CPU Scheduling

Basis of multiprogrammed operating system

By switching CPU among processes

↓

makes computer more productive.

Single processor — One process can run at a time
Others may wait until CPU is free and can be rescheduled.

Objective of multiprogramming — To have some processes running at all times to maximize CPU utilization.

A process is executed until it must wait, typically for the completion of some I/O request

In a simple computer system, CPU just sits idle, all this waiting time is wasted, no useful work is accomplished.

(when a process needs any I/O request to be completed)

** Decreases Productivity

Multiprogramming e, we use this time preductively.

(when CPU waits for I/O op to be completed)

How ___

1) Several processes are kept in memory at one time.

(When ~~the~~ one process have to wait, the OS takes the CPU away from the process and gives the CPU to another process and this patteron continues.

CPU and I/O burst cycles

Process execution

CPU execution state — I/O waiting state (Input/Output)

Process is under CPU execution (time - CPU Burst)

CPU is waiting for I/O (time - I/O Burst)

Process execution,

load, add store read from file ← CPU burst

↓

I/O op ← I/O burst

↓

store increment index write to file ← CPU burst

↓

I/O op ← I/O burst

Final CPU burst — ends with a system req to terminate execution.

Preemptive and Non Preemptive scheduling

CPU Scheduler when CPU becomes idle
↓
OS must select a process in the ready queue to be executed.

Ei selection process টাকে CPU scheduler বলে।

The scheduler selects a process from the memory that are ready to execute and allocate the CPU to that process.

Dispatcher is the module that give control of CPU
↓
to Process Selected by CPU schedulers.

To stop one process & start another running
↑
time taken — dispatch latency

*** CPU schedulic decision circumstances

1. when a process switches from running state to waiting state.
2. _____ from running state to ready state (for example, when an interrupt occurs) holds its execution
3. _____ from ~~ready~~ waiting to ready state. i.e. (at the completion of I/O)

4. When a process terminates.

For situations 1 and 4

No choice.

A new process from waiting queue must be selected for execution.

Situation 2

Not waiting state ← (process didnot go to)
Not terminated.

* Interrupt occurs.

* Holds its execution and ready state এ চলে
* Lost control of the CPU

Who should get the CPU.

এই same process পাবে? (যে interrupt এই এসেছে)
নাকি অন্য process.

Situation 3 : same.

Circumstances 1 and 4 হলে Non preemptive or cooperative

2 and 3 — Preemptive

Process — Priority wise

shared region of memory — into which different processes has access and different processes ~~has~~ can write/read from that shared region of memory.

So following this preemptive scheduling,

* Lets say one process was waiting something
* Another process came and first process had to be preempted or halted and CPU assigned to 2nd process
* Now, 2nd process will read some inconsistent data as 1st process didn't finish.

Scheduling Criteria

CPU utilization
Throughput
Turnaround time
Waiting time
Response time

CPU utilization — Keep the CPU as busy as possible.

0 to 100% (range)
real system (40%) slightly loaded
(90%) heavily "

How much the CPU is utilized

Throughput: measure of work done by the CPU
(no of processes that are completed per unit time is called throughput)

Turn around time: Important criteria is (how long it takes to execute a process)

Interval from the time of submision to " " " completion of a process is the turnaround time.

Sum of the periods spent – waiting to get into the memory
waiting in the ready queue
executing CPU
doing I/O operation.

Waiting time: CPU scheduling algorithm does not affect the amount of time during which a process executes or does I/O.

Affects the amnt of time the process spends waiting in the ready queue.

Sum of the periods a process spends waiting in the ready queue.

Responstime. In an interactive system, turnaround time may not be the best criteria.
Often a process can produce outputs early and continue computing new results while previous results are being showed to the user.
Thus,
Another measure of the time is from the submision of the request to until the first response is produced

Time taken to start responding, not to output the response.

Scheduling Algorithms

Turn Arround time = Completion time - Arrival time

Waiting time = Turn Arround time - Burst time

Efficiency = $\frac{\text{Useful time}}{\text{Total time}}$

FCFS (nonpreemptive) (simplest)

i) The process that requests the CPU first is allocated the CPU first.

ii) Implementation FIFO queue.

When a process enters the ready queue its PCB is linked onto the tail of the queue.

Tail □□□□□ Head

→

FIFO.

When the CPU is free, it is allocated to the process on to the head of the queue

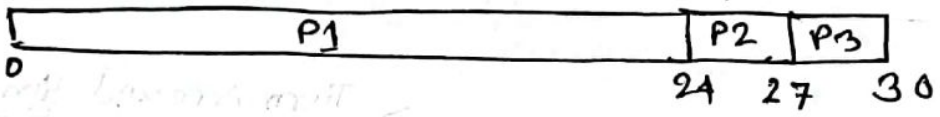
The running process is then removed from the queue

Avg waiting time - quite long

Let all of the process arrive at $\boxed{\text{time } 0}$

Process	Burst time (Time taken for execution)
P_1	24
P_2	3
P_3	3

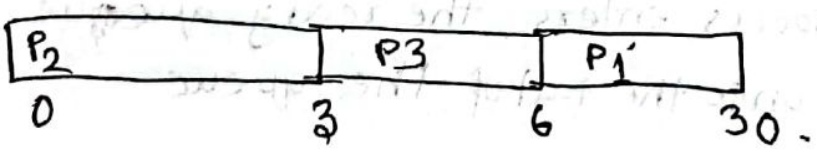
P_1, P_2, P_3 in FCFS order,



P_1 waited 0 ms
P_2 " 24 ms
P_3 " 27 ms.

Avg waiting time $(0+24+27)/3 = 17$ ms

P_2 P_3 P_1



Avg waiting time $(0+3+6)/3 = 3$ ms

↓

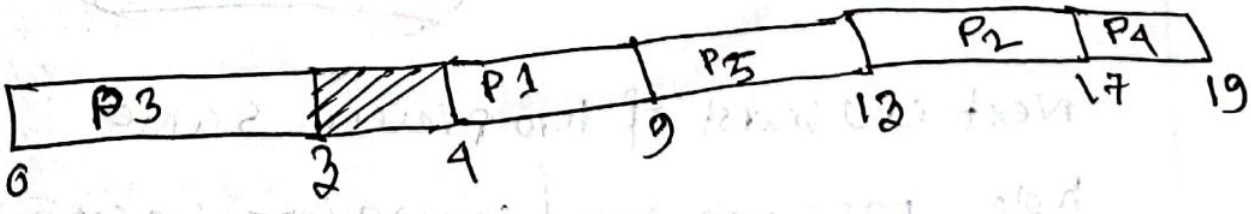
Depends on burs time & order.

* Troublesome for time-sharing system
  - where it is important for each user to get a share of the CPU at regular intervals
  - It would be disastrous to allow 1 process to keep CPU for a long time.

Solved Problem 1

If processes with higher burst time arrive before processes with ~~larger~~ smaller burst time, then smaller processes have to wait for a ~~long~~ time for longer processes to release the CPU. It is called convoy effect.

Process ID	AT	BT	CT
P1	4	5	9
P2	6	4	17
P3	0	3	3
P4	6	2	19
P5	5	4	13



P_1	0	3
P_2	1	2
P_3	2	1
P_4	3	4
P_5	4	5
P_6	5	2

D	P_1	D	P_2	D	P_3	D	P_4	D	P_5	D	P_6

0 1 4 5 7 8 9 10 14 15 20 21 23

Shortest Job First scheduling (Preemptive or Nonpreemptive)

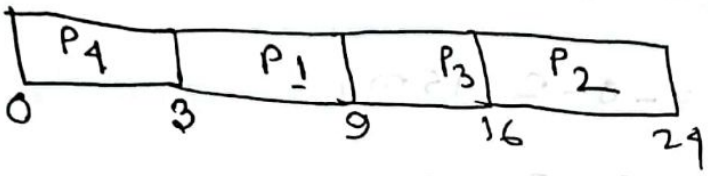
Length of the processes' next CPU burst

when the CPU is available, it is assigned to the process that has the smallest (next CPU burst)

Next CPU burst of two process same hole, FCFS use হয় (যে আগে আসবে তাকে Assign করা হয়)

SJF scheduling (Non preemptive)

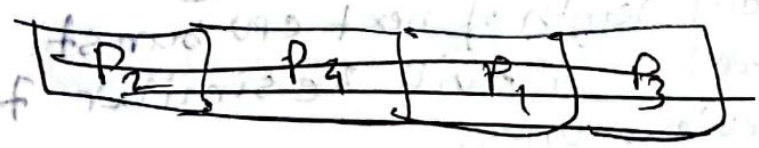
Process	Burst time	Completion	Arrival time
P_1	6	9	
P_2	8	24	
P_3	7	16	
P_4	3	3	

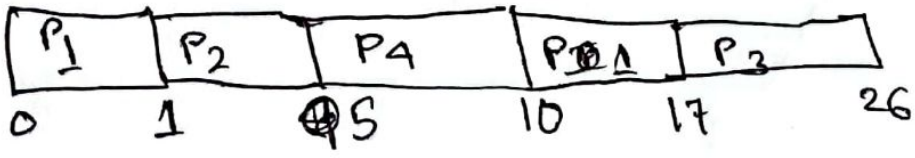


A.W.T $(3+16+9+0)/4 = 7$ ms.

Preemptive

Process ID	Arrival time	Burst time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5





SJF Scheduling (Preemptive)

Waiting time = Total waiting time - No of ms Process executed - Arrival time

$P_1 = 10 - 1 - 0 = 9$ ms

$P_2 = 1 - 0 - 1 = 0$ ms

$P_3 = 17 - 0 - 2 = 15$ ms

$P_4 = 5 - 0 - 3 = 2$ ms.

Problems

(i) Knowing the length of the next CPU req.

(ii) Although SJF is optimal, it cannot be implemented at the level of short term CPU scheduling.

(iii) There is no way to know the length of the next CPU burst.

Solution : * Approximate SJF scheduling,
* Predict length of next CPU burst.
* Expect " will be similar to previous ones.

Process ID	Arrival time	Burst time
P_1	0	12 (−2) 10
P_2	2	4
P_3	3	6 (−2) 4
P_4	8	5

Average waiting time = 2

→ Gant chart.

P_1	P_2	P_3	P_4	P_3	P_1

0, 2, 6, 8, 13, 17, 24

$P_1 = 17 - 2 - 0 = 15$

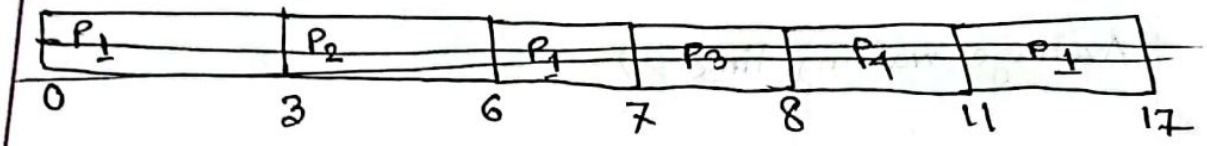
$P_2 = 2 - 0 - 2 = 0$

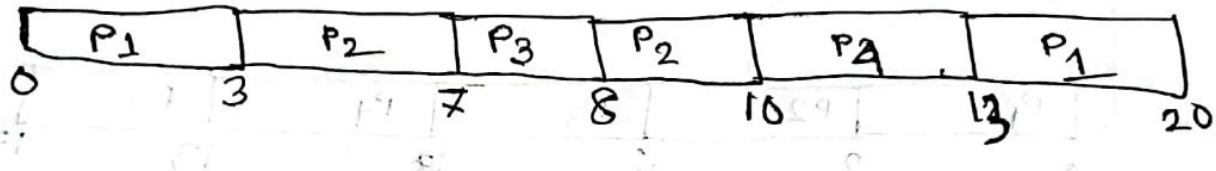
$P_3 = 13 - 2 - 3 = 8$

$P_4 = 8 - 0 - 8 = 0$

$15 + 8/4 =$

Process ID	Arrival Time	Burst Time
P1	0	10 (-3) 7
P2	3	6 (-4) 2
P3	7	1
P4	8	3





Priority Scheduling

(1) Priority is associated with each process.
CPU allocated with highest priority
Equal-priority _ FCFS order

SJF is a priority scheduling _ The larger CPU burst lower priority

can be preemptive or non-preemptive.

Prempt existing process if high priority process arrive.

Process

P_1

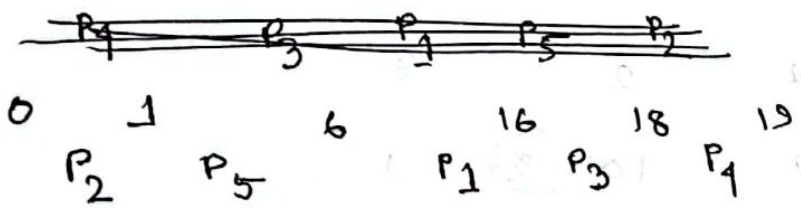
P_2

P_3

P_4

P_5

Smaller the value



$P_1 = 6$

$P_2 = 0$

$P_3 = 16$

$P_4 = 18$

$P_5 = 1$

Problems - Indefinite blocking or starvation

↓

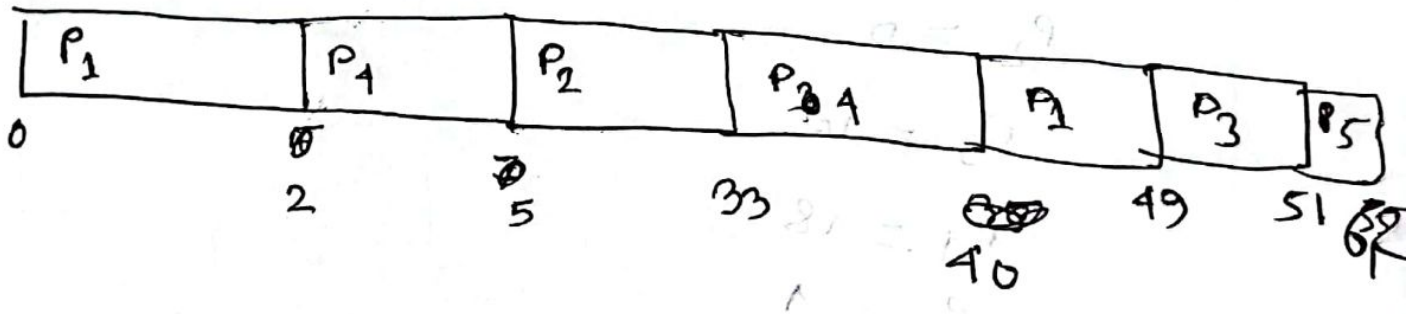
process that is ready to run but waiting for the CPU can be considered blocked.

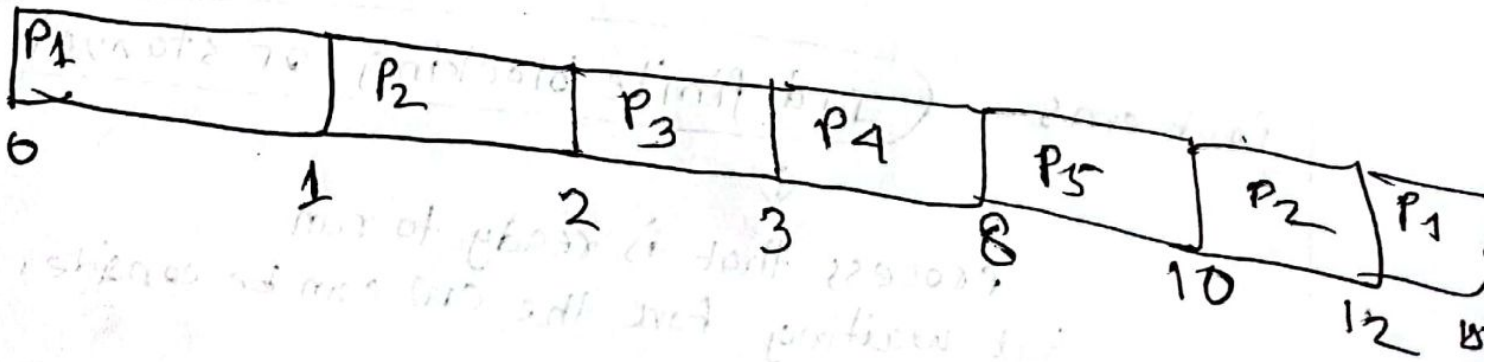
low priority processes wait indefinitely

Solution: Aging

└ Increasing Priority of processes that wait in the system for a long time.

	AT	BT	Priority
P_1	0	11 (−2)	2
P_2	5	28	0
P_3	12	2	3
P_4	2	10 (−3)	1
P_5	9	16	4



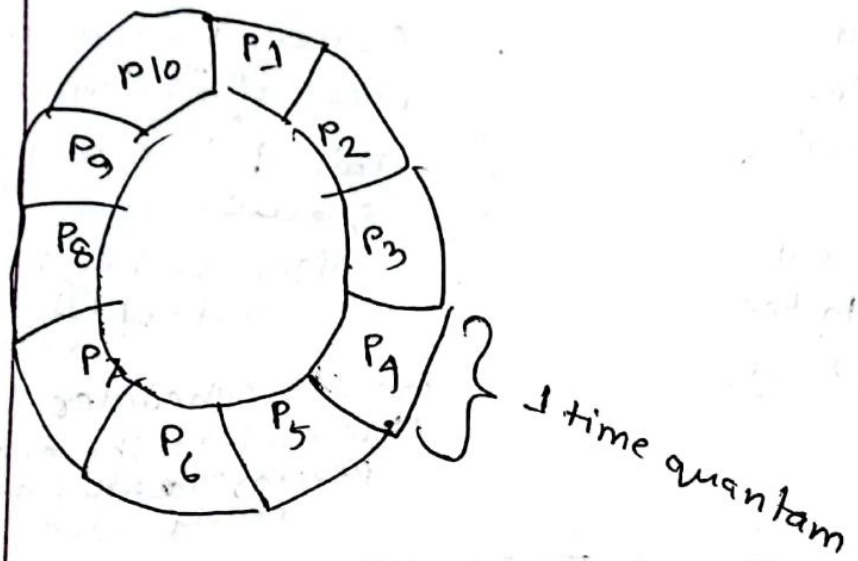


Round Robin Scheduling

Designed for — Time sharing system

Similar to — FCFS ~~[illegible]~~ with preemption (added to switch btwn processes)

— small unit of time (time quantum or time slice) (10-100 ms)



CPU given to a process for a particular time quantum

Implementation:

Ready queue (FIFO queue) of processes

New process — tail of the ready queue

CPU given for a particular time quantum

One of the two thing will happen

Process have CPU burst less than 1 time quantum.

Process will release CPU voluntarily.

Then CPU will proceed to the next process.

Process have more CPU burst than 1 time quantum.

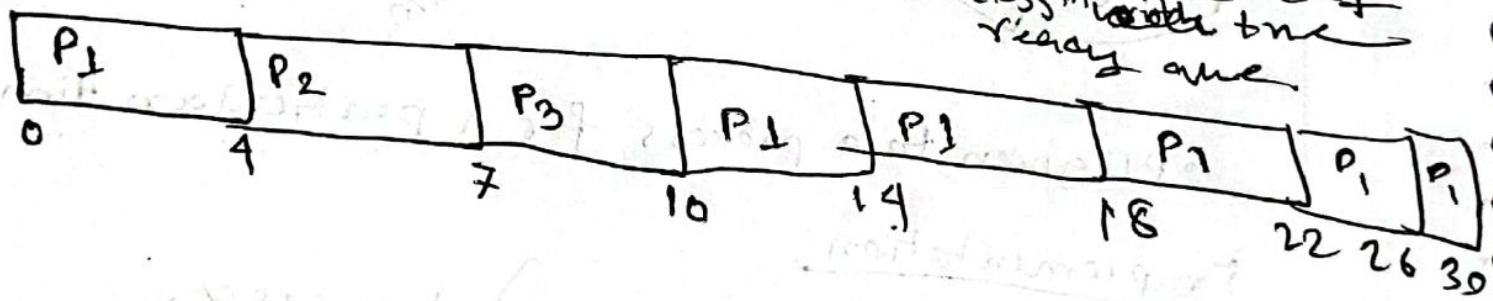
tail □□□□□□□ Head

Context switch executed

process will be put at the tail of the ready queue.

After saving the state of the process

The CPU scheduler will then select the next process in the ready queue

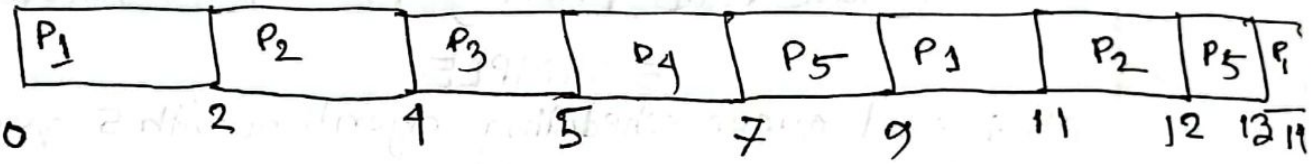


Turn around time = ~~30~~ Completion time − Arrival time

Waiting = Turn around − Burst

Waiting = Last start time − Arival time − (Preemption × Time Quantum)

Process ID	Arrival Time	Burst time
P1	0	5 (−2) (−2) 1
P_2	1	3 (−2)
P_3	2	1
P_4	3	2
P_5	4	3 (−2)



Multilevel Queue Scheduling

A class of scheduling algorithm has been created for situations in which processes are classified into different groups.

They have Different response time scheduling needs.

Foreground (quick) Background (not so quick)

Foreground
(Interactive)
Direct Interaction w/ user
(More priority)

Background
Batch
(Not interactive)

~~Foreground~~

Then scheduling will take place within this queues
And also among this ques.

A multilevel queue scheduling algorithm partitions the ready queue into several seperate queues.

Partitioned based on
- memory size
- Process Priority
- Process type.

Needs to be quick
proper time sharing

Each queue – its own scheduling algorithm.

Foreground queue – RR algorithm. (Proper TimeSharing)
Background " – FCFS "

should have fixed-priority preemptive scheduling

EXAMPLE

Multilevel queue scheduling algorithm with 5 queues.

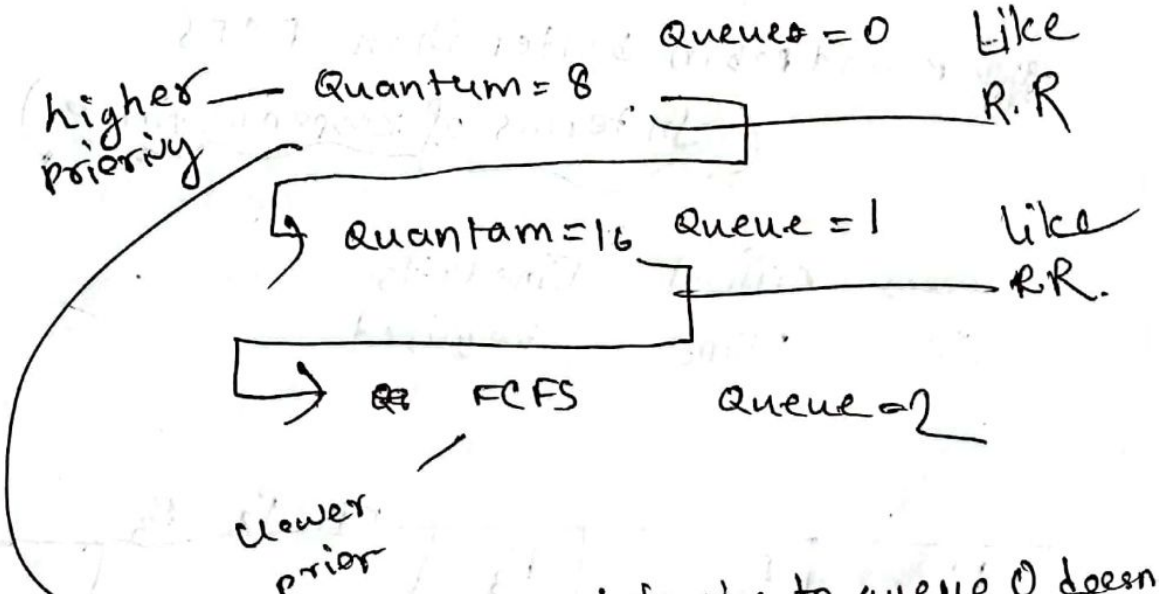
highest prio
→ System processes.
→ Interactive "
→ Interactive editing processes
→ Batch processes
→ Student "
lowest prio

Executed when all the above queues are empty (processes above it are executed)

There must be scheduling among the queues which should be implemented as a fixed priority preemptive scheduling Processes will be preempted when a process with higher priority queueu comes.

Multilevel Feedback-Queue Scheduling

- can move process between queues (from one queue to another).
- separate processes according to the characteristics of the CPU burst.
- If a process uses too much CPU time, it will be moved to a lower-priority queue.
- (leave) I/O and interactive processes to higher prior queues
- A process that waits too long in a lower priority queue may be moved to a higher priority queue. This form of aging prevents starvation.



If the process belonging to queue 0 doesn't finish execution within the time quantum then will be moved ~~tower~~ to the tail of Queue 1.

* Priority $\propto$ Waiting time
Starts with priority = 0
Quantum = T units.

Round Robin.

Contention Scope

User threads are managed by thread library & kernel is unware of them. To run on the CPU, user threads are mapped to kernel and this is done by light weight process.

The thread library schedules user threads to run on a LWP, it is known as $\boxed{\text{process contention scope}}$, since competition for the CPU takes place between the threads belonging to the same process.

To decide which kernel thread to schedule onto a CPU, the kernal uses system-contention scope (SCS)

PCS is done according to priority. The scheduler selects the runnable thread with the highest priority to run

Multiprocessor Scheduling If the processors are identical in terms of ~~there funct~~ their functionilities then we can use any processor.

One Approach: All system activities are carried out by one processor, the master server
Other processors executes the code.

This assymetric multiprocessing is simple becoz only one processor accesses the data structures reducing the need for sharing

Approach 2 Each process is self scheduling. All processes

Schema Evolution:

Field Tags

→ Forward Compabitibility

→ Backward Compatibility

Struct Person {

1: required string username → Data type → FieldName

2. optional i64 favourite Number

}

Field tag cant be changed.

→ Fields:

Backward compatible – পুরাতন নতুন field req হয় না

can't remove any field which is req

maybe be ready in the common queue, or each processor might have its own private queue of ready processes.

Processor Affinity The attempt to keep a process running on the same processor.

Soft Affinity: Attempting to keep a process running on the same process but not guaranting it Solaris

Hard Affinity: Doesn't let a process to migrate at all. Linux

Push Migration A specific task is assigned to check the load on each processor after a certain interval of time. If it finds imbalance, then migrates the waiting tasks of a busy processor to an idle one

Pull Migration: Idle processor nijei kaajta তুলে।

Multicore Processors (Complicate scheduling issues)

↑ SMP system - several threads run concurrently

multiple processor cores on the same physical chip

Each core — has a register set to maintain its architectural state (appears to the OS as a separate physical processor)

SMP systems with multicore Processors $\boxed{\text{faster}}$ than each processor with its own physical chip.

Memory stall when a processor accesses memory, it spends a significant amount of time waiting for the data.

cache miss - Accessing data not in cache memory.

Multithreaded processor core: Two or more hardware threads are assigned to each core.

If one thread stalls waiting for memory, then the core can switch to another thread.

Each hardware thread appears as a logical processor that is available to run a software thread.

2 ways to multithread the processor —

— coarse-grained

— fine-grained

Coarse grained: A thread executes on a processor until memory stall occurs. During the delay, the processor

should switch to another thread to begin execution.

Cost of switching between threads is high, as the instruction pipeline must be flushed before the other thread can begin execution on the processor core. Once this new thread begins execution, it begins filling the pipeline with its instructions.

Fine grained switches ~~wit~~ between threads at a much finer level of granularity, typically at the boundary of an instruction cycle.

— uses logic for thread switching. As a result cost for switching between threads is small.

~~Multilevel~~

multithreaded multicore processor

— 2 different level of scheduling

(i) OS chooses which software thread to run on each hardware thread.

(ii) each core decides which hardware thread to run.

VIRTUALIZATION

— single CPU system acts as a multiprocessor system (through virtualization)

— 1 host OS
many guest OS.

*** HOST OS — creates & Manages VMS

*** GUEST OS — Each VM has a GUEST OS installed & applications running within it.
(Used for time sharing or real time operations)

*** Actual time-Slice থেকে অনেক বেশি Time লাগাতে পারে virtual CPU তে। (Depends on how busy the system is)

virtualized OS recieve only A portion of CPU

CamScanner