SDS Page No.
Date / /

sums → scheduling algorithm.

* Scheduling Algo. is a way of selecting a process from ready queue and put it on the CPU.
⇒ We try to keep multiple processes in the Ready Queue → present in RAM.
⇒ Now we need to select between these processes & execute on CPU.

Assume we are using UNI processor system (only one processor) we can execute only one process at a time.

Scheduling Algo

Pre-Emptive	Non-Pre-Emptive
→ Shortest Remaining time first	→ FCFS
→ longest Remaining time First	→ Shortest Job first
→ Round Robin	→ longest Job first
→ Priority Based.	→ Highest Response Ratio Next
	→ Multilevel Queue

* Pre-Emptive means if we take out any process from ready queue and put it on Processor and now in running, we can remove this process from processor, put back into ready queue & can execute that process later.

* Non-Preemptive means if we put any process on processor, the entire burst time is completed and then & then only new process is added to Processor.

Non-preemptive → Ready → Running → complete execution.

Page No.
Date

Say if there are 5 processes each taking a variable and different amount to time. And single processor executes in a manner like P_1 → 3sec then P_2 → 2 sec then P_3 → 5 sec then P_4 → 1sec then P_5 → 4 sec.

→ we are completely executing a process and then moving to a new process ⇒ non-preemptive.

But in preemptive, say 5 processes are present & each takes 5ms. So we start with P_1 → 2msec, then move to P_2 → 2ms .... if we only complete task of 2ms and we move ahead putting the previous task back to ready queue and then again utilizing it to processor. Reason behind this may be Time quantum. Adv. is the responsiveness.

Other reason is prority, if any high prority process comes, then we do ~~not~~ have many gaps to rush in the prority process.
And if one process takes long but other takes short. both are equally treated.

12:03 # Arrival time ⇒ Point of time when a process enters ready queue

3 min # (Duration) Burst Time ⇒ Time duration required by a process to get executed on CPU.

Completion Time ⇒ Point of time at which the process gets executed (end time)

12:08

2 min buffer → waiting time

In Non-Preemptive ⇒ RT == WT

SDS | Page No. | Date / /

12:08 - 12:03 = 0.5

* **Turn Around Time:** [completion time − Arrival time]
  ↳ total time taken.

5 - 3 = 2.

* **Waiting Time:** [Turn around time − Burst Time]
  ↳ buffer or wait [a queue]

* * **Response Time:** [(the time at which a process gets CPU 1st time) − (arrival time)]

⇒ entry = 12:03 & CPU = 12:04 = 12:04 − 12:03 = 0:1 min.

* Processes are of 2 types CPU time → already tell the time of execution (duration)
  & I/O time → that have on no. of diff. processes that require I/O that adds up waiting time.

First come First Serve: [FCFS]

Q]

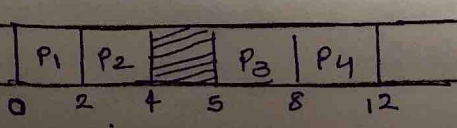
Process no.	Arrival Time	Burst Time (duration)	Completion Time	TAT [CT − AT]	WT [TAT − BT]	RT. [1st CT − AT]
P_1	0	2	2	2	0	0−0=0
P_2	1	2	4	3	1	2−1=1
P_3	5	3	8	3	0	5−5=0
P_4	6	4	12	6	2	8−6=2

Criteria → 'Arrival Time' → as FCFS is based on provide service to one who came first which is based on time of arrival.

Mode → Non-Preemptive (complete execution)

Avg TAT = (2+3+2+6) / total process = 14/4 = 3.5

Gantt Chart ⇒



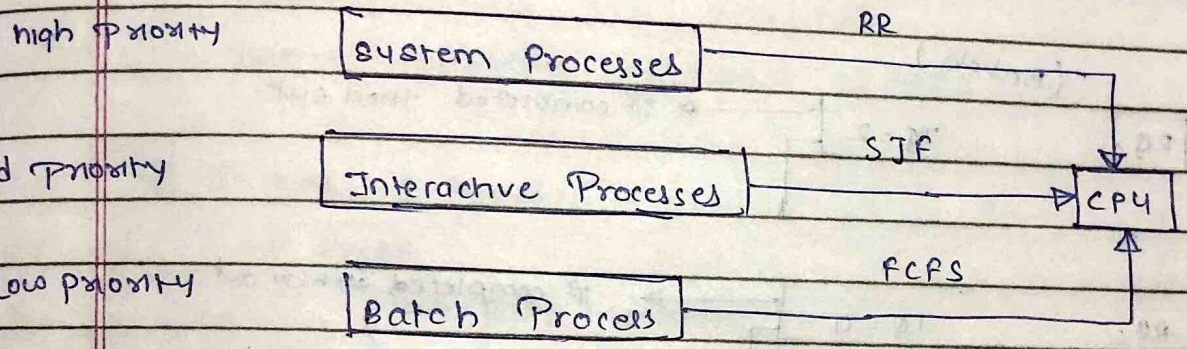
→ Time

By P_1 ends P_2 has already arrived (using arrival time) } P_2 suffers waiting time = 1

Multi Queue Processor Scheduling: Multilevel

⇒ Till now we only take one ready queue and execute them based on a scheduling algo.

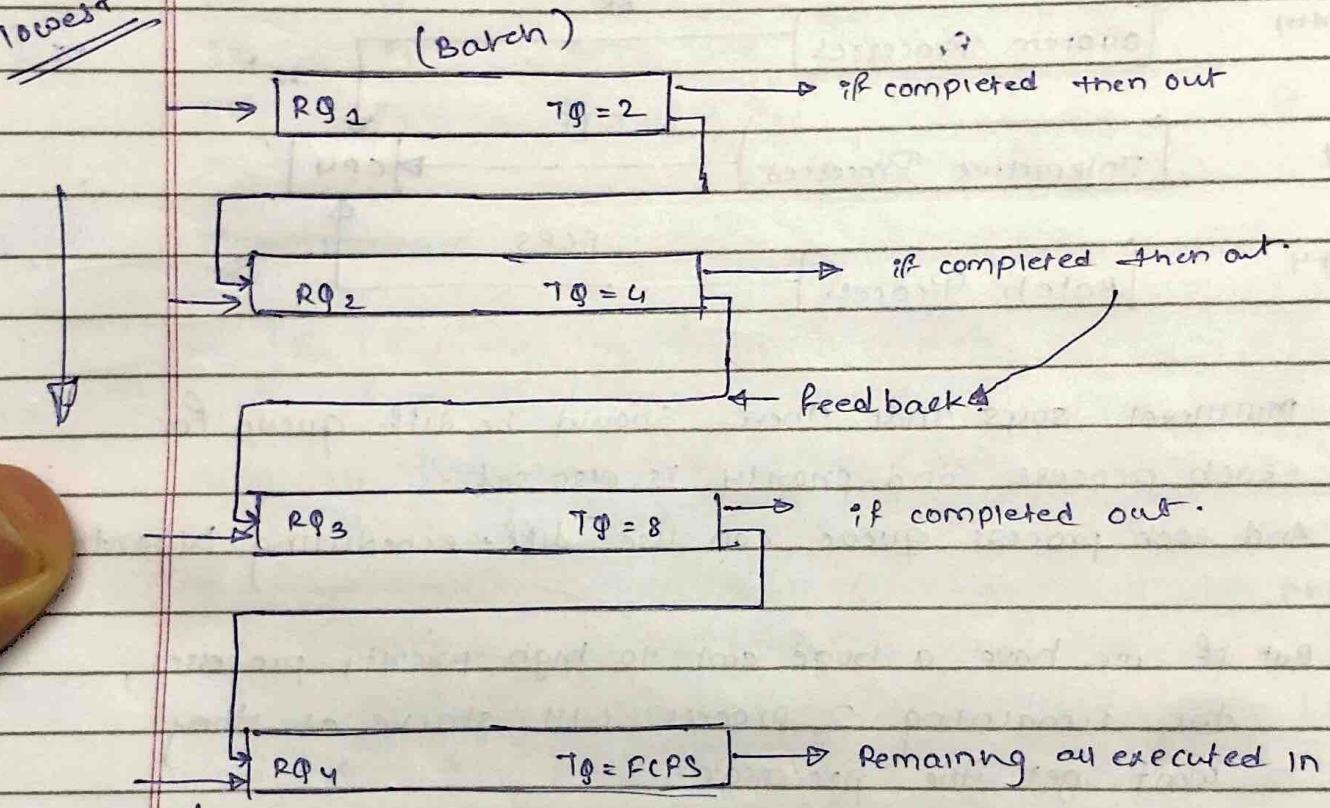
∴ Multilevel scheduling have 3 types of processes.



- Multilevel says that there should be diff queue for each process and prority is also set.
- And each process queue can use diff. scheduling algorithms.
- But if we have a huge amt to high prority processes, the remaining 2 process will starve as they won't get the processor.
- For this issue, Multi-Level Feedback queue is used.

Multi-level Feedback Queue:

- in multi-level queue, starvation issue is seen.
- But here we can solve this by using feedback from low priority processes and take them to processors.



∴ lowest priority Processes run & provide feed back, now if any high priority proces comes, it moves to FCFS & executes (no feed back required).

P_1 (batch) = 19 → 19 - 2 = 17 → 17 - 4 → 13 → 13 - 8 = (5)
goes to FCFS or final stage.

SDS Page No. Date / /

Process Management :

Process Concept :

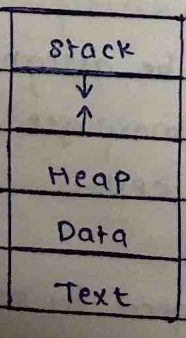
→ A process is an active running software or program on our OS.

→ A process is an 'active' entity that has accessed RAM, hardware & the CPU for his accomplishment where as program is sort of a 'passive' entity that runs for a short period of time.

→ The status of current activity of process is represented by value of program counters and contents of process registers.

→ We create computer programs as txt. files, when executed, create process for all tasks listed.

⇒ When a program is loaded to the memory, it break downs into 4 parts → stack, heap, text & data to form a process.



- Text ⇒ A process, at some time known as Text, includes current activity represented by Program Counter.
- Stack ⇒ contain temp. data, like funct. parameters, return add., local variables.
- Data ⇒ contains global var.
- Heap ⇒ dynamically allocated memory to process during runtime.

* context switching occurs only in kernel mode.
->context sw. time is a overhead

Page No.
Date / /

* Context Switching:
⇒ Process of saving the context of one process and loading another process.
⇒ Basically storing data of loading & unloading the process.
⇒ Generally occurs when high priority data comes in to ready state or interupt occurs, or switch to kernel mode, or PreEmptive scheduling.

* Processes are of 2 types

a) CPU Bound ⇒ requires more running time (CPU time). and generally do not stop, they complete their entire execution.

b) I/O bound ⇒ while running, they might stop and perform some other I/O task.

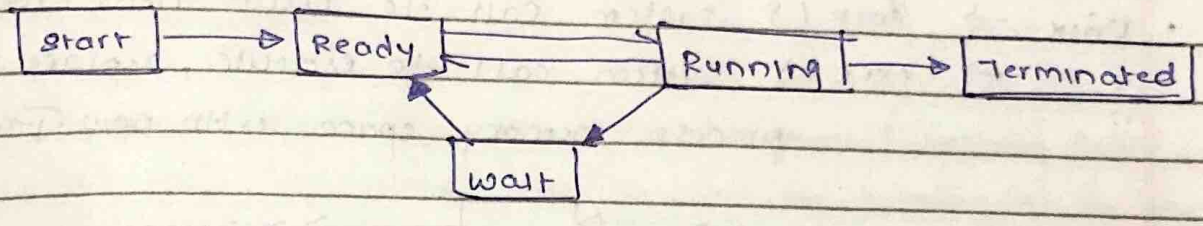
* States of a process
a) New state ⇒ initial state when the process is created / started.

b) Ready State ⇒ The process is waiting to be assigned a processor so that they can run. It may get interuppted by scheduler to assign CPU.

c) Running state ⇒ once processor is assigned by scheduler, the processor executes the process.

d) Waiting ⇒ If process needs any resources it goes into waiting state, & comes to queue again.

e) Termination ⇒ on finishing the execution, process moves to termination & waits to be removed from main memory.



Process Creation:

→ A process can be created in a system for different operations. and also if any ~~para~~ parent process ~~is~~ is running, it may create a new child process using fork().

↳ A child can have only one parent process but a parent may have multiple child processes.

→ When a new process is created the O.S assigns a unique PID and inserts a new entry in process table.

→ Later the space required for process like stack, PCB etc. is allocated and various values present in PCB are initialized. like PID value, then process register values, stack ptr & program counter ptr values.

* The priority will be lowest by default. and then the process will undergo its states.

→ Resource sharing : Parent & children share all resources.
: children share subset of parent resources.
: Parent & children share no resources.

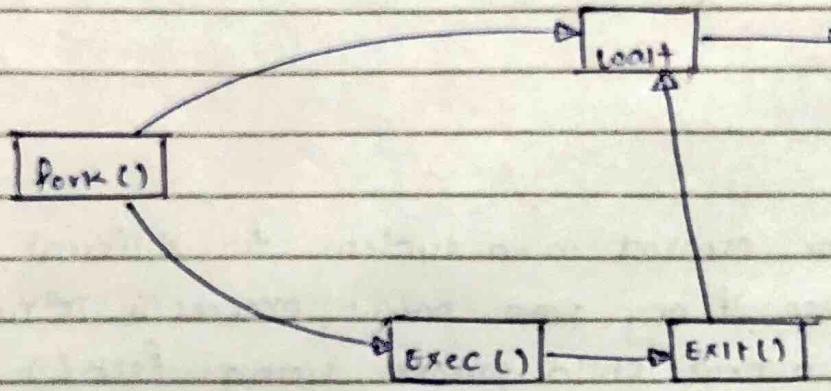
→ Parent & children execute concurrently. } Execution.

→ Parent waits till its children terminates

→ child is duplicate of Parent } address space

→ child has Program (another (new)) loaded to it

- Unix → fork() system call to create new Process.
  → exec() system call to execute, replace the process memory space with new Program.



Process Termination:

→ When the system call ⇒ exit() is called, the process is terminated.
Else, processes are terminated themselves when they finish execution.
→ The OS uses Exit() to delete all the context of the process. And all resources held by process are taken back by the O.S. like virtual memory, files, buffer, etc.

A parent may terminate a Process bcoz of
a) task given to child is not required rn.
b) child takes more resources than limit.
c) parent itself is exiting, child is also deleted.
↳ cascading termination

Process Suspension :

⇒ If any process are put into blocked / waiting state, the O.S. suspends the process by putting it in the suspended state - & transferring the process back to disk.

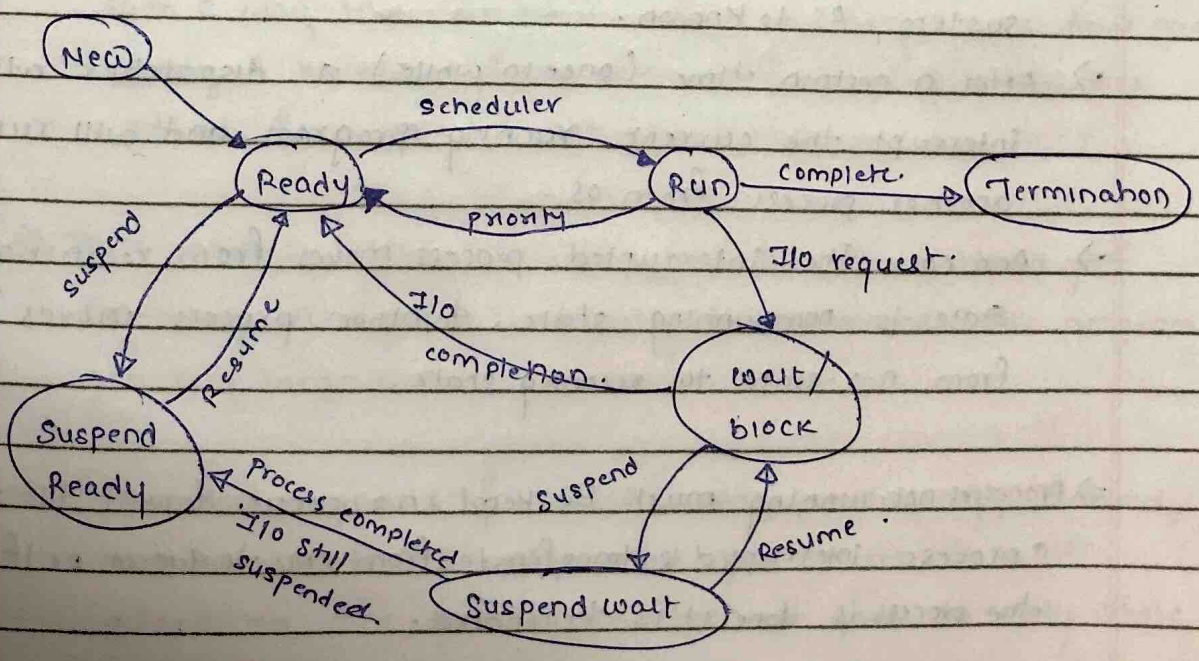
⇒ Along with ready, running, blocked a suspended state is also used in OS.

⇒ Thus if process is ready to run from suspend or blocked state we have seperate ready queue for them.

⇒ It may occur if any interupt arrives or high prority task arrives. OR if the process takes too long for execution.

* Cascading Terminaton: If parent process terminates, all its child process also terminate.

⇒ process states withe suspension.



New ⇒ newly created process

Ready ⇒ ready for execution

Run ⇒ running state inside CPU

Wait ⇒ requested I/O

Terminated ⇒ end complete execution.

Suspend ready ⇒ if ready queue is full ~~or I/O is only remaining.~~

Suspent wait ⇒ wait queue is full.

Two State Process Model :

⇒ As name suggests, it only includes 2 states, one is Running & another is not-running.

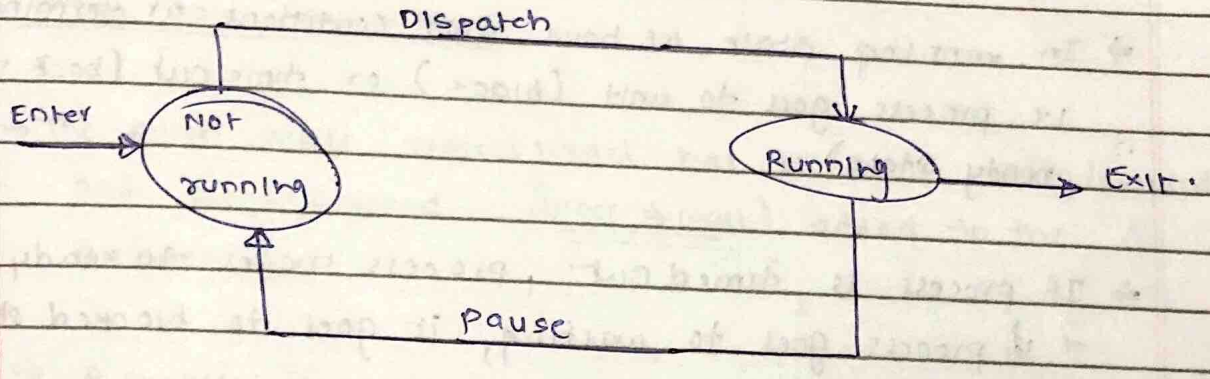
- Running ⇒ state in which process is currently executed.
- not-running ⇒ in which process is waiting for execution.

⇒ Firstly when process is created a PCB is also created, and allowed all the data and thus process can enter the non-running state. If any processes exits the system, OS is known.

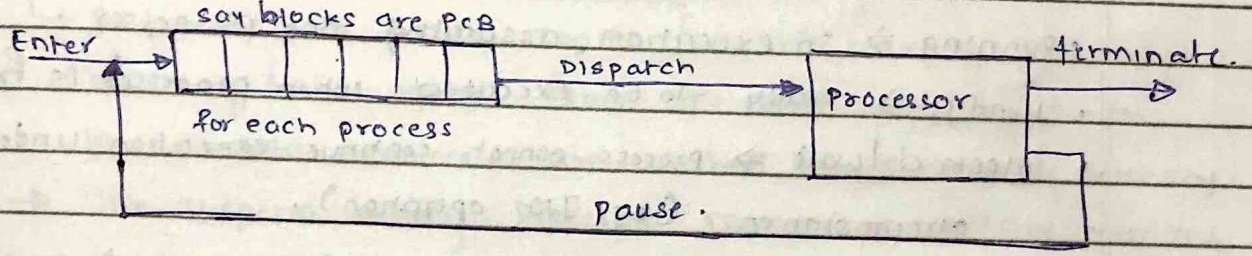
⇒ After a certain time (once in while) a dispatcher will interrupt the current running program and will run another process from OS.

⇒ And now the interrupted process moves from running state to non-running state, & other process moves from n-r. state to running state.

⇒ Processes not running must be kept in a queue & wait till their turn. a process interrupted is transfered from CPU to queue or if the process is done it is terminated.



using queues:



Five State Process Model:

⇒ In a five state model, the states have been split in such a way like we have 2 non-running states that are ready & blocked. Along with this 2 more states are added for New & Exit/Terminate.

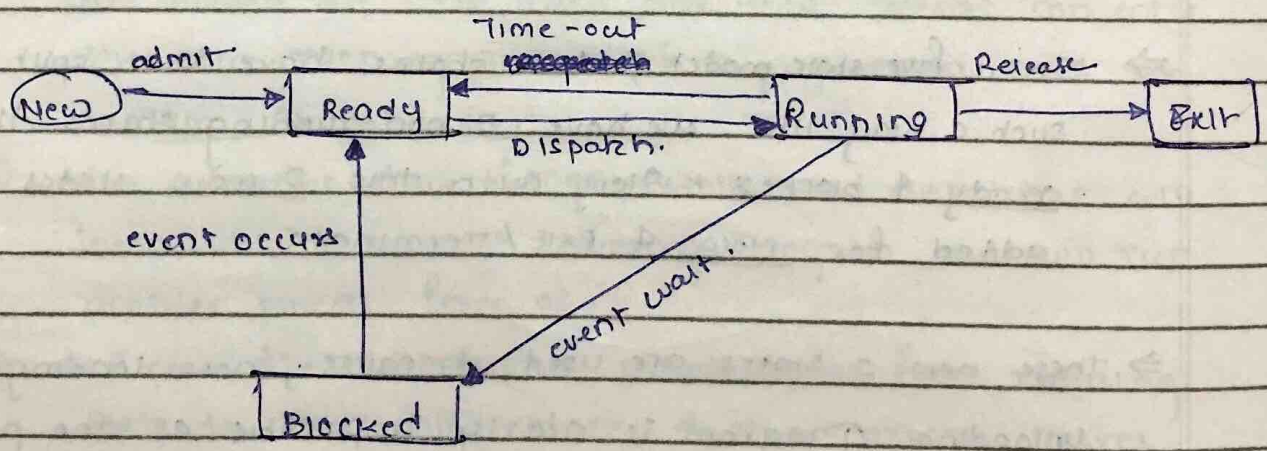
⇒ These new 2 states are used because, now loading & unloading Program is nearly impossible as the programs are very large.

⇒ This model consists of New, ready, running, blocked, and exit state. If any new job occurs it is 1st added to the queue and then it goes to ready state. After which the process goes to running state.

⇒ In running state we have two conditions or terminate, i.e. process goes to wait (block) or time out (back to ready state).

⇒ If process is timed out, process moves to ready state
If process goes to waiting, it goes to blocked state and then to ready state.
If no issue occurs, process gets executed.

- Running ⇒ in execution, assuming one processor.
- Ready ⇒ ready to be executed when processor is free.
- Blocked / wait ⇒ process cannot continue execution under some circumstance (eg: I/O operation).
- New ⇒ New process has been created, but not yet ready for execution & is not loaded to main memory.
- Exit / Terminate ⇒ success full execution. / released by OS.



- If process terminated, data is not preserved by OS.
- CPU may stay idle if processes go to block state.

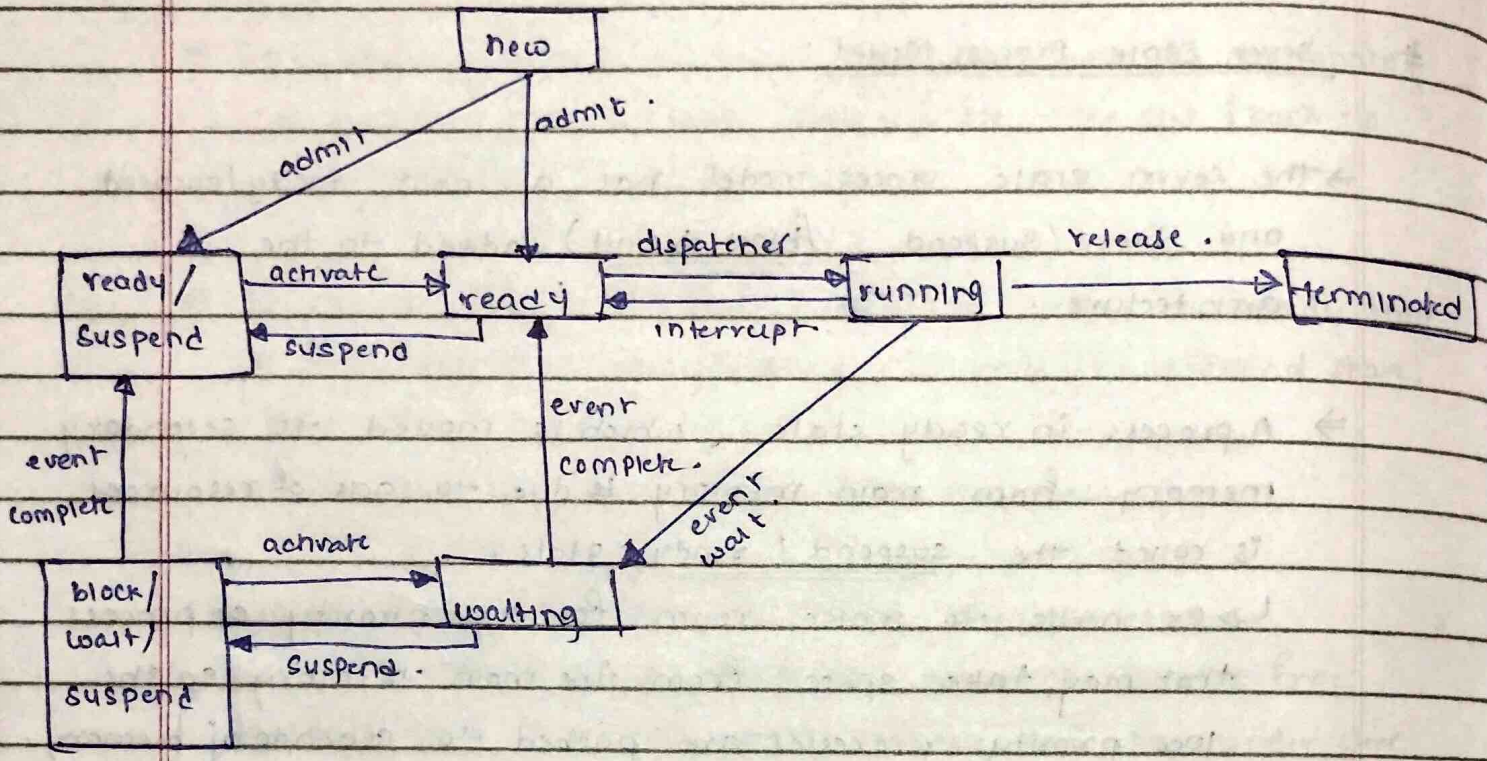
Seven State Process Model:

→ The seven state process model has a new ready/suspend and block/suspend (block → wait) added to the architecture.

⇒ A process in ready state, which is moved to secondary memory from main memory due to lack of resources is called the suspend / ready state.
↳ Os needs to make room for high prority process that may take space from the main memory so the low prority processes are pushed to secondary memory.
→ The suspend ready processes remain in 2ndary memory until main memory is freed.

⇒ Instead of removing the process from ready queue, it is better to remove blocked process that are waiting for resources in main memory. [suspend wait] state.
⇒ Thus they can wait in 2ndary memory to make space for high prory processes in main mrmory.
Once main memory is available they complete their process.

- New → newly created processes.
- Ready → ready for execution, waiting for CPU.
- Running → in CPU for execution.
- Exit → completion, OS release.
- Block/wait → Processes in main memory waiting for execution.
- Boalt Suspend → contains process present in 2nd ary memory.
- Ready Suspend → contains process present in 2nd ary mem. but ready for execution. as soon as they are loaded to main memory.



Process Description

⇒ OS controls & maintains tables of info about each entity like memory tables, IO tables, file tables, process tables.

⇒ Manages the data regarding current status of each process and resources used by it.

* Memory Tables: (what is memory)
  - keeps the track of which part of memory is used and by which process.
  - decide which processes are to be loaded in the memory.
  - Allocate & deallocate memory.

* Device tables: hide peculiarities of hardware devices.
  - Buffer caching system.
  - General device driver code.
  - Drivers for hardware.

SDS | Page No.
Date / /

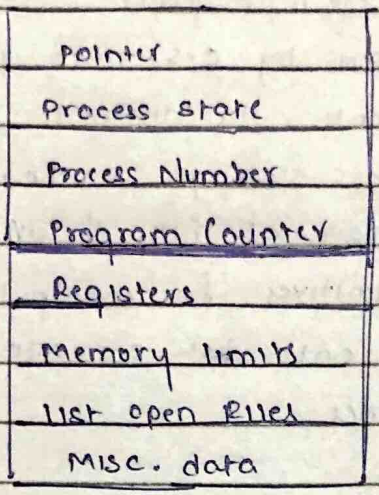
* File Tables : (File management) | File types | software | directories.
→ Files are mapped by O.S. onto physical device.
→ The OS implements the abstract concept of file by managing mass storeage device.
- Creation & deletion of files & directories.
- support of primitives for manipulating files & dir.
- mapping files onto disk storeage
- Backups of files.

* Processes : (what is process, program, execution).
→ A process is a unit of work in a system, where this system consist of collection of processes.
- creation & deletion of user & system processes.
- Suspension & resume of processes.
- mechanism for process synchronization.
- mechanism for deadlock handling.

Process Control Block (PCB).

⇒ while creating a process the O.S. performs several operations. To identify a process, it assigns a PID to each process. An OS needs to keep a track of all the processes.
⇒ To maintain track of all processes PCB is created, it is used to monitor the execution status of process.
⇒ When a program makes a transition from one state to another, the OS system must update info. in the PCB.

Diagram



Process Control Block.

- Pointer → Stack pointer, saved data regarding when process is switched just to retain previous state current pos.
- Process State → Stores the resp. state of process.
- Process No. → PID is process No.
- Program Counter → stores ~~pointer~~ counter that stores ~~data~~ data/count to next instruction ~~that~~ is to be executed.
- Register → CPU registers, includes accumulator, base, general purpose etc.
- Memory limits → Info. about memory management. Page tables, memory tables etc.
- Open file list → list of files used ~~fo~~ by process.

⇒ PCB also contains info about interupts that a process may have generated & how OS handles them.

⇒ PCB plays a crucial role in context switching.

⇒ Real time systems may require more info in PCB such as deadlines, priorities, etc.

⇒ PCB has info about processe's virtual memory management & also facilitates inter communicaton. & info about shared resources.

⇒ PCB helps in fault tolerence to the OS.

⇒ overhead is main issue in the PCB as data is written to PCB at that time the os is idle.

⇒ complexity, scalability & security are issues.

Page No.
Date / /

Threads:

⇒ A thread is a subset of process also known as lightweight process.

→ A process can have more than one thread and are managed independently by scheduler. and all threads are inter-related to each other.

→ Threads are mainly used to improve the processing of an applicanon. In reality only one thread is executed at a time but context switching is very fast.

⇒ The thread uses the memore of the processor. i.e. shared memory. [Stack space, Register Set, Program Counter]

Multi-Threading

⇒ A process of multiple threads executing at the same time.

* There are Two threading Models that are user-level threads and kernel level threads.

* User-level Threads:

→ In this type the Thread is not created using system calls.

→ The kernel has no work in management of user-level threads.

→ User-level threads can be easily implemented by the user.

→ When user-level threads are single-handed process then, managed by kernel-level thread manages them.

* Adv (user-level)

→ implementation of U-L-T is easier than K.L.T.
→ context switching time is less.
→ More efficient then K.L.T.
→ only consist of Program Counter, Register Set, and stack space.

* disadv (ULT)

→ lack of coordination betn Thread & kernel.
→ whole process gets block if any thread cause issue.

* Kernel - Level Threads :

→ An thread that can recognise the O.S. easily.
→ Kernel level T. have their own thread table where it can keep track of system.
→ The OS. kernel helps in managing K.L.T.
→ K.L.T. have more context switching Time.

* adv (KTL)

→ up-to-date info of all threads.
→ when any process require more time to process, kernel-level thread provides more time to it

* disadv (KTL)

→ slower than ULT.
→ implementation is ~~tes~~ more complex.

*	ULT	KLT.
	• ULT are implemented by Users	• KLT are implemented by O.S.
	• O.S. doesnt recognise ULT	• KLT are recognised by OS.
	• Eeasy implementation	• Complex implementation.
	• less Context Switching time	• High Context Switching Time.
	• If one ULT performs blocking operation, entire process will get blocked.	• If one kernel T. performs a blocking operation, then another thread can perform/ continue execution.
	• ULT cannot be multi-Threaded	• KLT can be multi Threaded.
	• Easy to create & manage.	• More time to create & manage.
	• ULT has its own stack, but they share same add. space	• KLT have their own stack & their own seperate addr. space thus better isolation.
	• ULT crashes, entire process is ~~dra~~ down thus less fault Tolerence	• KLT are more independent, thus one's crash may not affect other.
	• ULT dont talke full adv. of system resources. as they dont have access	• It can access system features

SDS | Page No.
Date / /

* MultiThreading can help improve efficiency of CPU, responsiveness in app., better resource utilization.

* Multi-Threading Models:

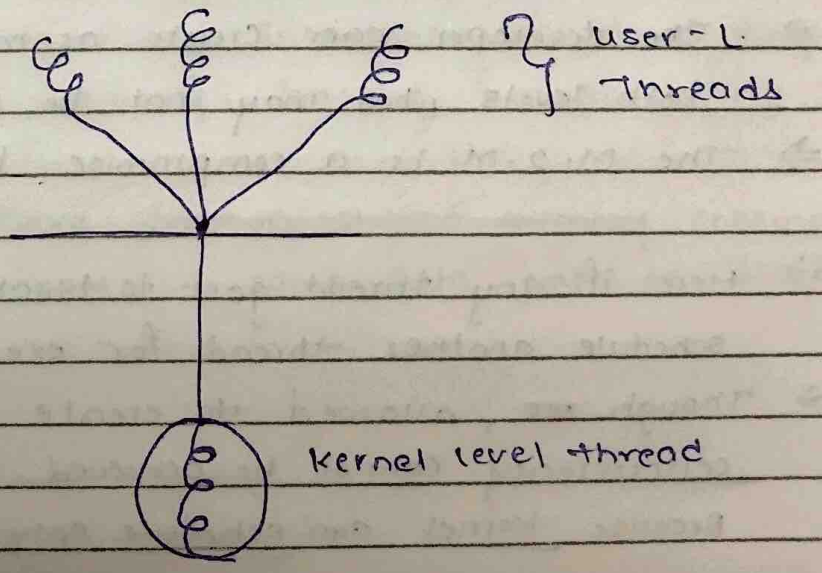
⇒ Multi Threading allows to divide a process into multiple threads. In multi Thread, more than one thread perform the same task.

a) Many to one (M.T.1 model).

⇒ Many ULT. mapped to one kernel Thread. Facilitating in better context switching and are easy to implement

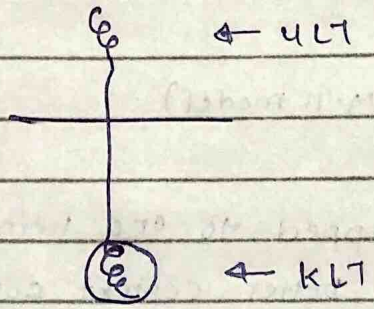
⇒ since there is only one-kernel level thread schedule at a given time, we cannot take adv. of hardware acceleration offered by multi-thread processors.

⇒ if blocking comes, the model blocks the entire process.



b) One to one

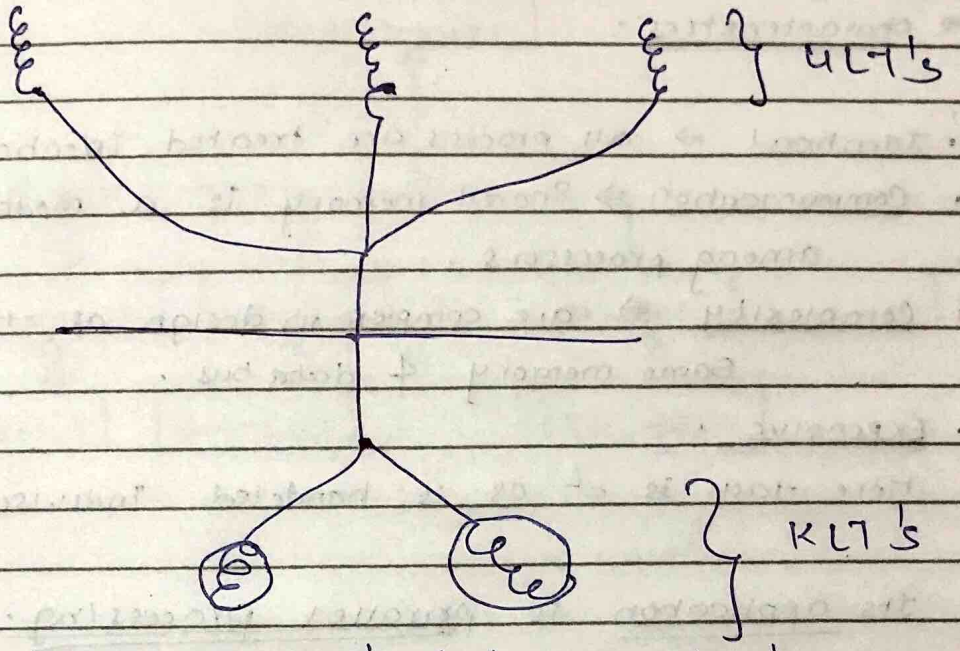
⇒ A single ULT is mapped to a single KLT.
⇒ This helps in running multiple threads in parallel.
⇒ The generation of every new ULT must include creating a corresponding kernel Thread causing overhead.



c) Many to many

⇒ Here we have several user-level threads and several KLT.
⇒ The no. of KLT is dependent on particular application.
⇒ The developer can create as many threads at both levels, but may not be same.
⇒ The M:2:M is a compromise beth other 2 models.

⇒ Here if any thread goes to block, kernel can call/schedule another thread for execution.
⇒ Though we, allowed to create multiple KT, true concurrency cannot be achieved.
Because kernel can schedule only one process at a time.



* It can be several ULT's but less KLT's.

Symmetric MultiProcessing: (SMP).

⇒ MP. involves computers hardware & software architecture where there are multiple processing units executing programs for single OS:

⇒ SMP refers to computer architecture, where multiple identical processes are interconnected to a single shared main memory, having full access to I/O devices ~~HHE~~.

⇒ All process have ~~common shared memory~~ common shared memory & same data path or I/O bus.

* characteristics :

- Identical ⇒ all process are treated identically.
- Communication ⇒ Shared memory is a mode of communication among processors.
- Complexity ⇒ are complex in design as they share same memory & data bus.
- Expensive.
- Here task of os is handeled individual processors.

⇒ Its application is parallel processing. where time sharing system have assigned task to diff. processors running in parallel to each other, also uses multi Threading.

* adv

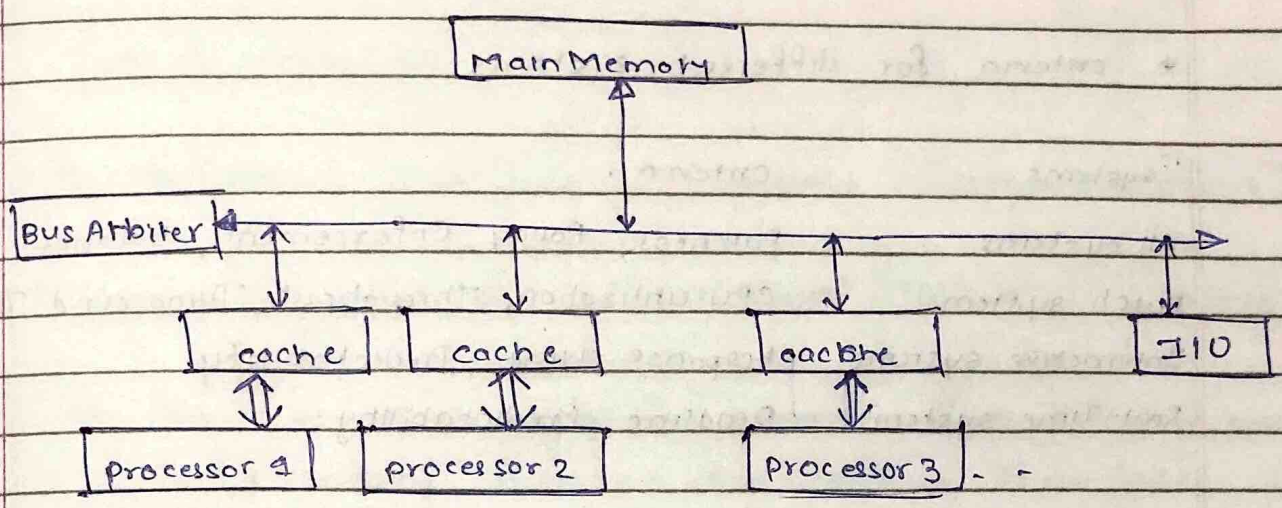
↳ since task can be run by all processors, throughput value increases.

→ Failing of a processor doesn't fails system, as all are capable of processing.

* disadv

→ since all processors are treated equally by os, designings becomes difficult.

→ comman main memory causes more size tending to be costlier.



2.3

Scheduling Criteria :

Scheduling of ~~user~~ processes ~~is done~~ that allows one process to use the CPU while another process is in standby due to unavailability of any resources such as I/O etc, thus making full use of CPU.

* There are a bunch of scheduling criteria that we need to consider when we are choosing scheduling algo.

→ CPU utilization
→ Throughput
→ Turnaround time
→ Waiting time
→ Response time
→ Deadline
→ Fairness
→ Predictability
→ Policy Enforcement
→ Balance.

* criteria for different systems:

systems	criteria.
All systems	Fairness, Policy Enforcement, Balance
Batch systems	CPU utilization; Throughput, Turnaround Time.
Interactive systems	Response Time, Predictability
Real-Time systems	Deadline, Predictability.

* User Oriented, Performance Related:
  - response time
  - turn around time → time pt. of submission to results completed (en
  - deadline → maximize percentage of deadlines met.

* User Oriented, other:
  - Predictability ⇒ a job should run the same regardless of the load.

* System Oriented, Performance Related:
  - throughput → amt. of processes completed per time unit.
  - processor utilization → ratio of time processor is busy.

* System Oriented, other:
  - fairness → no process should starve.
  - enforcement of priorities → scheduling policy should favour processes of higher priority.
  - balancing resources → keep resources busy.

Types of scheduling:

⇒ Pre Emptive and Non-Pre Emptive. [explained earlier].

* Long Term scheduler: Job scheduler / Fast / absent in sharing system

→ This involves selecting the process from storeage pool in secondary memory and loading them into ready queue in main memory for execution. [Handelled by long term scheduler].

→ The L.T.S. handles / manages the degree of multiprogramming. It must select a careful mixture of I/O bound and CPU bound processes to yeild max throughput
If it selects too many CPU bound processes then the I/O devices are idle & vice versa.

* Short Term Scheduling: CPU scheduler / very fast / min. in time sharing system.

→ short term scheduling involves selecting one process from ready queue & scheduling them for execution. A scheduling algorithm is used to decide which process will be scheduled to execute next.

→ The short term scheduler executes much more frequently than long-term as a process may execute for few milliseconds.

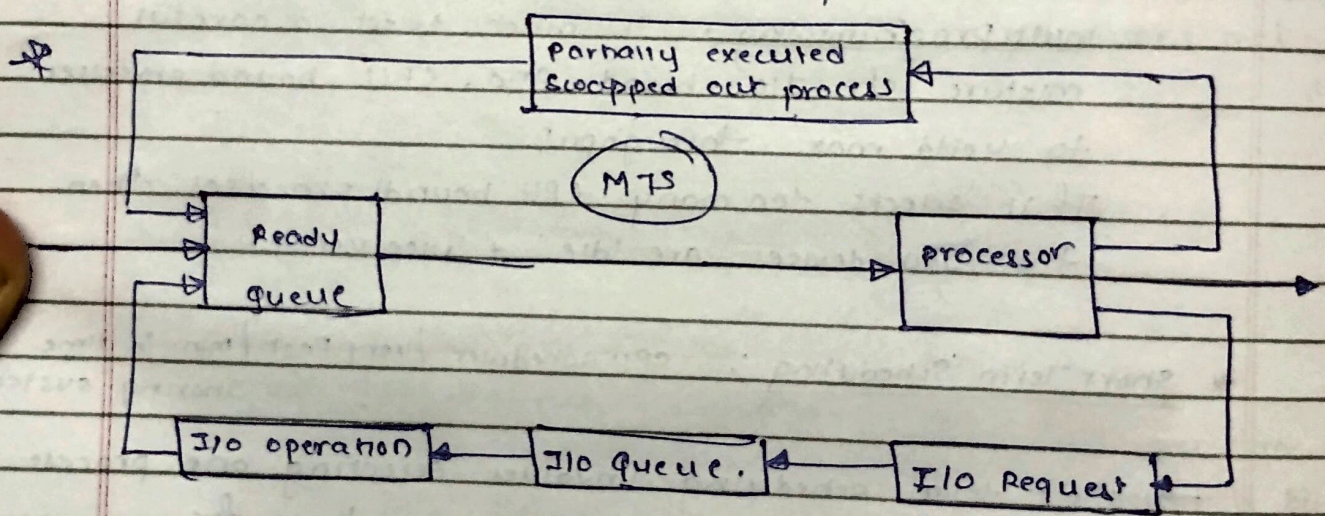
→ The choices of S.T.S are very imp, because if it selects a process with huge Burst Time, then all processes after that will have to wait for long time.

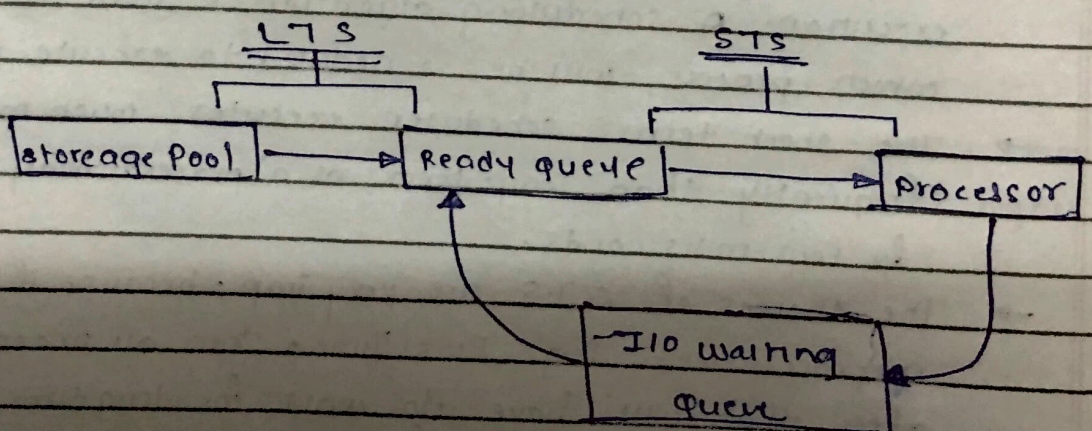
⇒ This is known as starvation & may happen if decision are taken wrong.

* Medium Term Scheduling:

- MTS involves swapping out process from main memory. The process can be swapped in later from the point it stops executing.
This is called as suspending & resuming of process and is done using MTS.

- Helpfull in reducing the degree of multiprogramming. Swapping is also usefull to improve mix of IIO & CPU bound processes in memory.





SDS Page No. Date / /

Classification of multiprocessor system:

* loosely coupled or distributed multiprocessor, or cluster
  → consists of a collection of relatively autonomous systems, each processor having its own main memory & I/O channels.

* Functionally specialized processor:
  → There is a master, general purpose processor, specialized processors are controlled by the master processor & provided services to it.

* Tightly coupled ~~multi~~ multiprocessor:
  → Consists of a set of processor that share a common main memory & are under the integrated control of an OS.

Granularity

Grain	Description	Sync. Interval (instructions)
Fine	Parallelism inherent in a single instruction stream	< 20
Medium	Parallel processing or multitasking within a single app.	20 - 200
coarse	Multiprocessing of concurrent processes in a multiprograming env.	200 - 2000
Very coarse	Distributed processing across network nodes to form a single computing env.	2000 - 1M.
Independent	Multiple unrelated processes.	not-applicable

* Independent parallelism:

- No explicit synchronisation among processes
  → each of them represents a seperate independent app. or job.
- used in time-sharing system.

  ⇒ each user is performing a particular applicatn
  ⇒ multiprocessor provided the same service as a multiprogrammed uniprocessor.
  ⇒ as more than one processor is available, avg response time is less.

* Coarse & very coarse Grained Parallelism:

- synchronization among processes, but at very gross level.
- Good for concurrent processes that are running on multiprogrammed uniprocessor.

  → can be supported on a multiprocessor with little or no change to user software.

* Medium Grained Parallelism:

- Single app. can be effectively implemented as a collection of threads within a single processor.

  → programmer must specify the potential parallelism of an application.

→ there needs to be a high degree of co-ordination and interactions among the thread of applications, leading to a medium-grain level of sync.

- Because of various threads of an application interact so frequently, scheduling decisions concerning one thread may affect performance of entire app.

* Fine-Grained Parallelism:

- Represents a much more complex use of parallelism found in the use of threads.
- It is specialized and fragmented area with many diff. approaches.

Design Issues:

* Scheduling on a multiprocessor involves ~~three~~ three issues, and the approach taken will depend on degree of granularity of application and no. of processors available.

→ Issues:

- actual dispatching of a process
- Use of multiprogramming on uniprocessors.
- assignment of processes to processors.

Assignment of process to processors:

dynamic / static

* assumming all the processes are equal, it is simplest to treat processors as a pooled resource & assign process to processors on demand.

} static or dynamic needs to be determined.

static

* If a process is permenently assigned to one processor, from activation until its completion, then a dedicated short Term Queue is maintained for each processor

} advantage is that there may be less overhead in scheduling function.

allows the grp or gang scheduling.

* a disadv. of static assigment is that one processor can be idle with an empty queue, while another processor has backlog.

* Both dynamic & static methods require some way to assign process to a processor.

a) Master / Slave approach.

→ Key kernel function runs on a particular processor.
→ Master is responsible for scheduling
→ slave sends service requests to master.
→ It is simple and it requires little enhancement to a uniprocessor multiprogramming OS.
→ conflict resolution is simplified as one processor has control of all memory & I/O resources.

• disadv → failure of master brings down entire system
→ master can become a performance bottleneck.

b) Peer Architecture :

→ Kernel can execute on any processor.
→ Each processor does self-scheduling from pool of available processes.

disadv • it complicates the OS.
• OS must ensure that it does not chooses the same process and that the processes are not lost from the queue.

Process Scheduling:

• usually processes are not dedicated to a processor
• A single queue is used for all processes:
→ if some sort of priority scheme is used, there are multiple queues based on priorities.
• System is viewed as being a multi-server queuing arch.

Thread scheduling:

• Thread execution is seperated from the rest of processes.
• An app. can be a set of threads that cooperate and execute concurrently in the same address space.
• On a uniprocessor, threads can be used as program structuring aid & to overlap I/O with processing.

- In multiprocessor, threads can be used to exploit true parallelism in an application.
- Dramatic gain in performance are seen in multiprocessor system.
- Small differences in thread management & scheduling can have an impact on application that require significant interaction among threads.

Types

1) Load sharing (Processes are not assigned to a particular processor

⇒ simplest approach that comes over most directly from uniprocessor env.

⇒ FCFS, smallest no. of threads first, & PreEmptive smallest no. of threads first, are some versions.

⇒ Adv :
- load is distributed evenly across processors.
- no centralized scheduler required.
- global queue can be organised & accessed using any of the ~~queue~~ schemes.

⇒ disadv
- Central queue occupies a region of memory that must be accessed ing manner that enforces mutual exclusion betn process, thus can lead to bottleneck.

- Preemptive threads are unlikely to resume execution on same processor, thus caching can be less efficient.
- If all the threads are treated as a pool of threads, it is unlikely that all threads of a program will get processor at a time. It may compromise performance.

2) Gang scheduling: [A set of threads, scheduled to run on a set of processors at some time on one-2-one basis]

- simultaneous scheduling of threads that make up a single process.
- usefull for medium grained to fine grained ~~pralle~~ parallel application, whose performance, severely degrades when a part of app is not working.
- Benificial to any parallel application.

* Benifits ⇒ synchronization blocking may be reduced, less process switching may be necessary, performance will increase.
⇒ scheduling overhead may be reduced.

3) Dedicated Processor Assignment:

[provides an implicit scheduling defined by, assignments of threads to processors]

- When an application is scheduled, each of its thread is assigned to a processor that remains dedicated to that thread until app is completed.

- If the thread of an app is blocked waiting for I/O, the that thread's processor remains idle.
  → There is no multiprogramming of processors.

- In a highly parallel system, with tens or hundreds of processors, processor utilization is no longer as a metric of performance.

- The total avoidance of process switching during the lifetime of program deades speed of program.

4) Dynamic scheduling:

[The no. of threads in a process can be altered, during the course of execution].

- for some app. it is possible to provide language and system tools that permit no. of threads in a process to be altered.
  → allowing the os to adjust the load to improve utilization.

- Both the os & app are involved in making scheduling decisions.

- The scheduling responsibility of os is primarily limited to processor allocation.

- If app. can handle this, then this is a better method than others.

SDS | Page No. | Date

Cache Sharing

⇒ • Cooperative Resource Sharing
  • multiple threads access same set of memory location.
  • eg: multithreaded applications.
    : producer consumer thread interactions.

⇒ Resource Contention:
  • Threads if operating on adjacent cores compete for cache memory location.
  • If one thread gets more cache memory the other one will get less, leading to performance degradation.
  • Thus objective of contention-aware scheduling is to allocate threads to cores to maximize the effectiveness of shared memory & minimize need of off-chip memory.

Real time Systems

⇒ Here the OS & in particular the scheduler is most important.
⇒ the correct working depends not only on logical result but also on time when we get results.
⇒ try to handle the events that occur in real time & try to keep up with them.

eg: ⇒ control of Lab exp.
⇒ robotics
⇒ etc

* hard real time tasks → need to be compulsorily completed in stipulated amt. of time. else may cause fatal damage to system.

* soft real time tasks → has a deadline that is not mandatory & can be completed after deadline.

Characteristics of Real Time Systems:

- Determinism → operation are performed at fixed predefined interval
- Responsiveness ⇒ amt of time required to execute the task.
- User control ⇒ user needs to specify the hard / soft task & must use fine-grained system. (single instruction)
- Reliability ⇒ should be reliable as they are used in life saving objects. & may lead to fatal loss.
- Fail soft operation ⇒ a charac. that refers to ability of a system to fail in such a way as to preserve as much as data & capabilities possible.

Real time Scheduling:

- we decide scheduling based on systems scheduling analysis whether it is dynamically done or statically.
- Else the result of analysis produces a scheduler plan.

* static table driven approach:
  - performs a static analysis of feasible schedules of dispatching.
  - result is a schedule that determines, at run time, when a task must begin execution.

* static Property driven pre-Emptive approach:
  - a static analysis is performed but no schedule is drawn.
  - analysis is used to ~~design~~ assign priorities to tasks so that a traditional priority driven pre-Emptive scheduler can be used.

* Dynamic planning based approach:
  - feasibility is defined at run time rather than prior to start of execution.
  - One result of analysis is a schedule or plan that is used to decide when to dispatch this task.

* Dynamic Best approach:
  - no feasibility analysis is performed.
  - System tries to meet all deadlines & abort any started process whose deadline is missed.

* Deadline scheduling
→ Real Time os are designed with objective of starting real-time tasks as rapidly as possible & emphasize a rapid interrupt handing & task dispatching.
⇒ RT. apps are concerned with completing the task in stipulated time.
⇒ Priorities provide tool to not capture completion info when time is valuable.

* Ready time → task ready for execution.
* Starting deadline ⇒ task must begin.
* Completion deadline ⇒ task must end.
* Processing time ⇒ duration of execution
* Resource Requirements ⇒ resources required for execution.
* Priority ⇒ measures / importance of task
* sub task scheduler ⇒ task may be decomposed into some task.. (sub tasks).