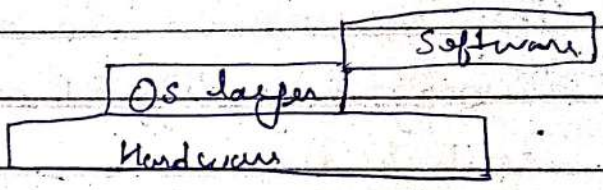
Application Program:- (AP)

1) they are in high level language.

The processor in a computer may not understand an ~ in their higher level lang. forms. So it has to be translated into an equivalent machine lang. prog. so that processor can executed, known as a binary image.

Operating system

It is a software entity which controls our op's of all the hardware sources and provide a users a virtual software machine.



It is a well organised collection of special programs providing an execution environment.

The user utilize a computer to solve their problems by executing Application programs but the processor will not do only Application programs but also the operating system. And when the processor executes the OS programs we say the OS has taken control of the system. OS has to allocate resources to application when they are needed & then reclaim when not needed.

It should resolve access conflict bet^n competing users & enable resource sharing. When OS is doing all these it has to provide a set of services to the application and actual hardware will implement those services. Therefore each OS defines a set of service access points. And application will connect to the appropriate access points to obtain the desired service from the OS.

Whenever OS makes a call to service access point it is known as System Call or monitor call or supervisory call or OS call.

They are similar to ordinary function invocation. OS may have different mechanisms to make the system calls & those calls may have different parameters, therefore method of passing parameters to different calls

Therefore the use of system call will be difficult to application developer. Therefore to make application development easier & also portability (system call is different OS), due to these two issues the IEEE suggested a standard known as POSIX standard, which defines a set of API (app. prog. Interf These APIs will help application developer to write system independent application.

It is the system developers who will implement POSIX APIs & make it available of App developer in form of libraries.

So - system library defines a set of standard function through which app. interact with OS.

The most important part in a OS is known as Kernel (collection of prog., subset of OS).

It is the minimal OS program which is loaded in main memory during Boot-strapping and it will reside permanently in main memory until system is shut down. Generally it resides in lower region of main memory map.

I/O device opr's:-

The processor will use the I/O devices. To the external world some devices store transient data (not storable / changing) & some store storable data

↳ eg:- data to printers
data in form of packets to NIC

These I/O devices are connected to the host computers by using various peripheral or I/O controllers. Controllers know how to operate the device & each I/O controller implements a small set of instruction. known as device registers or operating registers. This registers are sole interface to controller. And the processor architecture will implement some special machine instructions called I/O instructions to read & write register of the devices.

These are basically divided into four categories:-

Command
Status
Input
Output

The I/O controller carry out I/O opr on behalf of the processor. The processor will direct the I/O by writing on I/O by command register & input data to input register. After receiving a command from processor controller will take over & it will do it in its own way & own speed. When this command execution is complete the controller will convey this by writing to output & status register.

One processor has given command for execution, it is to be confirmed by two two options:-

- Programmed I/O — one CPU has given command it will check for completion in a programmed manner
- Interrupt I/O
  - I/O controller interrupt the CPU after completion.

Process Abstraction

P - program
U - user
O - one
m - multiple

We know that the processor will execute on the behalf of users Condition - OPOU, MPOU, MPMU)

Two ways of CPU:-

sequentially

concurrently

When concurrency is there, the processor will execute the parts of the programs of different programs. The problem is the processor executing the instruction does not know which program's instruction it is executing. It does not differentiate btn instruction of different program. It is OS which will decide which instruction of which program will get executed. So the OS with may interleave the execution of all the programs but the need rises to supervise various concurrent programs. So therefore the each program will build on execution context. This program execution behave correctly under execution context. The OS tracks down the program execution in extraction of process & the process is nothing but an entity or object the representation of single computation & also models the execution context of the model.

OS is not a process.

Process & program execution are one & one corresponds. OS starts program execution by creating a process & after completion destroy it. This process is allocated part of main memory to hold data. OS builds up the initial execution context of the process in main memory.

PROCESS ADDRESS SPACE (~)

Application process will execute instruction from their own application programs as well as some instructions related to OS. If an user process either accidentally or intentionally corrupt the program or data of the OS then our system could be in inconsistent state. So it is duty of OS & underlying hardware to avoid these cases. Whereas in multiprocessing environment more than one process either of same or different program may be there in the system. So here additional job of OS would be that one process should not be able to modify data of other process. This task of protection is achieved by implementing ~. This consist of many diff. entities & each entity has distinct individual address (name or number which can uniquely identify the entities in the system). It should be noted that an entity in address space can be accessed only by using address relative to address space, which is known as logical or private address space of the process.

Silly the kernel processes also constitute an address space known as kernel address space.

The collective address space of all the process of an application is known as application space. Kernel it should be noted that all application process will share common kernel space. Whenever the OS will notice some violation it will terminate the corresponding process. Generally we talk that every process should be in its own kernel space but during a system call a process can access both kernel & address space.

The switching from private address space to kernel space is known as address space switching.

System Calls, Interrupt Action, Exceptions

System calls as we known are ordinary function calls implemented in special ways. Process executes a special m/c instruction is used specifically to make a system call & immediate two action take place -

i) instruction will flip the processor from user operating mode to kernel operating mode.

ii) current prog. execution flow changes.

Processor { Kernel mode — can execute special instructions
User mode — ✗

The processor in some process environment and execute instruction from an OS would be called system call handler.

Basically we have migrated from user space to Kernel space for the duration of system call, when this system call handler ~~is~~ is complete kernel will execute another special instruction & control will again go back to process which made the system call.

There are two other situation when processor is flip from user to kernel mode -

- interrupts
- exceptions.

Interrupt can come to processor at any time hence called asynchronous event. Processor will handle interrupt only after executing current instruction and interrupt service program is executed in context of interrupted process.

> Exceptions are synchronous events because even the program will raise exception only when the operation is completed.

Exceptions can be due to illegal expression, divide by 0, arithmetic overflow, addressing violation.

⇒ For each of these three type of service the processor will execute a sequence of inst. from OS programs. This sequence of instruction execution is called kernel execution or kernel path.

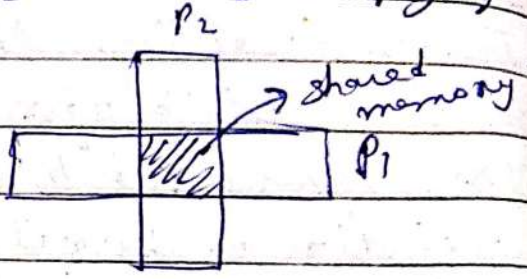
(IPC)

Inter process communication & process synchronization

In modern system the processes work together to complete the application task, these processes are called operating process, they are loosely coupled (each has own address space, own speed, the execution speed is not known in advance). They just interact with each other from time to time by exchanging info^{n}. So we can say that change of info^{n} among process is called inter-process communication. One process cannot violate address space of each other so they will need the assistance of OS to setup communication channel bet^{n} them & for each mechanism there will be some communication primitives (interface op^{n}s have to supported by OS).

That means all comm^n will take place via kernel space, every send or receive op^r is performed by making a system call. So OS provides diff^n mechanisms like -

- the process can communicate without copying data via kernel space.



In shared memory region is used when OS will map part of process address space to the same physical mem. location so all co-operating process has access to the locations. Whereas in indirect schemes the process will make system calls for easier manipulation the infor^n is structured into shared data or shared variables, they are units of infor^n referenced by OS & each shared variable has a unique name/address & ~~time~~ type.
The process will communicated among themselves by manipulation values of shared variables by op^r supported by these variables. Each process executes its own op^r on these shared variable, which are specified by its own program. More than one process is trying to alter value of shared variable concurrently.

⇒ Inter-process communication & synchronization

This may have -ve effect like consistency of value of these variables may be compromised.

So the coordination is required for sharing the variable & this coordination is known as synchronization. OS will implement diffs. synchronization schemes for different purposes.

Protection & Security:-

⇒ The program and Data
↳ stored in HDD

The private info" belonging to users like program or data resides on device like HDD so OS should provide protection from malicious program ~~execution~~ & unauthorised ~~used~~ users are allowed unhindered ~~access~~ access. We know that in a system ~~every~~ most of the user might be authorised to have access to subset of resources.

⇒ Files

The OS store program & data in abstraction of files & some user will own a file & owner expects that access to the file is to only authorised user so OS must ensure privacy of contents & further

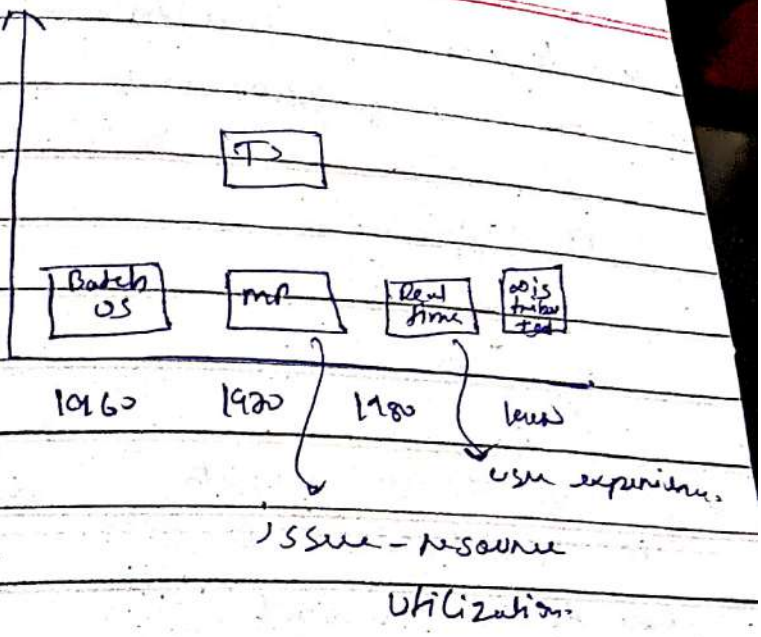
Moreover the owner may want that the different set of user should have different rights the ways in which they can use the file.

» In today's world the computers are generally network. So it a responsibility of OS that in addition to internal security whatever information they are exchanging over the net also should be safeguarded?

i) Instruction Execution
ii) A processor executes a program
iii) A processor executes a process
iv) OS or kernel executes a program
v) " " " " " process
vi) A process executes an instruction
vii) " " " a program
viii) " " " the OS or kernel
ix) A user executes an application program

mp - multiprogramming
TS - time sharing

Classification of OS :-



Multiprocessor system

More than one processor is there & instructions can be executed simultaneously. They will share system bus, system clock and main memory & even devices. Such systems are called tightly-coupled system.

Therefore a multiprocessor system can execute
-- at the real time

⇒ simultaneously - actually >1 process are executed
⇒ concurrently - gives an illusion of simultaneous process but not really

~~Note~~ :

*) degree of simultaneous process execution depends on no. of processors.

Multiprocessor

Symmetric

each processor executes some copy of resident OS, take its own decisions, cooperate with others

Asymmetric

Each processor is assigned a specific task.
One processor is master & other subordinate - Master will distribute work.

The kernel path synchronization is a complex issue, kernel should be highly concurrent. The synchronization techniques designed for uni-processor systems may not work here.

> balancing of work load

> fault tolerance :- (graceful degradation)

If one processor fails other keep on functioning to complete other works.

Multi-User:- Can be used by more than one user.

Each user can interact the application with concerning what the user are doing. They need to competes for resources.

Security was major issue.

Mondum some accounting measures are required

Multi-programmed System

More than one program in main-memory

Multi-process (or multi-tasking)

which executes many processes concurrently.

Objective:- best utilization of resources.

improving the performance

Os has to do the switching among processes

- task switching
- process "
- context "

Time-sharing system

An interactive system the users directly interact to submit execution requests & expects results back. Such a system should able to support both multi-programming & multi processing.

Basically we employ rapid time division multiplexing of processor time.

So the addition responsibility of OS is to provide health to user for effective interaction of system.

Preemptive System :-

Re-entrant Kernels

A reentrant program is one which does not modify itself or any global data.

- A reentrant kernel is one which many program can execute some kernel without affecting others.

In non ~ a process doesn't modify kernel program but can modify global kernel data. So when OS is executing no other process should be allowed to execute the program eg: interrupt or exception.

Generally an OS is composed of both re-entrant & non-re-entrant function.

Monolithic kernels & micro kernels

↓

Operating system
- all functionalities added in
- bulky

↓

we provide minm basic functions others have libraries.
- OS not bulky.

When we design new OS

→ designer need to have clear idea about aim & goals.

→ then goals should be realistic in accordance to available hardware & development tools available.

→ designer should be clear about working of diff hardware resources

- they should have clear policies in the mind.
- They should think of developing using concepts of subsystems. (first simpler then complex designs).

CHAPTER-3

Views of OS

Programmers View

They simply Acts as an extension of basic machine instruction set.

Logical View

It describes the internal function of organization of the system. i.e, breaks up the system into subsystem & specifies a manner in which these subsystem will interact with each other.

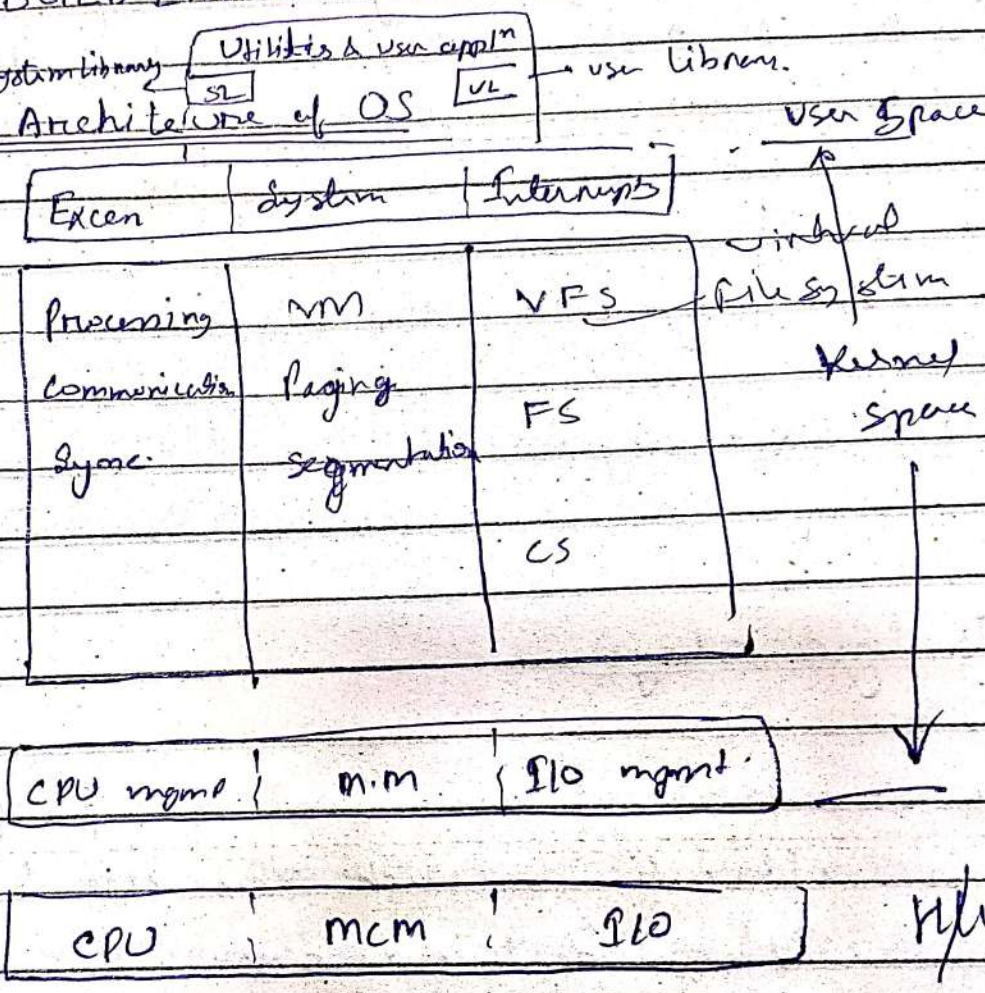
Physical View

It describes the physical structure of the OS programs, data then also provides a map of the logical software entities into their physical representation (i.e dealing with all data structure & physical orgn of data).

Development View (build up environment).

This view will describe the orgn of source code in a development environment. It will also help with maintaing various versions of source code base also called BUILD Environment.

Layered Architecture of OS



Then a thin horizontal layer in bottom in dotted lines is divided into - CPU, memory device & IO devices.

In middle layer it is further divided into - CPU mgmt, memory mgmt, various IO subsystems & device drivers.

In the CPU mgmt, the software in the manager will multiplex the CPU among processes & when it receives back a CPU it decides based on scheduling parameter to which process it will be allocated

Similarly the memory manager will keep track of where OS, resides & keep account of allocation & specific space to specific processes & decides all allocation & deallocation policies

- static allocation
- dynamic allocation

→ I/O device management :-

Middle layer subsystem

A process is treated as a unit for purpose of resource allocation & work scheduling & each process will need certain resources to accomplish the task. OS will track allocation of resources to diff" process & store info in process related data structure. So this will determine process priorities for resource allocation. Then actual may on components

- Interprocess comm & sync.

One process can create another process called child process. The child & parent may execute the same or different application concurrently. The process mgmt subsystem will implement primitives for process creation, interruption, wait resume etc.
And into also maintains the parent child relationship of all the processes.

2) Virtual Memory:

achieving better efficiency for memory.
increasing multiprogramming scope.

3) File mgmt:

Info^n is stored for longer duration even when computer is shutdown. Different devices have different ways of storing data. Even with helped device driver it is difficult to manipulate. So Info^n that user sees as file (device independent concepts) whereas device store info in form of physical signal. File system will bridge the gap. It will help user organise files.

File creation, deletion, read, write etc. are defined in file system.

Communication system:-

The Network computers do not share memory of devices but interact with each other by exchanging message. So communication subsystem of OS will help in this using unique ports. A local process wants to communicate with remote process connect itself to local port, then it will write the data to local port with help of suitable comm" driver & OS will arrange to transport data to remote computer at a designated remote port layer.

Virtual File System

Utilities are system program helpful to user if they are not part of OS.

Shell:- read a new command from keyboard, execute the command & then report about it.

Library:- collection of info" → data or service. A program library is an entity containing machine language code which may be com

link time, load time, runtime.

IL is a binary image executed by CPU.

→ Diffn" both library & a utility is — library is not a standalone application program but utility is. Library can only help somebody.

- Static library
- Shared Library
- dynamic link library.

System Libraries

i) Static Libraries
ii) Shared Libraries
iii) DLL (dynamic link).

(i) Application executable. The installation is format application link time that means binding both program identifier of appn & library at link time. Every appln will have own copy of static library if ~ is changed or updated we have to recompile or relink all appln using ~.

(ii) PL is not installed into an executable but is link to executable at 'load time' : before execution starts. Can be shared by many different application. Need not to recompile program if upgraded etc other than until specification of api's change.

iii) ~ can be used any time during the running stage. Either they are installed at link now they can added at load time. They are linked to appln only when needs them. But here they have to be explicitly opened when an appln need them, binding is done at run time. Either program will explicitly close DLL when not needed or implicitly OS closes them when execution terminates.

Boot Strap –

Now we can even have multiple bootstrap programs. Diffⁿ OS have diffⁿ bootstrap program.

Time Management :-

Module Management :-

Modern OS allow for interfaces for addition of new devices to kernel at runtime. This enabled by adding new device drivers to the kernel in the form of kernel modules at runtime. A kernel module is an executable obj which can be link to & unlinked from kernel, this way we can maintain kernel size to minm level. A kernel module may reference kernel data & function.

CHAPTER-4

Processes :-

The concept of process is one of the many structuring tools devised to master the complexity of large software systems & we know that OS is also is large complex system, so process are used to handle the runtime complexity. Originally process are called job & task, when only one CPU is dedicated to one program no user interaction was allowed therefore no additional Mgmt was needed to execute the progr

Process was simply known as job execution. But in multiprogramming ~~era~~ sharing of CPU & memory was there. So each program had requirements like - maintaining status of its space for their own storage, a mechanism to record its current execution point. OS supply all this info we both which is not provided by program, this info provided by OS was structure called process of that program execution, therefore the things which we need to know is what a process is, what are its components, its roles, states that it goes through, how process are created & managed & destroyed.

We know that, for performing a task means executing a suitable program. Whenever a specific task comes OS should have a suitable program which can execute that task. Entity which carries out task mention in given program is called a process. Every process will behave acc to logic of program. We can say that process & program execution are in one to one correspondence with each other.

A process will encapsulate program code & program data, but it will also encapsulate the private data to be manipulated by program & other info required for its mgmt by OS. OS executes a program using a process as a handle.

Process has 2 basic characteristics -

i) unit of resource scheduling
ii) unit of CPU schedule for execution.

Process identification

A process is uniquely identified by its 'Pid' - process identifier. Pid ← Pt which uniquely identifies a process. OS will assign a unique value to a process whenever it creates it. This value will remain unchanged until process is destroyed.

All the control info" related to a process defines an execution context also known as process context - each process has its own within its context of a, & this context info" is structured in form of object called process control block 'PCB' or process descriptor. PCB is a data structure

Once a process is created the OS will allocate a process descriptor & address of description will used to identify the process. Physical memory is limited, so we may need to recycle some PIDs for new process. So PID values are kept in dependent of PD address. For each process in addition to PID, OS will maintain many diff^{n} identifiers used for controlling, access to resources & for interprocess comm^{n}. eg:- user identifier - each process's owner's user, identity of user is stored in another process group identifier. Process group identifiers are mainly used for multicast messages and interprocess messages.

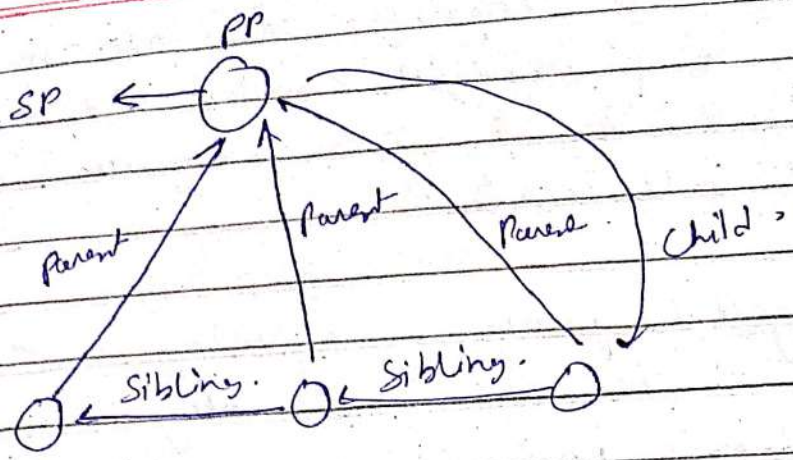
Interprocess relationships

child of a process.

Parent-child relation forms a tree of process with root process.

Process with no child is leaf ~~of~~ internal process.

This relationship can implemented through set of pointers (stored in PDs) - parent pointer, child pointer, sibling pointer.



→ parent pointer points to descriptor of parent.
→ " " of next process point to null.
→ child pointer will point to last child alive and do null if no child alive.
→ sibling pointer points to next younger process alive.

Various process States

By state of a process we mean the gross indication of what is current condition & situation of process. Value of all data in its address space process context will collectively determine the state of process. These state of process can be :-
Some of common states are :-

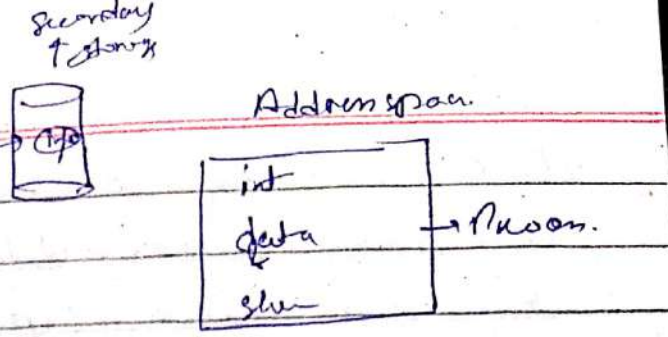
i) Starting state - signifies that process has been just created, OS is setting up initial execution context & initial private address space & will be allocation some resources.

ii) Ready state: signifies that process is ready to run but CPU not available, waiting for CPU use queue until it is allocated it

iii) Running state: process is running in CPU, i.e CPU is executing instructions from current address space & if new need of resources arise OS will give it to it.

iv) Waiting state: it is not ready yet & accumulating required resources. This indicates process is waiting for event like I/O. it will not resume execution until event occurs. It will sleep until OS wakes it

v) Exiting: May be normal or abnormal. OS will clean up process address space & resources will be freed up of the process.




```
File info;
int item;
open (info, read);
while not
        end-of-file (info)
read (info, item);
- - -
- -
-
print
Stop;
```

There can be two kinds of relationships i.e a -

- a single execution of a sequential program (program will consist of main program & set of function during execution control will shift main to function acc^{n} to logic.

execution of program constitute single process. OS will not be aware of existence of function. B

- a many to one relationship exist^{s} betn many process & a program. When several execution of program exist in program at same time. And during such execution a program coded will inform the OS about the parts which are to be executed concurrently.

The OS will consider each execution as process & have many to one relationship, such programs is known as concurrent program. And they will coexist in system at some point of time.

a) Child process:- Main process will create another process - child. Main advantages of creating child processes are computational speedup.

priority of children.

b) Priority for critical functions:- Parent process will decide

c) Protecting parent process from error:-

The OS cancelled a child process if an error occurs during its execution, it will not affect the parent process.

From the programmers view point processes are just means to achieve concurrent execution. The main process will create child processes, assign priority to them for achieving certain objective & main process & child process will interact to achieve common goal.

The interaction may involve change of data & also processes coordinate activities with one another.

The OS will provide four function to provide programmer viewpoint

i) creating child process & assign priorities
ii) terminating " "
iii) determining status of child process
iv) sharing info & synco between child & parent

There are four kind of process interaction -

i) data sharing

Shared data may become inconsistent if many process update data at some time. Hence processes must interact if it is okay to work the data -

ii) message passing

Process exchange info by sharing messages.

iii) synchronization

Fulfill a common goal process must coordinate with each other.

Signal is a mechanism to convey occurrence of some exceptional situation in a process.

From OS point of view the process is an execution of a program & for this OS will create process schedule them for use of CPU & terminate them. And to perform scheduling OS should know which process requiring CPU. For that it has to monitor all process. So that it should be knowing which process is doing what. OS will take help of process states

A process has a few components

- PID
- code
- data
- stack
- CPU state

other than CPU

It also use other resources like memory I/O's, therefore OS has to maintain this info^n

And in process control block -

pid, process state, register values, program counter value.

The process environment contains the address space of a process. The OS will create the process environment by allocating memory to process & loading the code in allocated memory

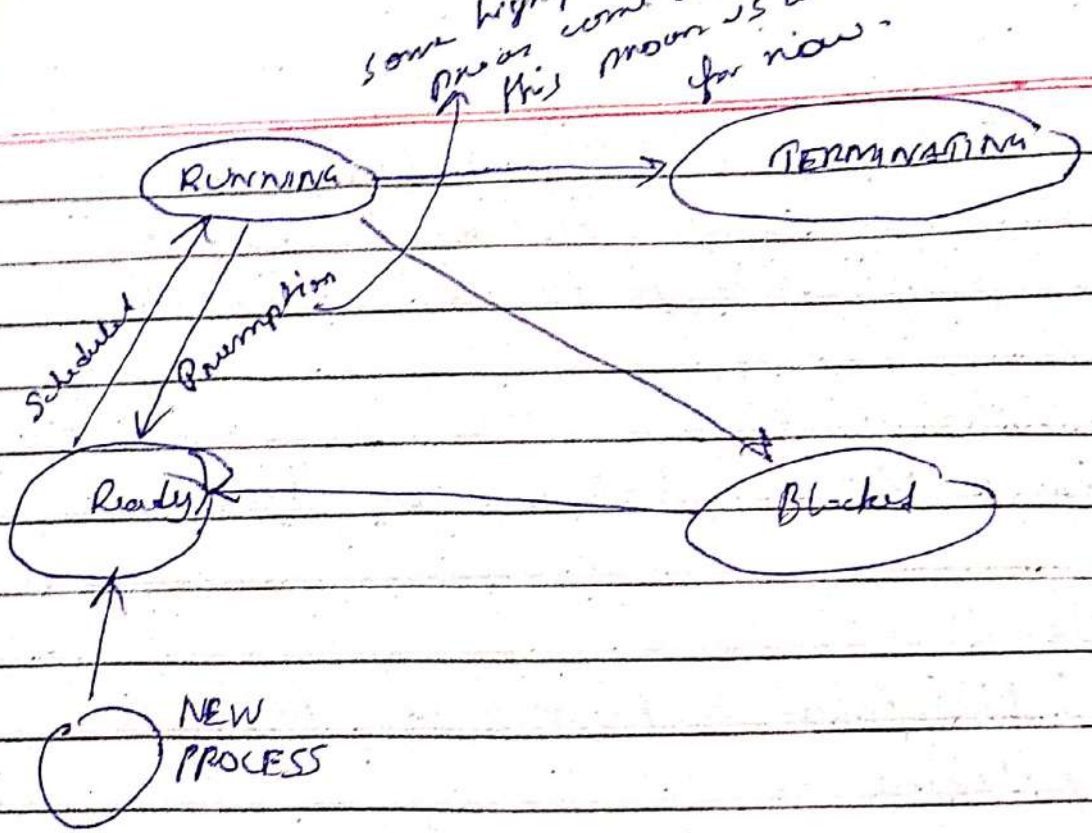
& setting up its data space. OS will also put info^{n} concerning access to resources allocated to process & its interaction with other process with the system.

→ PCB is a kernal data structure. Kernel will use three fundamental routine to context process

- scheduling
(Selecting process to be executed next)
- dispatching
(set up the execution of selected process on the CPU)
- context save
(Save info^{n} related to a running process when its execution is suspended)

(In other words we can say that context save is reverse of dispatching).

A straight transition for a process pi is a change in state caused by occurrence of some event in system.



The figure shows the fundamental state transition of process.

RUNNING :- CPU is currently executing instructions of that particular process code.

BLOCKED :- Process has to wait until a resource request made by it is granted or it wishes to wait until a specific event occurs. CPU shouldn't be allocated in this state.

READY :- Process wish to use CPU to continue its operation, but it hasn't been scheduled yet.

TERMINATION of proces has completed normally or OS has aborted it.

Threads

They provide a low cost method of implementing concurrency. Process switching has two overhead-
- execution related overhead.
- resource use related overhead.

The proces environment contains info related to resources allocated to proces which usually is large size of proces state info which is a overhead.

To reduced switching overhead we can eliminate resource related overhead in some situation. Because occurrence of event may result in switching from proces pi to pj. If both pi & pj belong to same applh they will share their code, data & resource only value in stack & registers will vary.

A Thread is a programm execution that uses resources of a proces. It will have its own stack, own CPU state, thread of same proces share code, data & resource with each other

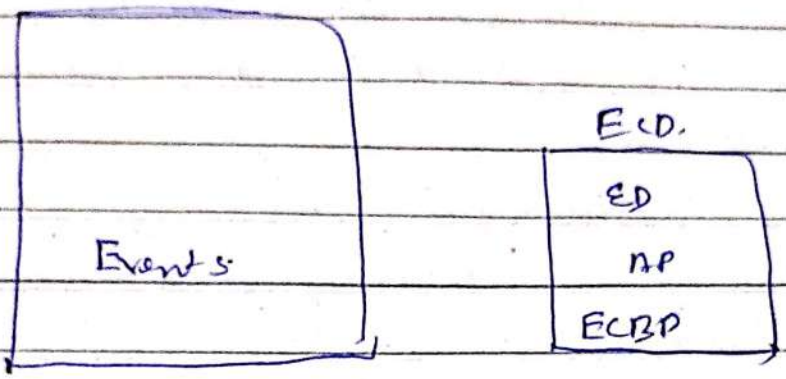
The kernel will allocate a stack to thread control block (TCB) to each thread. This thread will execute within envi" of P, & OS know this thread will save on CPU stack & steel pointer when switching both threads of some process. The use of thread split the process into two parts:

Thread state transitions are analogous to process states, when a thread is created it is in ready state. Because process already there necessary resources theoritically

* thread cannot enter blocked state like process, but it can enter block state because of process sync: requirements

ECB

PCB.



Thread is to ensure.

- Correctness of data sharing & sync.

- concurrency will increase -

Function or subroutines that uses static or global data can produce incorrect results when invoked concurrently.

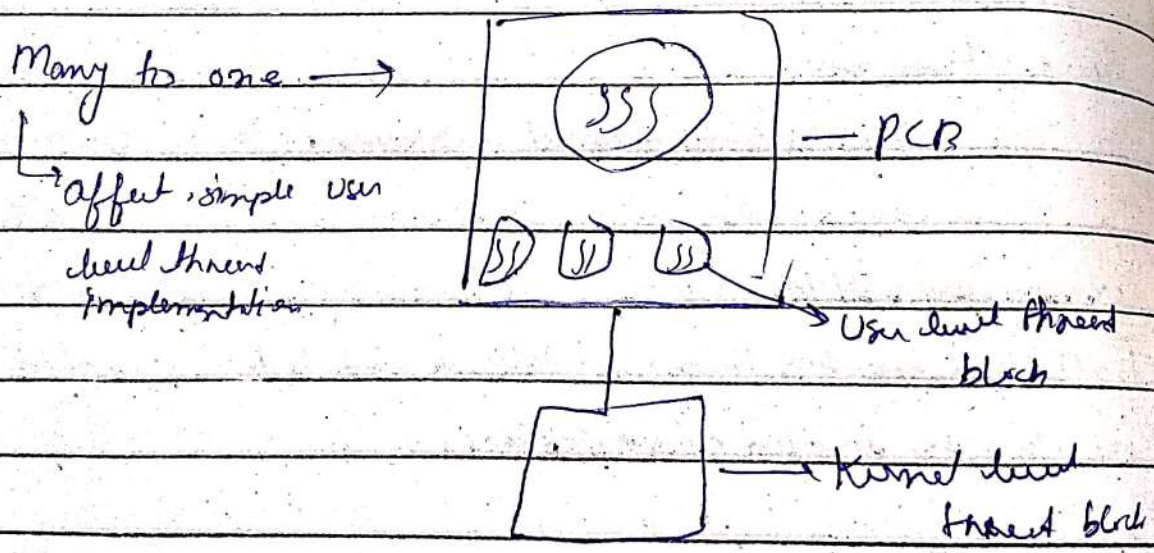
⇒ Thread unsafe:

Application using thread must be coded in thread safe manner & must invoke routines from thread safe library.

Kernel level thread.
User level thread.

Thread in running state invoke a library function to sync its function with other threads, the library stop scheduling & switch to another thread. But the switching is not visible to kernel.

If thread library cannot find ready thread it will make system call. This process will unblock with an event activates.



One to one — similar like kernel level threads.

Many to many — implementation is complex

OP
P)

Interacting process

$P_i \longleftrightarrow P_k$ if $read\_set_j \cap write\_set_i \neq \phi$

- $read\_set_j$ → set of data item read by P_j
- $write\_set_i$ → set of data items modified by P_i

*) Process which do not interact are called independent process.

⇒ Race condition :→ 3 machine instructions:
- load, add & store
- during execution of concurrent process.

⇒ ds, fj do → $f_i(F_j(ds)) \neq f_j(F_i(ds))$

*) A race condition exist in process p_i & p_j if update of update $(S_i) \cap$ update $(S_j) \neq \phi$

Control Synchronization:

- may be required at any point in lifetime.

Q. A program is to be designed to evaluate the batch form of a computation which consists of following actions:-

① Compute $Y = HCF(A_{max}, X)$

here array A contains n element & A_{max} = maximum value in array A.

② Insert Y in A

③ ~~Array~~ Arrange A is descending order.

Ans:-

(i) Read elements of array A

(ii) Find A_{max}

(iii) Read X.

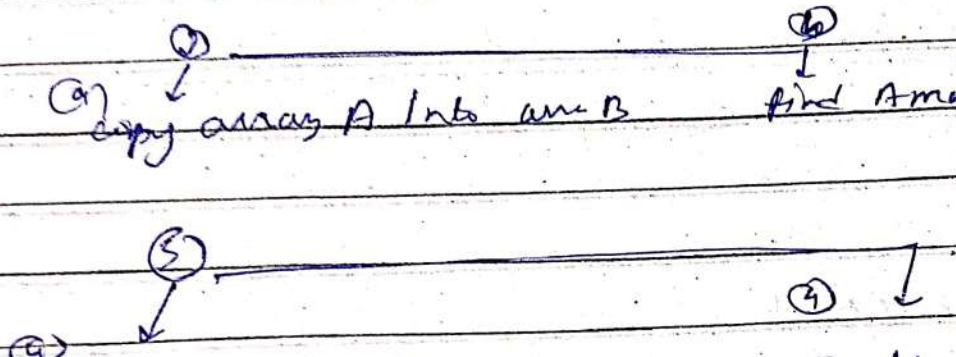
(iv) Compute Y

(v) Include Y in array A & arrange A in descending order.

Now to decide which of these statements can perform concurrently we will consider each of the statement as separate process & find which of them can interact.

(1) & (3) can be performed concurrently.
by (2), (4), (5) they are interacting processing they cannot be executed concurrently as they share array A & one process is modifying A.

Concurrency can extended by splitting (2) & (5)



Now 2(b) & 5(a) are independent of each other hence executed concurrently.

After 2(b) step 4 can be performed concurrently.

P_1 P_2 P_3

Read n elements of A & copy A to B | Find Amax | Read X

P_4 P_5 P_6

Compute Y | Arrays B is descending order | Insert Y

P_1 & P_3 → concurrently

P_1 & P_2 → X (shared array A). P_2 can be initialized after P_1 ends

P_4 → can be initiated after P_2 & P_3 either terminates

P_5 → " " " " P_1 terminates

P_6 → " " " P_4 & P_5 are terminated

SCHEDULING

Scheduling will determine quality of service. (QOS).
It will also influence the performance of the system.

Request related :-

i) Arrival time :- time when a user submits a job.
ii) Admission time :- a time when the system starts considering the job for scheduling.
iii) completion time :- time when a job is completed.
iv) deadline :- it a time by which a job must be completed to beat the response requirements.
v) Service time :- it is the total time of CPU + I/O time.

LTS - long time scheduler MTS - midium " STS - short "

vi) preemption :- forced reallocation of CPU
vii) priority :- it is the tie breaker rule
viii).

User's service related :-

→ ① Individual level
② System level.

Individual -

i) Deadline overrun :- amt. of time by which completion time exceeds its deadline.

late ↑ early ↑

i) deadline overrun can be positive or negative.

ii) fair share :- it a specified share of CPU time that should be devoted to execution of a process

iii) response ratio :- time since arrival + service time / (CPU time + waiting)

iv) response time :- it is time bet^n submission of a job to the time the time becomes available.
(mainly to interactive application, give input & receive response).

v) turn-around time :- (non interactive appl^n) it is a time bet^n submission of a job & its completion by the system.

vi) waited :- it is the ratio of tat to its service time.

1
10
11
100
10.1
110
11

Mean response time: average of response time of all request serviced by system.

Mean turn-around time:

Scheduling related:

i) Schedule length → time taken to complete a specific set of jobs.
ii) Throughput → avg no of jobs completed by system in one unit of time.

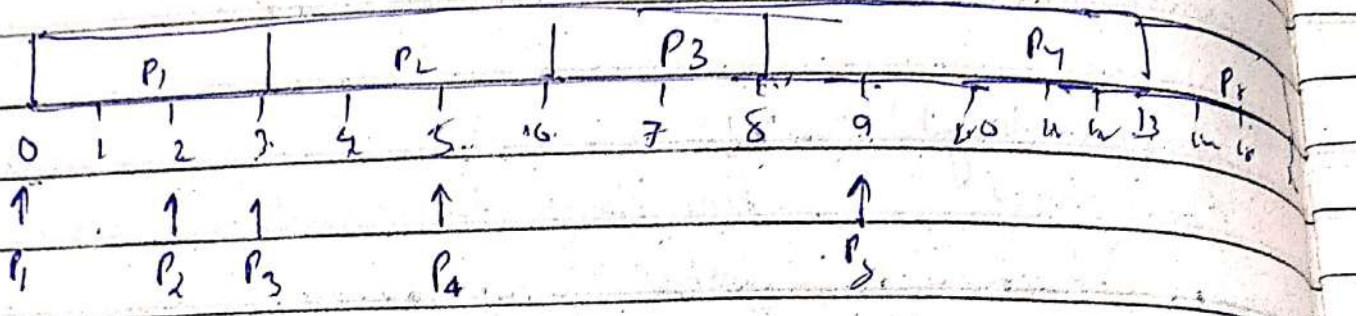
Scheduler has three fundamental techniques:-

i) priority based
ii) re-ordering of request - FCFS
iii) variation of time slice.

eg:

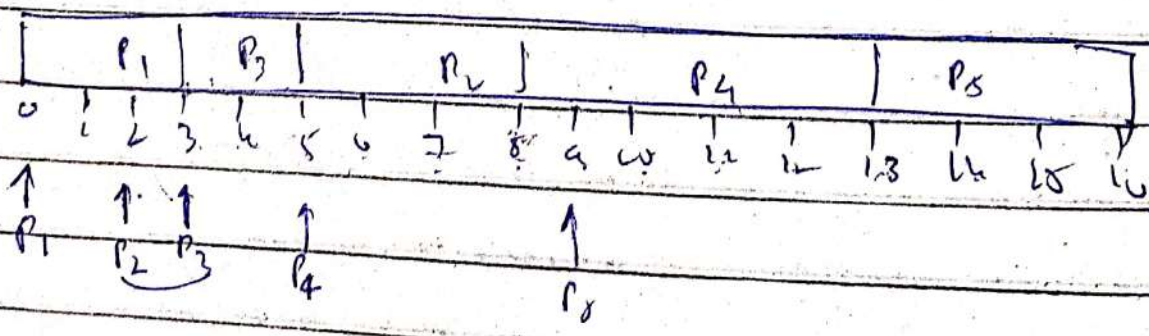
	P_1	P_2	P_3	P_4	P_5	
AT	0	2	3	5	7	(arrival time)
ST	3	3	2	5	3	(service time)

FCFS



	TAT	WTAT	RTAT	ATAT
P_1 -	3	$3/3 = 1$		
P_2 -	4	$4/3 = 1.33$	$\frac{3+4+5+8+7}{5}$	1.75
P_3 -	5	$5/2 = 2.5$		
P_4 -	8	$8/5 = 1.6$	$= \frac{27}{5}$	$\frac{15+20+18+12}{15}$
P_5 -	7	$7/3 = 2.33$		
			$= 5.4$	

SRN (less service time allows)



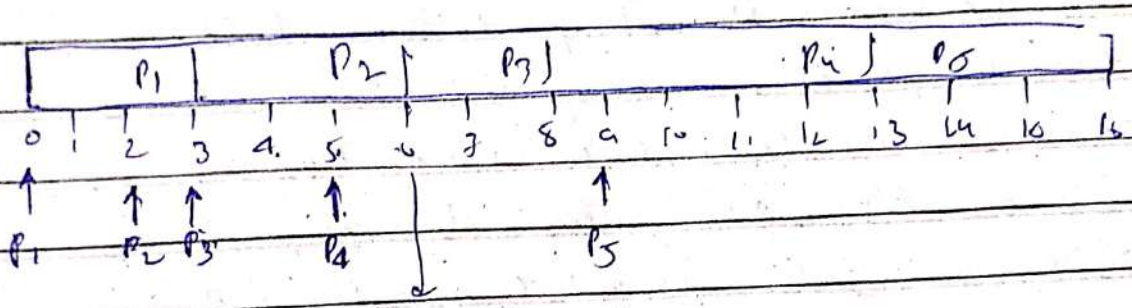
as
SJF (P_3 < P_2)

2.8 / 2.80

75 | 117 | 33 | 1.40 | 1.5

	TAT	WTAT	ATAT	AWTAT
P_1	3	3/3 = 1		
P_2	6	6/3 = 2	= 24/5	= 4 + 8/5 + 7/3 / 5
P_3	2	2/2 = 1	= 9.2	= 60 + 24 + 35 / 157.5
P_4	8	8/5 = 1.6		= 119/75
P_5	7	7/3 = 2.33		= 1.58

HRN (Highest response ratio Next)



P_3 rr = 3/2 = 1.5

P_2 = 1/5 = 0.2

	TAT	WTAT	ATAT	AWTAT
P_1	3	1		
P_2	4	1.33	5.4	1.35
P_3	5	2.5		
P_4	8	1.6		
P_5	7	2.2,		

Dhondare

Implementation Guideline Scheduling

Priority Based

i) An IO bound job will have higher priority
ii) Process priorities are static.
iii) Process scheduling is preemptive.
iv) Scheduler will maintain two separate list - ready process & blocked process & select highest priority process from ready list.

It will also maintain pointer CRP (currently running process) which will point to entry in PCB of the process in running state. Whenever interrupt occurs the context save mechanism will save PSW & CPU registers into appropriate field. And when a process terminates PCB is removed from list & for selecting scheduler will scan PCB list from beginning of ready list.

*) If scheduler do not find any ready process

Round-Robin

Two list will maintain. PCB of newly created process is added at end of ready queue & of a terminating process is removed.

When a process finishes its time slice, but its execution not finished it gets back to ready queue.

Multilevel scheduling

Multiprogramming & time sharing represents extreme position on giving high level of efficiency & guarantee of good service. ~ will provide a hybrid soln.

The scheduler will maintain no. of ready queue & process in different queue have diff^{n} priorities & receive different time slice.

⇒ High priority process will have small value of time slices (as service will be good).

Low priority process are expected to keep CPU busy (provide good efficiency).

Each ready list has a pair of attributes.

↳ i) time slice

ii) scheduling priority.

time slice $\propto \frac{1}{\text{priority}}$

Scheduler will scan the highest priority queue until no process are there if will go to medium & then to low priority queue.

Of course some drawbacks are there :-

i) Categorization is to be done into various level. classification is static i.e if wrong classification is done that it will stay for some fault.

ii) cannot handle computational or I/O behaviour of a process.

To ~~comp~~ overcup these drawbacks :-

- Multilevel Adaptive scheduling was proposed also known as Multilevel feedback scheduling.

- we observe recent CPU & I/O ~~process~~ usage used to determining scheduling level.

So it schedular can adapt to change in behaviour of process.

eg- ~~IBM 704~~ IMP 7904 (CTSS scheduling)

↓

compatible Time sharing system.

Fair share scheduling

A fair share is fraction of CPU time that should be devoted to execution of group of process belongs to some application or project. Scheduler will consider a process from a group if the group is not given fair share then it will set value of timeslice.

6) Lottery Scheduling:

Power management :-

Sleep Mode → light

→ heavy.

Holder

Real Time Scheduling:-

The real time OS are diff^{n} from general purpose OS.

i) Real time OS are designed for specific purpose
ii) " " " " " fast response & deadline.
iii) General real time OS are used in small devices.

- tailor made, less standardised.

RE (real-embedded) systems.

Most RE systems are P/D dominated & part of a physical system.

Generally RE deals with both interrupt & exception handling: Exceptions are introduced by the environment. Such type of scenarios may not come in design phase, so therefore we need both mechanisms.

Another major diff is deterministic timing behaviours of RE system (within a known & expected amt. of time). Features in general OS's whereas this is non-deterministic.

Various structures of RE OS:

i) basic interrupt driven task model
ii) round robin based models
iii) mixed " " "
iv) monolithic " " "

Real time scheduler must make sure that task must their deadline.

Fundamental question in scheduling is to check whether a given set of task can be executed in system without missing dead line.

Each real time task is assigned a priority & deadline & almost all real time task are executed periodically.

Most RE algos are priority based preemptive algos.

Real time scheduling can be classified into two categories, static & dynamic priority algos.

Periodic task model is simplest & sufficient for many systems:

i) all tasks run periodically in simple CPU & deadlines are end of their periods.

ii) tasks are independent (P1, P2, P3 ---

iii) execution time for each task is fixed.

iv) ignore the context ~~as~~ fixed time.

two different types of scheduling algorithms introduced :-

i) RM (rate monotonic)
- fixed priority scheduling algo
- high probability deadline

ii) EDF
- dynamic priority
- higher priority to task with current earliest deadline.

Two imp objectives of RT scheduling :-

i) at any time the scheduler selects the highest priority ready task

iii) selected task runs until it completes its execution or another task with higher priority is ready for execution.

*) Ready queue
   - ordered by task priority.

*) Wait queue
   - contain task which have already run & waiting for next period to run again.
   - ordered by earliest start time.

Scheduler examine task in wait queue if any task is to be moved to ready queue & compares head task of ready queue to currently running task, if priority is higher than a context is invoked.

↑ rate monotonic

RM Scheduling:

The way we calculate priority is important as dependent upon task periods (frequency).

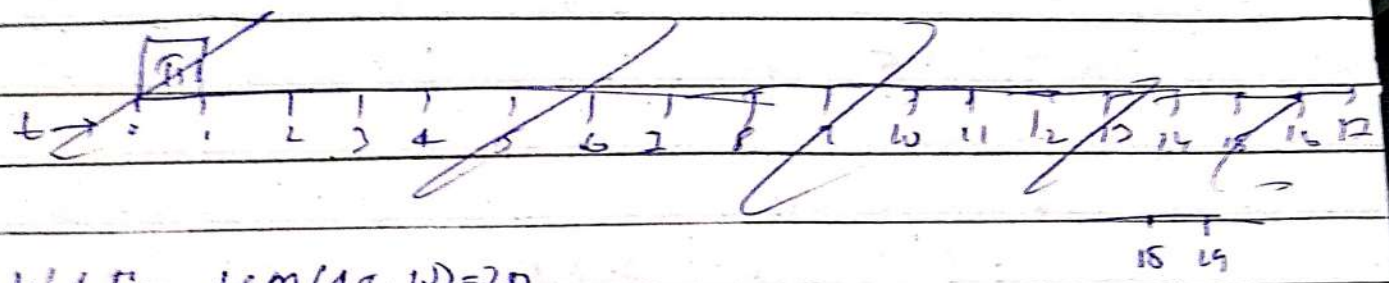
Task with higher rate of occurrence has higher priority i.e. Hence rate monotonic.

eg:- T_1 (1, 4) = 1/4 ← Task period.
T_2 (2, 5) = 1/5 tasks are activated at every
T_3 (3, 10) = 1/10 starting point
↓
freq.

∴ priority $T_1 > T_2 > T_3$

at time t0 all are ready, T_1 has highest priority hence start execution & complete at $t_0 = 1$, now T_2 & T_3 are in tie, naturally T_2 will start, at time t=3 T_3 can start but at time 4, T_1 will become active thereon he is both T_1 & T_2 so T_1 will execute naturally as it has high priority

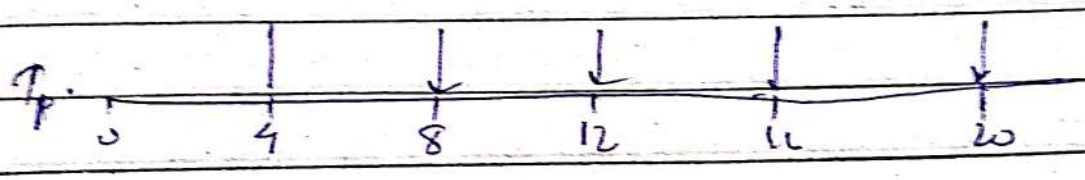
Timing diagram

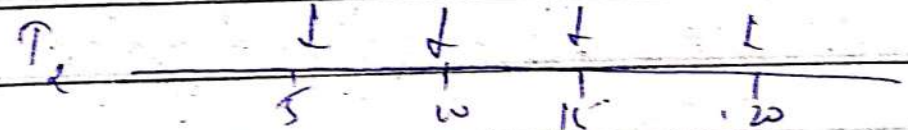


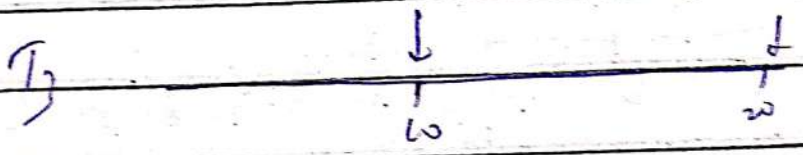
Solution LCM(4,5,10) = 20

5 + 4 + 2

× 1 × 2 × 3 = 17 theoretically possible







Eg: $T_1(3,4)$ $U_1 = \frac{1}{4}$

$T_2(2,5)$ $\frac{1}{5}$

$T_3(3,12)$ $\frac{1}{12}$

theoretically not schedulable

LCM = 12. 3 + 2 + 1

× 2 × 2 × 3 = 13

Q. $T_1 (1,3)$
$T_2 (1,4)$
$T_3 (2,5)$.

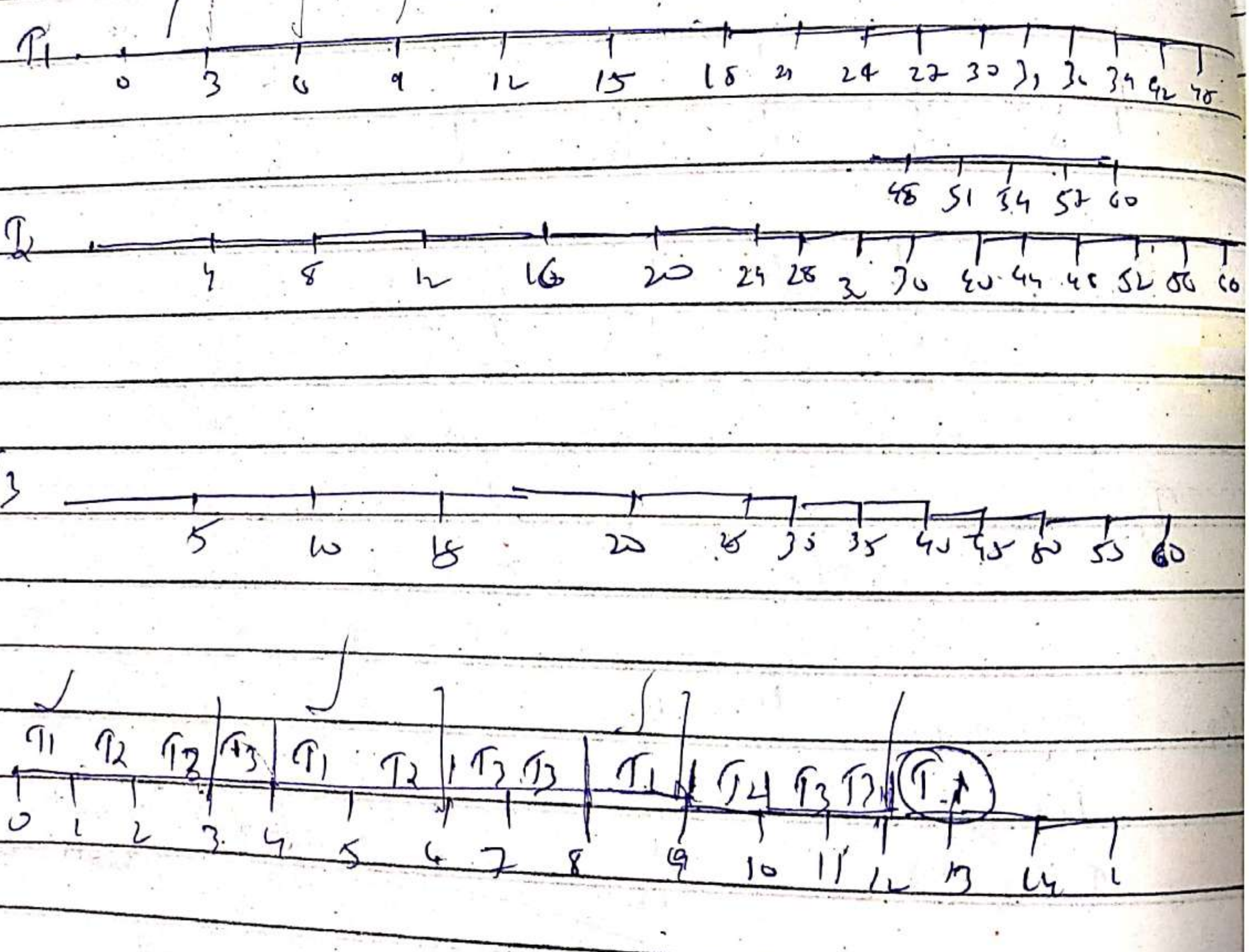
lcm = (3, 4, 5) = 60

20   15   12

×1   ×1   ×2 = 20+15+24

= 59 , theoritically possible as < 60

means T_1 should always come both those intervals if not then scheduling is incorrect.



(EDF)

Purpose of EDF is assigning priorities dynamically. Here the highest priority task is one whose deadline is earliest, therefore priorities may change at any scheduling point.

Use notation $S(T, t, d)$

task scheduled at time t for duration d

eg $S(T_1, 0, 1)$ - task T_1 must scheduled at time=0 for 1 unit time.

$S(T_2, 1, 1)$

$S(T_3, 2, 2)$

$S(T_1, 4, 1)$

T_2

a_1

Memory Management System

Memory management system:-

We know that if we store many applications at any point of time the multiprogramming performance would increase. Portion of memory where OS occupies should be memory, portion in which process are loaded that we call dynamic memory.

Of course OS will also use dynamic memory. We also know that main memory is limited so anything is needed. To do all this things we need to part of manager known as memory manager which will deal with allocate / deallocate, sharing of memory, protection. In general memory management is concerned with units of memory allocation when we main memory the units are constructed, how various process are referred this units, whether this units can be shared & how this units are protected.

Core issue over here is assigning suitable memory address to instruction & data. Each program is written with their own addressing scheme independent of physical address space. Who is involved in this. All this it is part of ~.

Of course OS will get some help from processor hardware. ~~of course~~