mutual exclusion.

Date - 1.11.23.

Semaphores

↓

integer variable

counting | binary

- It is an integer variable which is used in mutual exclusion manner by various concurrent co-operative process to achieve synchronization.
- Only 3 operations may be performed on a semaphore all of which are atomic :-

1. Initialize
2. Decrement
3. Increment.

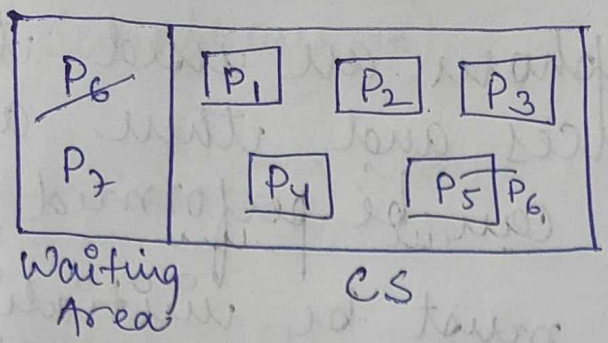
- The decrement operation may result in blocking of a process & the increment operation may result in unblocking of the process.

Its advantages are :-

1. It solves the CS problem
2. Manages various resources and,
3. can decide the order of execution of different processes.

* It is of 2 types -

1. Counting Semaphores ($-\infty$ to ∞)
2. Binary Semaphores (0 or 1)



Lock = ~~1~~ ~~1~~ ~~1~~ ~~1~~ 0

* Operations :-
  P(), Down(), Wait()
  V(), Up(), Signal()

* Counting Semaphore :-

* Wait (semaphore)

```
{
    S.value = S.value - 1;
    if (S.value < 0)
    {
        Put process PCB in suspend list (sleep);
    }
    else {
    return;
    }
```

* Signal (semaphore)

```
{
    S.value = S.value + 1;
    if( S.value ≤ 0)
    {
    select a process from suspend list
    wake();
    }
    else {
    return;
    }
```

→ Counting semaphores are used to manage different resources and there are 3 operations that can be performed, i.e

1) A semaphore must be initialized to a non-ve integer value. If semaphore has -ve value that means those no. of processes are currently blocked.

2) The semaphore wait operation decrements the semaphore value. The process continues it execution when value of semaphore is +ve.

3) The signal operation increments the semaphore value if the resulting value is less than or equal to 0. Then a process blocked by a wait operation is unblocked.

eg – S = -4 (4 process in waiting area)
S = 0 (CS is full)
S = 10 (10 resources can enter CS)
S = 10, 6 wait
4 signal } S = 10 - 6 + 4
= 8

S = 17, 5P
3V
1P
S = 14

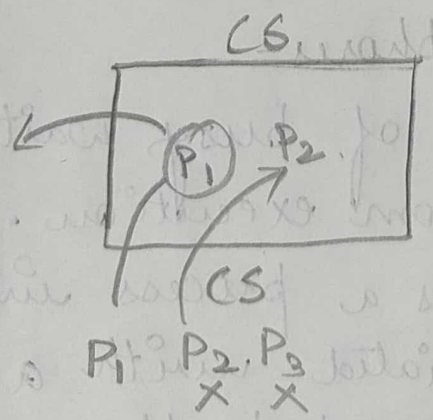
* Binary Semaphore :-

* Wait (Semaphore s)

```
{
  if (s.value == 1)
  {
    s.value = 0;
  }
  else {
    Block and put in suspend list, sleep();
  }
}
```

Signal (Semaphore s)

```
{
  if (suspend list is empty) {
    s.value = 1
  }
  else {
    select a process from suspend list, wakeup();
  }
}
```



When a process is in CS, s.value = 0

→ A binary semaphore can be initialized with 0 or 1, the wait operation checks the semaphore value, if the value is 0, then the process executing the wait is blocked. If the value is 1, then the value is changed to 0 and process continues execution.

The signal operation checks to see if any processes are blocked in this semaphore. If so, then a process blocked by wait operation is unblocked. If no processes are blocked, then the value of semaphore is set to 1.

eg - $P_2 \rightarrow P_1 \rightarrow P_3$ (Order of execution)

⇓

Applcn of binary semaphores

wait(s_1)		wait(s_2)
P_1	P_2	P_3
signal (s_2)	signal(s_1)	

* Advantages Of Semaphores-

(i) It solves the problem of busy wait as the processes are blocked from execution. The block operation places a process into a waiting queue associated with a semaphore & a set of process is sent to the waiting state.

→ A binary semaphore can be initialized with 0 or 1, the wait operation checks the semaphore value, if the value is 0, then the process executing the wait is blocked. If the value is 1, then the value is changed to 0 and process continues execution.

The signal operation checks to see if any processes are blocked in this semaphore. If so, then a process blocked by wait operation is unblocked. If no processes are blocked, then the value of semaphore is set to 1.

eg - $P_2 \rightarrow P_1 \rightarrow P_3$ (Order of execution)

⇓

Applcn of binary semaphores

wait(s_1)		wait(s_2)
P_1	P_2	P_3
signal (s_2)	signal(s_1)	

* Advantages Of Semaphores-

(i) It solves the problem of busy wait as the processes are blocked from execution. The block operation places a process into a waiting queue associated with a semaphore & a set of process is sent to the waiting state.