→ Set of programs that Manages, collect & coordinate all part all functions of the system.

— OPERATING SYSTEM

Collection of various programs which realize effective utilization of a computer system in a given computing environment.

- OS acts as Resource manager & allocator

P O S D C O R B
(staffing, Resources, Distribute, Budgeting)
Planning/Processing organising, Co-ordinate

② provide platform to perform user applications

- OS is a system software

The main resource, it manages is computer hardware effective use in form of processor, storage, input/output device & data

⇒ OS performs many function :-

1. Implementing the user interface
2. Sharing hardware among user
3. Recovering from error
4. Scheduling resources among user
5. facilitate parallel operation
6. Accounting for resource usage
7. Preventing user from interfering with one another

OS
- Kernel (Main part) → Permanently resp OS run
- Non Kernel (Supporting prog)

Kernel

The kernel layer consists of OS programs which reside in memory at all times.

Non-kernel programs normally reside on hard disk & are loaded in memory when needed.

Non kernel programs use facilities provided by kernel program.

Programs in the user interface layer similarly use facilities provided by non-kernel progrm

Only the kernel programs interact with computing system hardware.

⇒ Classes of Operating System.

OS class	Period	Prime concern	Key concepts
• Batch processing	60's	CPU idle time	Spooling, turn around time
• Multiprogramming	70's	Resource utilization	Program priorities preemption
• Time sharing	70's	Good Response Time	Time slice, round-robin scheduling
• Real Time	80's	Meet the deadline	Real time scheduling

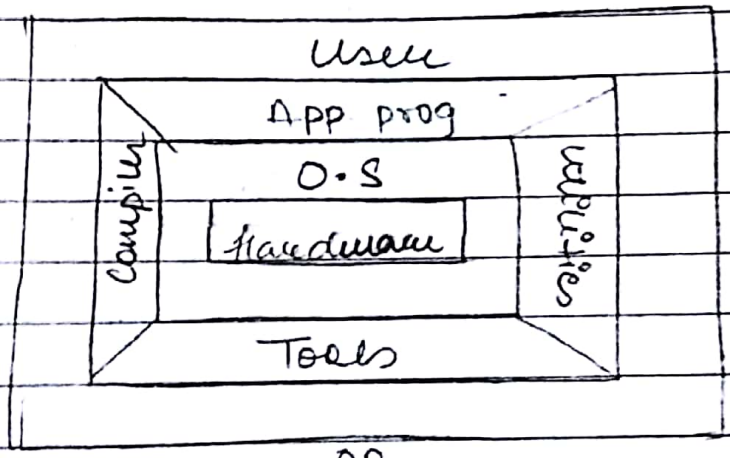
• Distributed	90's	Resource sharing	Transparency, distributed com

Functions & Modules of O.S

state of process determine, creation of processor & ck

1. Process Management system
2. Memory management system
3. Device Management system (Accepting data from keyboard)
4. File management system
5. Security management system
6. Communication management system.

Diagram



— Block dia of O.S organ

OR

User
Shell
Interface b/w kernel & user
Kernel
Interface b/w hardware & kernel
(Hardware)

Arch of unix sys — fig

P + 08

Bare Machine Approach.

Machine without O.S
Computer used to be programmed directly in machine language, without any system software approach.

Program could be entered into RAM through front panel switches

Results of execution could be displayed on a set of LEDs mounted on front panel itself.

Set of LED same monitor

TYPE's OF O.S :-

Serial Processing

The programs were executed strictly in serial manner

P_1 P_2 P_3 P_4 →
serial manner

Program were code were written in assembly language & high level language.

Batch Processing O.S

With the invention of hard disk drive, things were much better.

A memory resident portion of the Batch O.S called batch monitor.

Batch monitor used to read, interpret & execute these commands automatically & group the pooled jobs into separate batches placing similar jobs having identical needs in the same batch, based on pre-specified criteria.

ADV - Time saving as loading & unloading is reduced.

Multiprogramming O.S

CPU bound + I/O prog
then useful.

It refers to the concept, wherein more than one programs used to be activated concurrently; one of the active programs being in "run-mode" utilizing the CPU & other utilizing the I/O device at the same time or being in a 'wait state' waiting for resources to be available.

This improved the utilization of system resources, thus increasing the system throughput

No of task performed per unit time

⇒ Interactive Time Sharing System. O.S

Such systems, in addition to multiprogramming support some additional features like permitting user interaction with the system through OS commands.

Each interactive user is assigned a time slice in a round robin fashion, during

jese got gape mein kuch hai

user should get a feeling that only its work is been done by the CPU.

which it controls the CPU. During the time-slice, the process gets control of CPU & tries to complete its computation.

If the computation is not completed during the assigned time slice, then the time slice, then the running process is preempted; then it has to wait for time slice in the next cycle to resume computation from where it was preempted. The time slice cycle is so adjusted that each user has a feel as if the CPU is assigned to it all times.

active thing
↑
• Process control Block → PCB; value of PC is stored in PCB
↓
Program in execution

Multiprog. — Multitasking — Multithreading

↳ Multiprocessing → Processor ...

Multithreading → same resource, KO share

↓ jyoda h.

deadline

* Program in execution — Process.
Passive thing ↓ Active thing.

27/1 ⇒ Process management

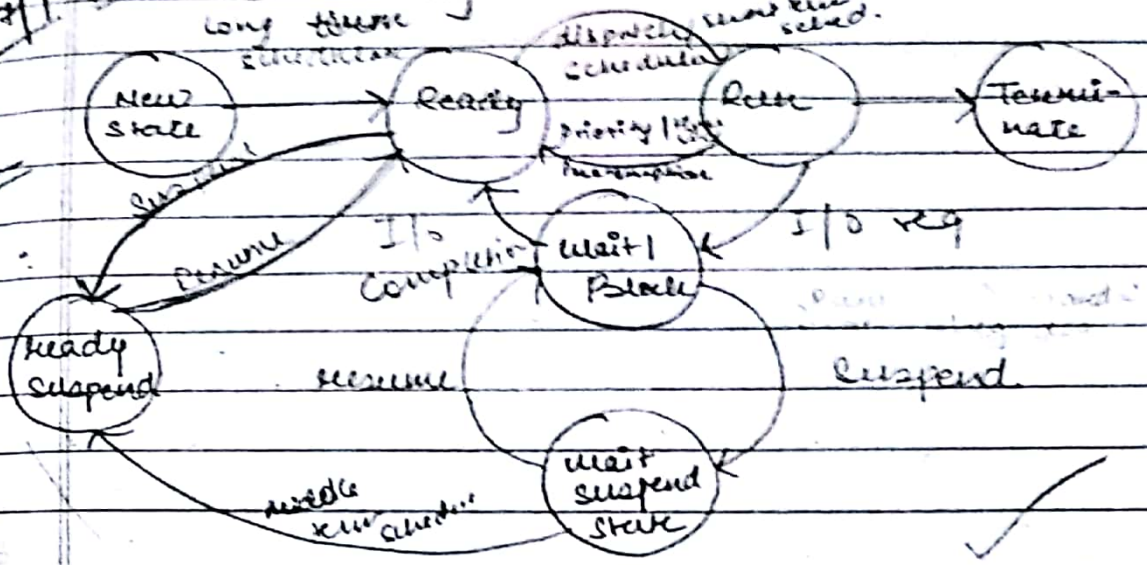
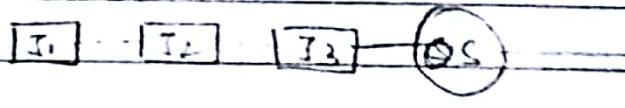


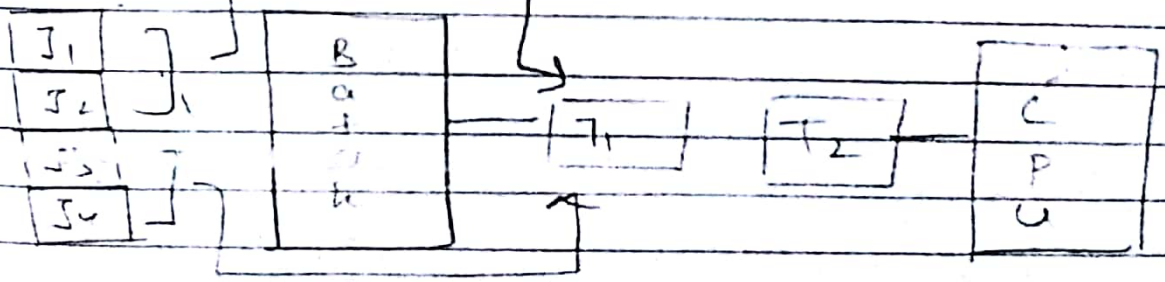
fig - State transition diag of process state.

Block diag.

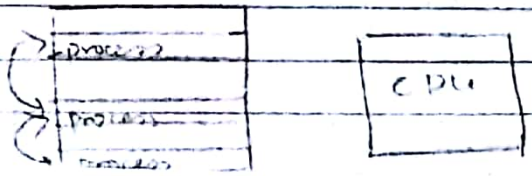
① Serial O.S



② Batch



③ Multi.



→ Multiprog. is not benificial here
→ Time oriented job. Objective + (↑) throughput

Real time O.S - not stored in hardware ∵ time consuming

It refers to the environment of embedded system (chip device mai hm embedded hO jate hai, so that hardware se transfer karne mai time na waste hO), with a very rigid requirement to complete the processing of input data, in a pre-specified time. The input data in such system, is invariably received from real world sensors. The outputs may be used to control some real-world process. For eg - an air defence system, receiving information about aircraft movements from radars & control weapons for the elimination of hostile aircraft.

Real time

- Hard Real Time
- Soft real time

Hard Real time

All critical tasks have to be completed strictly within the specified time limits. Not meeting the deadline can be catastrophic ; like an embedded system controlling the operation of a refinery. In such system, all code & data must be in RAM / ROM.

II Soft Real time -

This is less restrictive type of the real time system. If the input data is not processed within specified time intervals, the results may not be catastrophic, but the output may loose its utility. The soft real time systems are not suitable for industrial & defense use.

Distributed OS

A distributed OS provides easy sharing & good utilization of resources located in all computer in the system. To achieve this, it permits a process to utilize the resources located in a remote computer using the networking component. It may also execute processes of a program in different computers.

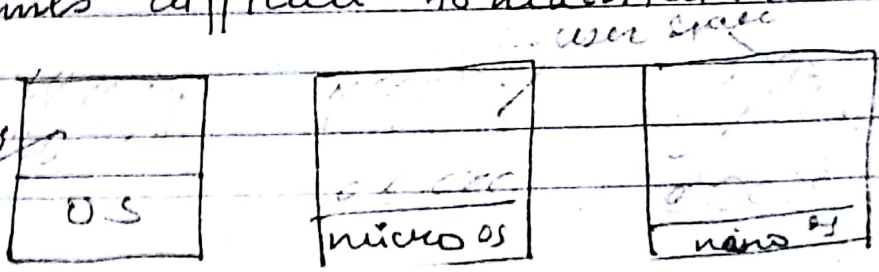
Bridge b/w application & hardware

⇒ Monolithic Kernel → No extra communication channel req. is faster

A monolithic kernel is a large single piece of code, composed of several logically different program pieces. It is one single large program where all the functional components of operating system have access to all the data & routine. (All those components reside in the kernel space) such a program, with the passage of time, grows more & more complex, & becomes difficult to maintain.

Advantages

ADV - ... less time consuming ... monolithic



Unix use micro kernel
Windows use monolithic

To avoid these problems, modern monolithic kernels are structured in strictly functional

Monolithic kernel	Microkernel
Bulky	Light
less space for user	More user space
Diff. to enhance	Easy to enhance
low degree of multipr.	degree of Multiprocessing
High execution	Slow execution

unit. One unit cannot directly access data & routine belonging to other units. The units follow strict communication protocols to avail services from one other. This is though purely a programming paradigm, & whenever be the internal structure, every part of the operating system runs in the kernel mode on behalf of the running process.

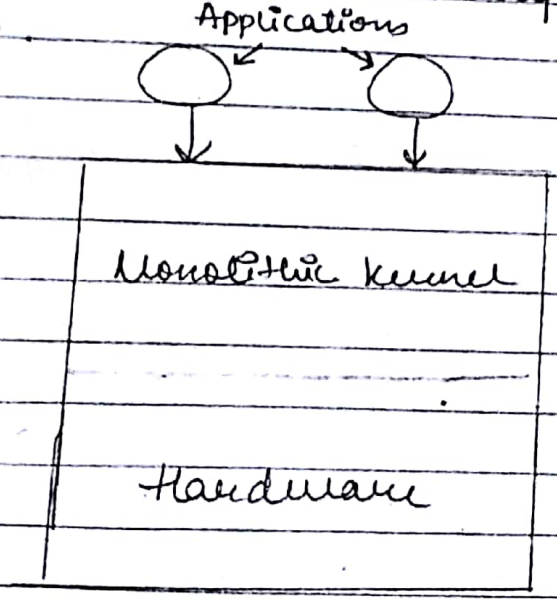


fig → Monolithic kernel

Testing - Finding error
Debugging - Fixing error

Microkernel Architecture

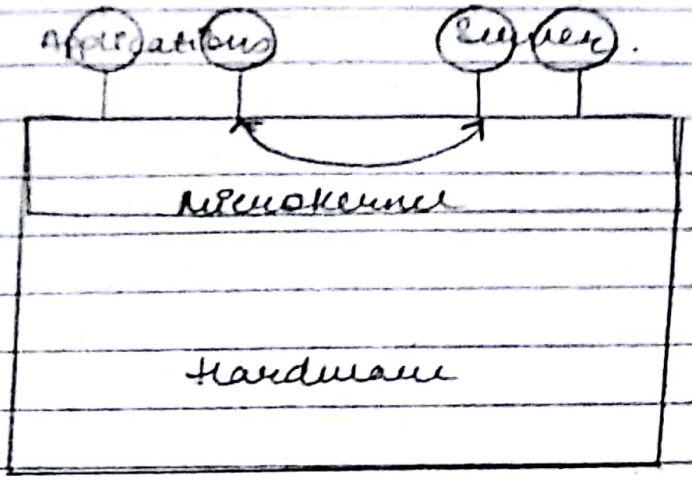


fig- Microkernel Architecture

A microkernel system is only that provides only the bare minimum functionalities in the kernel, & hence, is quite small & compact. The aim is a kernel that provides the most basic functionalities to construct the minimal operating system services; a few inter-process communication & synchronization primitives, a processor scheduler & an interrupts dispatcher.
The remaining functionalities are implemented through autonomous processes called operating system service processes or servers.
The server programs reside outside the kernel in different address space. Thus, servers are not truly kernel process. They run in the user space, & not in kernel space.

The user level server processes provide services that have been traditionally parts of monolithic kernel executed by client processes when they

operate on the kernel space. They include services such as *memory manager, device driver, system call handler & others.*
The services offered by the servers are called from the kernel via up calls.
A client does not interact directly with a server. Clients in fact are not aware of servers, & they interact with the operating system via regular system calls. This approach helps in making the kernel smaller. It is easier to develop & test new services or to modify existing services in such system.

Monolithic V/S Microkernel

The OS development code base is relatively easy to maintain for microkernel - based systems. However, it leads to slower execution of OS services as they are high overheads involved in it.

... (Same as written in last page ii) → Comparision.

Reentrant Kernel

Several processor may be in kernel mode at the same time. A reentrant kernel is able to suspend the current running process even if process is in the kernel mode.
Unix kernels are reentrant kernel. This means that several processes may be executing in kernel mode at the same time

Date 3.12.18

A kernel that implements "reentrant routines" are referred to as "reentrant kernels". When two tasks can execute the function at the same time without interfering with each other, then the function reentrant.

In non-reentrant kernel mode, every single process acts on its own set of memory locations and thus cannot interfere with the others. Reentrant functions only modify local variables but do not alter global data structures. To provide reentrant kernel, the kernel is implemented as a collection of reentrant functions.

If a hardware interrupt occurs, a reentrant kernel is able to suspend the current running process even if that process is in kernel mode. This capability is very important, because it improves throughput of the device controllers that issue interrupts.

Various Criteria for Measuring Scheduling

CPU Utilization : It is average fraction of time during which CPU is busy or during the time CPU is not idle or to consider useful work only thus excluding time spent executing the OS.

Throughput - It is amount of work completed in a unit of time i.e. number of user jobs executed in a unit of time.

Turn-around Time - It is the time elapsed from the moment a program or job is submitted until it is completed.

Finish time - Starting time

Waiting Time - Turn-around - Actual execution time of a job. It is time job spends waiting for resource allocation.

Response time - Response time can be classified in terminal response time & event response time. It is defined in time sharing system as the time that elapses from the moment the last character of the command enters & the result obtained from the monitor

Date .....

UNIX :
MULTICS (Multiplexed Information of Computer Service)
operating system was developed by Bell Lab, General Electric
Company and Massachusetts Institute of Technology (MIT) in 1965
UNIX (Unplexed Information of computing service was
developed by Ken Thompson, Dennis Ritchie.
Firstly UNIX is written in C language to install in any machine.

short term scheduling

CPU Management

CPU is the most valuable hardware resource in a computer system & it is the only resource that actually executes processes. The objective of OS is to raise the effective utilization of the CPU as much as possible. To do this, multiple processes are allowed to co-exist in the system. By switching the CPU among processes the system ensures that all processes have a fair share of CPU time.

Scheduling Objective.

In computer system, many application & kernel processes can run concurrently to improve the utilization of system resources. Processes use various resources to accomplish their tasks. The use of resources by concurrent processes often lead to conflict over resources, & unless resolved by the OS, the reliability of the system might be jeopardized.

The system needs to resolve access conflicts by allocating resources to processes in an orderly manner.

The set of rules by which the system evaluates, who is given a resource, when & for how long is called Resource allocation or scheduling policy.

The management of resource time is called resource scheduling. The goal of resource scheduling is to provide a predictable

PCB - Process Control box / block elements
→ Process Id, name, value of PC, stack reg
→ memory management info.

level of services to all the processes that complete for a resource.
Resource scheduling is purely an operating system activity, & applications are not involved in the related decision - making tasks.
Our objective is to rationally distribute CPU time among the processes. Many processes can be ready for execution at the same time & a single CPU can execute only one process at a time. The OS uses various criteria to select the best process :- The OS programs that implements CPU allocation policies is called the CPU scheduler or CPU manager. The CPU scheduler is a very important component of an OS even though it occupies a tiny fraction of the operating system code. When CPU becomes idle or the system decides to preempt the running process, the system invokes the scheduler to select a different process to run on the CPU. Then the system invokes a context switch (switching of context of 1 prog or other) routine to start the execution of the selected process.

response time
throughput
Turn around time
CPU utilization

→ OS overhead
→ Time consuming (context switch ...)
→ Costly

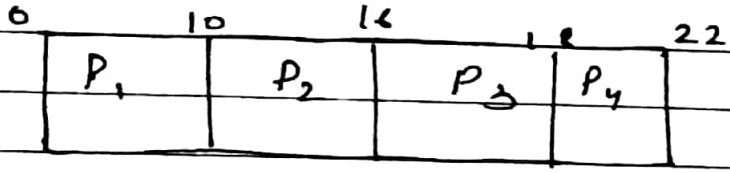
jo presently control ... PCB ... next ... no ho.

1. FCFS Scheduling
It is the simplest of all algorithms. It implements the ready queue as an ordinary first in first out queue.
Process that request a CPU enter at the tail end of the queue. They are de-queued from the head end of the queue. They

Scheduler always allocates an available CPU to the process at the head of the ready queue. Essentially the CPU is allocated processes in earlier of their making request for the CPU. The FCFS algorithm is very simple to implement; the ready queue can be constructed as a linked list of process description.

Process	Arrival Time (AT) ms	Next Burst Time (ms)	Finish Time (FT)	Turn around Time (FT − AT)	Waiting Time (TAT − ET)
P_0	0	10	10	10	0
P_1	1	6	16	15	9
P_2	3	2	18	15	13
P_3	5	4	22	17	13
				57	35

GANTT Chart.



Average (TAT) = $\frac{57}{4}$ = 14.25 ms

Average (WT) = $\frac{35}{4}$ = 8.75 ms

Throughput = $\frac{4}{22}$

Shortest Job first (SJF) (Non-preemptive)

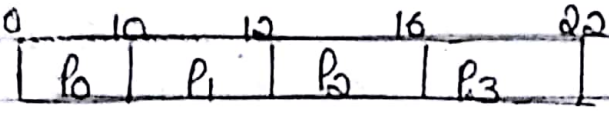
As this scheduling favours processes with shortest execution times, there is a possibility of process starvation happening in such system because shortest process may repeatedly longer once.

- SJF is a non-preemptive CPU scheduling discipline & hence as in FCFS an errorless process can perpetually take over the CPU.

Q.

Process	Arrival Time	Next Burst Time	Finish Time	Turnaround Time	Waiting Time
P_0	0	10	10	10	0
P_1	1	6	22	21	15
P_2	3	2	12	9	7
P_3	5	4	16	11	7
				51	29

* Gantt Chart →



* Average Turnaround Time →

$\Rightarrow \frac{51}{4} = 12.75$ ms.

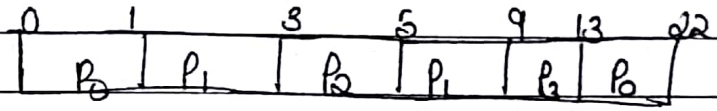
* Avg. Waiting Time →

$\Rightarrow \frac{29}{4} = 7.25$ ms.

SRTN (Shortest Remaining time Next):-

Process	AT	NBT (FT)	Finish Time	TAT	WT
P_0	0	10	22	22	12
P_1	1	6	9	8	2
P_2	3	2	5	2	0
P_3	5	4	13	8	4
				40	18

* Gantt Chart:-



- Avg TAT = 40/4 = 10ms
- Avg WT = 18/4 = 4.5ms.

* It is the preemptive version of SJF. SRTN has the same limitation as that of SJF, namely the scheduler must know the Execution times of processes in advance.

Therefore, from practical point of view it is hard to implement. However it has theoretical approval that it ensures minimal average turn around time.

Continue ...

with

Priority with Non-preemptive

&

Preemptive Algorithm to

SJF

Non preemptive

SRTF

Shortest remaining time first

Preemptive.

Types of scheduling

- Short term
- Middle term
- Long term.

4 Feb

Priority - based Non-Preemptive Algo

Non critical processes are assigned a higher priority than the less critical ones. At the time of scheduling, a process dispatches will be the one that has highest priority amongst the process waiting in the ready queue. Once dispatched, a process P_i is allowed to complete its Next Burst, even if another process of higher priority becomes ready to run during the execution of P_i.

Priority based preemptive algorithm.
At the time of scheduling, a process dispatched will be the one that has highest priority amongst the processes waiting in the ready queue. When a process P_i is executing, if another process P_j of higher priority becomes ready to run during its execution, then P_i will be preempted by P_j.

Non-preemptive

Q

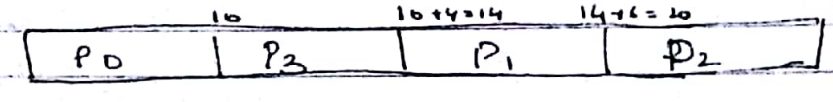
Process	Arrived Time	Next Burst Time (ET)	Priority	Finish Time	Turnaround Time FT-AT
P_0	0	10	5	10	10
P_1	1	6	2	20	19
P_2	3	2	4	22	19
P_3	5	4	0	14	9

$P_3 > P_1 > P_2 > P_0$ Priority order

Waiting Time TAT-ET
0
13
17
5

Gantt chart.

(i) Non preemptive



AVG TAT = $\frac{57}{4}$ = 14.25 ms.

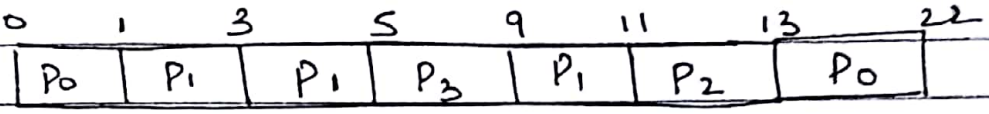
AVG WT = $\frac{35}{4}$ = 8.75 ms.

Throughput = $\frac{4}{22}$.

(ii) By preemptive method.

$P_3 > P_1 > P_2 > P_0$

GANTT Chart



Process	AT	ET	Prio	FT	TAT	WT
P_0	0	10	5	22	22	12
P_1	1	6	2	11	10	4
P_2	3	2	4	13	10	8
P_3	5	4	0	9	04	0
					40	24

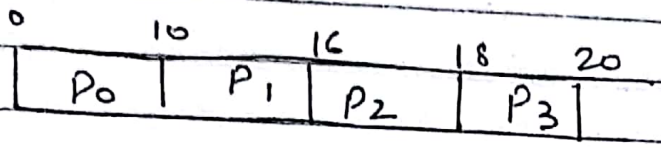
Avg TAT = 40/4 = 11.5 ms.

Avg WT = 24/4 = 6

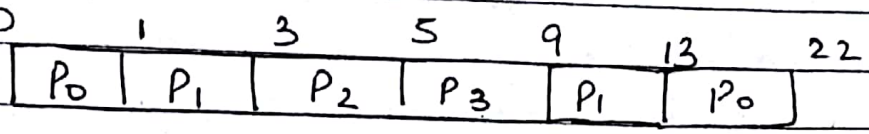
Process	AT	ET	Priority	FT
P_0	0	10	5	20
P_1	1	6	4	16
P_2	3	2	2	13
P_3	5	4	0	9

(1) Non - Preem.

Priority order $P_3 > P_2 > P_1 > P_0$



(2) Pemmptive.



Limitations of Priority Based Algo.

The priority - based algorithm suffer from the inherent possibility of starvation of low priority process. One remedy lies in enhancing the priority of a process in proportion to the time spend by the process in ready queue. This is known as priority adjustment, in relation to the age of a process.
This would exclude the probability of starvation of low priority process, since it ages in the ready queue, its priority will grow, finally enabling it to be dispatched. * to be conti.

Context Switch

✓ Store context of running prog in PCB &
· load the context of another prog's PCB in Reg/ hardware

Each process is modeled by a process descriptor. The execution context of a process is stored in its descriptor. Process context has hardware dependent part. When a process is running on CPU, the process owns some hardware resources such as the CPU register, the memory

PCB

management registers, etc. & the register values (at least those of general purpose registers) are normally different from those stored in the descriptor. Consequently, if the kernel decides to suspend the current process, the latest hardware context value must be saved in the process descriptor. Later when kernel again decides to resume the suspended process, the saved context is reloaded from the descriptor before the process resumes its execution. The task of switching CPU from one process context to another is called a context switch (also known as task switch, process switch & process dispatching)

Each execution of the context switch first saves the hardware context of the currently running process & then restore that of the next process to be executed. The operating system invokes the context switch right after the CPU scheduler selects a different process to run. Normally the scheduler is invoked only when a context switch is anticipated

Limitations of Priority Based Algorithms.

Average context switching overhead of a preemptive algorithm will be higher than average context switching time of the

corresponding non-preemptive algorithm will be lower than its non-preemptive counterpart.

Round Robin (RR) Scheduling

↳ used in interactive time sharing

This scheduling algorithm is specially tailored for interactive time sharing system. A small unit of time, called time-slice or time quantum is defined. The time slice normally varies from 10 to 100 ms. A new process is linked to the tail of ready queue. The program at the head of the ready queue is dispatched. The process is assigned an execution time, equal to specified time slice. Now there are 3 possibilities

(i) The dispatched process completes its execution within the assigned time-slice & terminates.

(ii) The dispatched process requests I/O within the assigned time-slice & is blocked on the suspended queue.

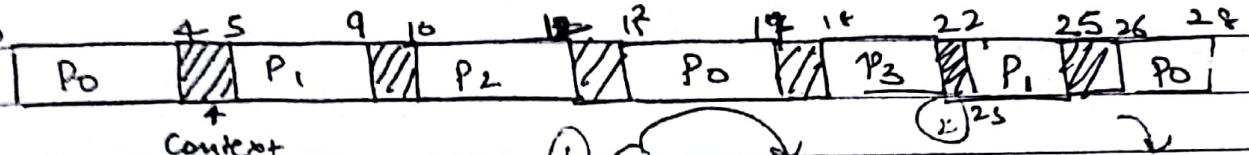
(iii) The dispatched process is not able to finish its execution within the assigned time-slice. At the expiry of time slice, a timer interrupt occurs & process is preempted. It is put at the tail of the ready queue.

Numerical, pto

Arrival Time	Next Burst Time	Finish time	F.T − A.T Turnaround	W.T
0	10	28	28	18
1	6	25	24	18
3	2	12	09	7
5	4	22	17	13
			78	56

Time slice = 4ms

Context switch = 1ms.



Ready que (1) (P0) P1 P2 ⇒ P1 P2 P0 P3

Not P3 ∵ 4 ms task P0 ne aya na.

(2) P2 P0 ~~P3~~ P1 P0

Completed. Now deleted.

$\frac{74}{4}$ $\frac{56}{4}$

RR scheduling contin.

Under all above conditions, the next process from the head of the queue is dispatched for the time slice.

In RR algo., the avg TAT & avg WAT are very large. If time slice is chosen to be longer than the longest possible process, the RR algorithm will become same as FCFS algorithm.

First come first serve

If the time slice is chosen to be small (closer to the context switching period) then context switching overheads will be very high, thus affecting the system throughput adversely. So time slice has to be carefully chosen. It should be small enough to give good response to interactive user. At the same time it should be large enough to keep the context switching overheads low.

Types of Scheduling

- Long term.
- Middle term
- Short-term.

Long term scheduling - LTS refers to the OS decision of admitting (loading into memory & taking up for execution) the pending batch jobs for the batch queue. Batch queue is normally reserved for resource-intensive & low priority jobs that could act as fillers when the interactive activity is low. Each batch job contains programmer specified or system assigned parameters of resource needs in terms of expected execution time, memory, & I/O requirements.

Long term scheduler or job scheduler keeps track of these specifications enable OS to decide their loading order of execution.

The primary goal of long term scheduling is to achieve a proper mix of I/O bounds & CPU bound jobs, so as to keep the CPU & I/O devices as occupied as possible.

M ⇄ H.D ⇒ Swap out
Swap in

Medium term scheduler or swapper - When a process is suspended after making an I/O request, it cannot make any progress till the requested I/O is completed by OS. Sometimes, a suspended process is removed from the main memory & shifted to secondary storage (i.e. swapped out), to make space for another process. Medium term scheduler is responsible for handling of swapped out processes. It determines when a swapped out process can be moved back into the memory (swapping-in) & returned from the same point, where it was swapped out.

Short term scheduling - It allocates the CPU amongst the pool of ready processes, resident in the memory. Assigning the CPU to a process is called 'Process Dispatching'.

↳ Can draw diagram
↳ SJF & SRTF → Can be added here

* * * * * *

- CPU Bound Job - The jobs, that spend comparatively more time performing number crunching operations (i.e., CPU operations) & comparatively less time performing I/O operations are called CPU bound jobs.

- I/O Bound Job - The jobs that spend comparatively more time performing I/O

M $\xrightarrow{\text{se}}$ H.D ⇒ Swap out

Swap in ←

Medium term scheduler or swapper - When a process is suspended after making an I/O request, it cannot make any progress till the requested I/O is completed by OS. Sometimes, a suspended process is removed from the main memory & shifted to secondary storage (Swapped-out), to make space for another process. Medium term scheduler is responsible for handling of swapped out processes. It determines when a swapped out process can be moved back into the memory (swapping-in) & resumed from the same point, where it was swapped out.

Short term scheduling - It allocates the CPU amongst the pool of ready processes, resident in the memory. Assigning the CPU to a process is called 'Process Dispatching'.

↳ can draw diagram

↳ SJF & SRTF → can be added here

* * *

* * *

- CPU Bound Job - The jobs, that spend comparatively more time performing number crunching operations (i.e., CPU operations) & comparatively less time performing I/O operations are called CPU bound jobs.

- I/O Bound Job - The jobs that spend comparatively more time performing I/O

In OS
memory = M.M
seco. mem = disk
called

operations & comparatively less time doing operations are called I/O bound jobs.

Performance Matrics
measurements.

The main objective of a short term scheduler is to offer "good" services to all processes in the system. Many metrics are used to evaluate the goodness requirements.

Throughput is defined as the number of works units completed per unit time. Throughput.
(Same thing done in various units for measuring throughput)

Proportionality & predictability

Proportionality & predictability are the higher level metrics. Proportionality is a measure the amount of expected time relative to its size, & predictability to that of the expected quality of services. It is natural to expect good service to guarantee that smaller jobs stay in the system for a smaller time than larger jobs.

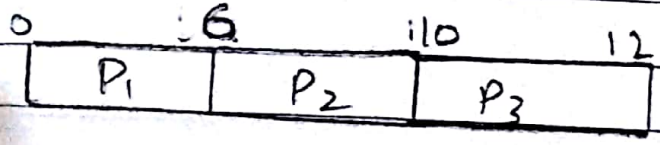
NUMERICAL

Ques.

Process	A.T.	Burst time	Finish	TAT = FT - AT	Wait TAT - BT
P_1	0.0	6	6	6	0
P_2	0.5	4	10	9.5	5.5
P_3	1.0	2	12	11	9
					14.5

Consider the above snap shot of processes & compute average turn around time & average waiting time of process for FCFC, SJF, & SRTN

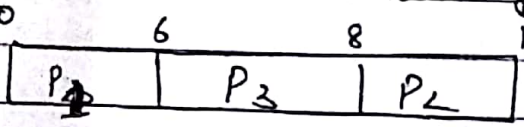
① FCFC



Average TAT = $\frac{26.5}{3}$ = 8.83

Average WT = $\frac{14.5}{3}$ = 4.8.

② SJF (Shortest job first



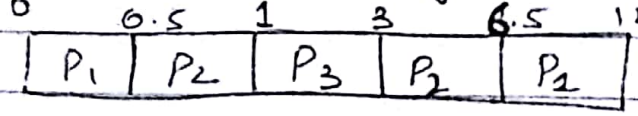
Process	AT	B.T	FT	Turn.	W.T
P_1	0.0	6 5.5	6	6	0
P_2	0.5	4 3.5	12	11.5	7.5
P_3	1	2	8	7	5
				24.5	12.5

Average TAT = $\frac{24.5}{3}$ = 8.1666

Avera WT = $\frac{12.5}{3}$ = 4.1667.

③ SRTN → This is preempti.

Shortest remaining time next.

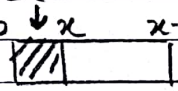


Proces	AT	B-T	FT	Turn	Waiting
P_1	0.0	6	12	12	6
P_2	0.5	4	6.5	6	2
P_3	1	2	3	2	0
				20	8

Average TAT = $\frac{20}{3}$ = 6.667

Average WAT = $\frac{8}{3}$ = 2.667.

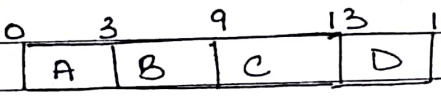
NOTE: If A.T of any process starts with non zero then, ans



Ques

Process	A.T	B.T	F.T	Turn A.	W.T
A	0.0	3	3	3	0
B	1.0001	6	9	7.9999	1.9999
C	4.001	4	13	8.9990	4.999
D	6.001	2	15	8.999	6.999
				28.9979	13.99790

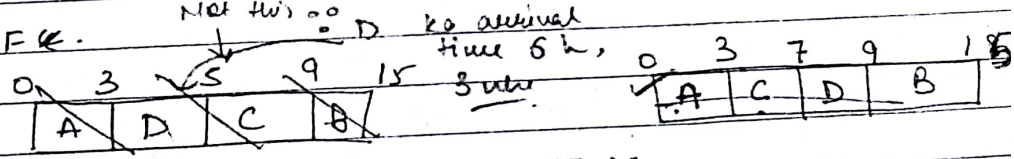
(1) FCFC



Average TAT = $\frac{28.9979}{4}$ = 7.24948

Average WT = $\frac{13.99790}{4}$ = 3.49948

(2) SJFE.



FT	TAT	WT
3	3	0
15	13.9999	7.9999
7	2.999	
9	2.999	

F.T - A.T

F.T	TAT.	W.T
3	3	0
15	13.9990	7.9999
9	4.9990	0.9990
5	-1.001	-3.001

Multilevel Feed Back Queue

* figure also

The major disadvantage with the preceding algorithm is that basically treat all jobs the same. This results in each algorithm favouring certain kind of processes. The Algorithm using Multilevel feedback Queue address this deficiency by customizing the scheduling of a process based on the process's performance characteristics.

Ques Evaluate the avg. waiting time for the following processes using 3-level feedback Queue scheduling, having time quantum 2 units for 1st queue 3 times unit for 2nd queue & FCFS for 3rd queue is used. First queue has highest priority than 2nd queue & 2nd queue has highest priority than 3rd queue all queue are preemptive. All processes enter through 1st queue.

Process id	CPU burst	Arrival
P_1	3	0
P_2	4	1
P_3	2	3
P_4	5	4
P_5	4	5
P_6	6	11
P_7	1	12
P_8	2	14

P_9	3	
P_{10}	4	16
		20

Gant chart.

0	2	4	6	8	10	11	13	14	16	18	20	22	25	27	30	31	33	34
P_1	P_2	P_3	P_4	P_5	P_1	P_6	P_7	P_8	P_9	P_2	P_{10}	P_4	P_5	P_6	P_9	P_{10}	P_6	

RR = 2 unit.

Multi level feedback queue

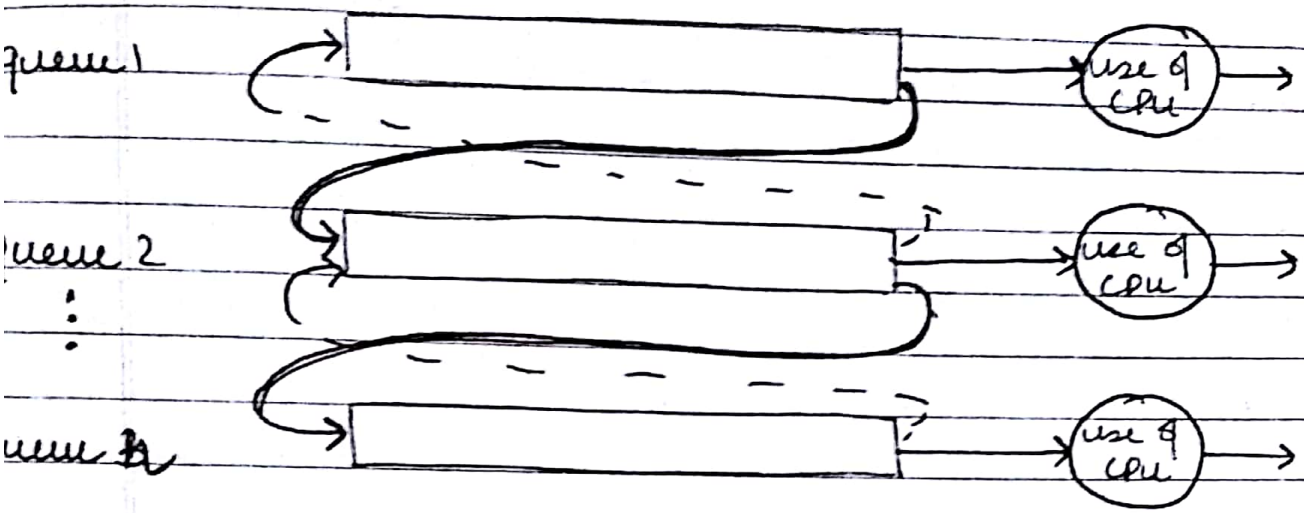


fig – last Page.

Multiprocessor Scheduling

If multiple CPUs are available, load sharing becomes possible; however, the scheduling problem becomes correspondingly more complex.

Asymmetric multiprocessing is simple because only one processor accesses the system data structures, reducing the need for data sharing.

In Symmetric multiprocessing, each processor is self-scheduling. All processes may be in

Cache → Locality of Reference property.

Common ready queue, or each processor may have its own private queue of ready processes. We must ensure that two processors do not choose the same process & that processes are not lost from queue. queue

Multiprocessor

Scheduling

- Symmetric → More diff.
- Asymmetric → less diff.

Multiprocessor scheduling

- Load Balancing
- Migration (transfer work from one local queue to other)
  - Pull migration
  - Push migration.
- Processor Affinity – (Affection of process from processing)
  - Soft affinity: After sometime process can be migrated.
  - Hard affinity: Processor give up to migrate the process.

[eg of worker not willing to leave one job | eg

- Virtualization

Consider what happens to cache memory when a processor has begun running on a specific processor. The data most recently accessed by the process populate the cache for the processor; as a result, successive memory accesses by the process are often satisfied in cache memory. Now consider what happens if the process migrates to another processor.

The content of cache memory must be invalidated for the first processor & cache for the second processor must be repopulated. Because of the high cost of invalidating & repopulating caches, most SMP systems try to avoid migration of process from one processor to another & instead attempt to keep a process running on the same processor. This is known as process affinity.

Load Balancing: On SMP system, it is important to keep the workload balanced among all processor to fully utilize the benefits of having more than one processor. Otherwise, one or more processors may sit idle while other processor having workloads along with list of process