CPU Scheduling

By: Dr Tarunpreet Bhatia

Assistant Professor

CSED, TIET

Disclaimer

This is NOT A COPYRIGHT MATERIAL

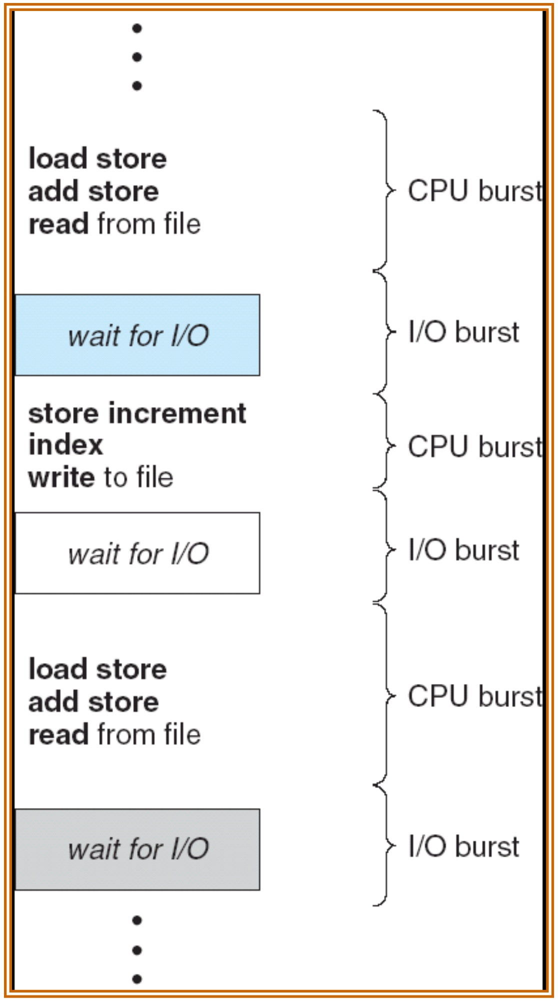
Content has been taken mainly from the following books:

Operating Systems Concepts By Silberschatz & Galvin,
Operating Systems: Internals and Design Principles By William Stallings

CPU Scheduling

- CPU Scheduling – Basis of Multiprogrammed Operating Systems.
- Multiprogramming is to have some process running at all times, to maximize CPU Utilization.
- Process Execution consists of a cycle of CPU execution and I/O wait.
- All the processes in the ready queue are lined up waiting for a chance to run on the CPU.
- The Records in the QUEUE are PCB of the Processes.

CPU – I/O Burst



load store
add store
read from file
CPU burst
wait for I/O
I/O burst
store increment
index
write to file
CPU burst
wait for I/O
I/O burst
load store
add store
read from file
CPU burst
wait for I/O
I/O burst

Dispatcher

- Module that gives control of the CPU to the Process selected by the Short Term Scheduler.

- Functions Involved are:

  - Switching Context.
  - Switching to User Mode.
  - Jumping to proper location in the user program to restart that program.

Dispatch Latency – Time it takes for the Dispatcher to Stop one process and Start another Running.

Scheduling Criteria

1. ***Burst time/execution time/running time***: *The time process require for running on CPU.*

2. ***Waiting time:*** *Time spend by a process in ready state waiting for CPU.*

   Waiting time (WT) = Turn around time – CPU burst time

3. ***Arrival time (AT):*** *When process is ready for execution.*

4. ***Exit time (ET):*** *When process completes execution and exit from the system.*

5. ***Turnaround time (TAT):*** *Total time spend in system.*

   TAT = ET – AT = BT + WT

6. ***Response time:*** *The time a process enters ready queue and get scheduled on CPU for first time.*

   Response Time = Time at which process first gets the CPU – Arrival time

Scheduling Criteria

7. CPU utilization – *keep the CPU as busy as possible*

CPU Utilization = Total CPU busy time/Total time required to process

8. Throughput – *No of processes that complete their execution per time unit*

Throughput = total number of processes /(Max exit time – min arrival time)

9. Response Ratio -

Response Ratio = (Waiting Time + Burst time) / Burst time

Optimization Criteria

i. ***Max CPU utilization***

ii. ***Max throughput***

iii. ***Min turnaround time***

iv. ***Min waiting time***

v. ***Min response time***

Scheduling Algorithms

- First come first serve (FCFS)
- Shortest job first (SJF)
- Shortest remaining time first (SRTF)
- Priority Scheduling
- Round Robin (RR)
- Multi-level Queue
- Multi-level Feedback Queue

FCFS

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule is:

FCFS

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule is:

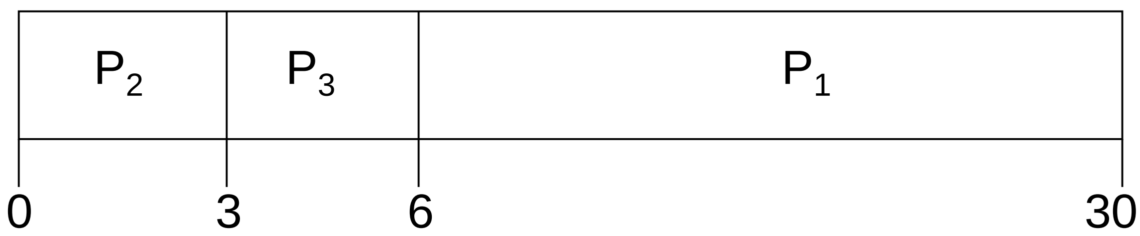
- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17
- Turnaround Time: P_1 = 24 , P_2 = 27, P_3 = 30
- Average TT: (24 + 27 + 30) / 3 = 27

FCFS

Suppose that the processes arrive in the order

P_2 , P_3 , P_1

- The Gantt chart for the schedule is:



- Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Average waiting time: (6 + 0 + 3)/3 = 3
- Turnaround time: $P_1 = 30$, $P_2 = 3$, $P_3 = 6$
- Average TT: (30 + 3 + 6) / 3 = 13 – much less than 27
- Much better than previous case

FCFS

Process	Arrival time	Burst time	Exit time	TAT	WT	Response time
P_1	0	12				
P_2	1	6				
P_3	4	9				

FCFS Scheduling (Cont.)

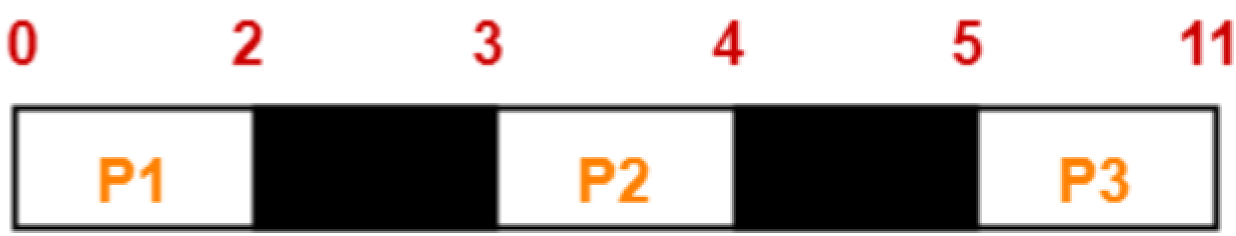
- **Convoy effect :** as all the other processes wait for the one big process to get off the CPU.
- **Advantages:** Simple, easy to use, easy to understand, easy to implement, must be used for background processes where execution is not urgent.
- **Disadvantages:** Suffer from convoy effect, normally higher average waiting time, no consideration of priority and burst time, should not be used for interactive systems.
- **No starvation, only convoy effect.**

FCFS

Consider the set of 3 processes whose arrival time and burst time are given below-

Process Id	*Arrival time*	*Burst time*
P1	*0*	*2*
P2	*3*	*1*
P3	*5*	*6*

If the CPU scheduling policy is FCFS, calculate the average waiting time and average turn around time.



Gantt Chart

FCFS

Now, we know-

- Turn Around time = Exit time – Arrival time
- Waiting time = Turn Around time – Burst time

Process Id	Exit time	Turn Around time	Waiting time
P1	2	2 – 0 = 2	2 – 2 = 0
P2	4	4 – 3 = 1	1 – 1 = 0
P3	11	11- 5 = 6	6 – 6 = 0

Now,

- Average Turn Around time = (2 + 1 + 6) / 3 = 9 / 3 = 3 unit
- Average waiting time = (0 + 0 + 0) / 3 = 0 / 3 = 0 unit

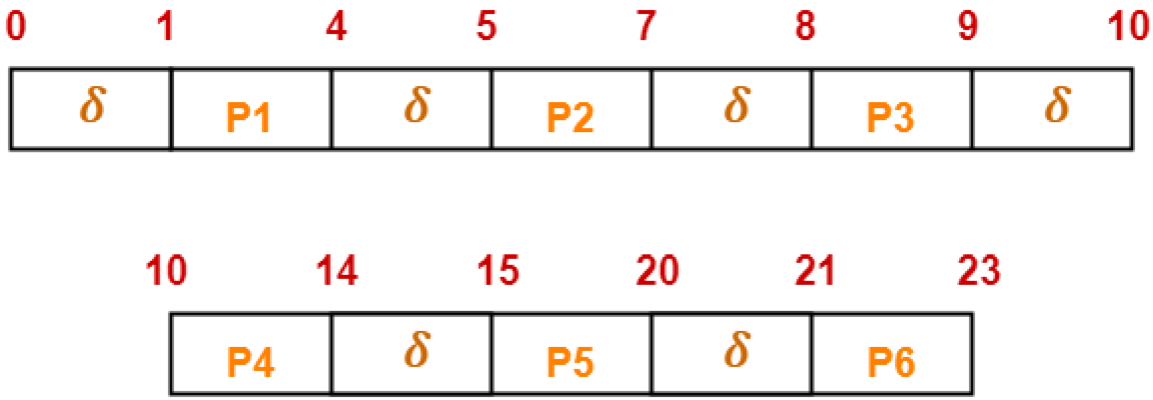
FCFS

Consider the set of 6 processes whose arrival time and burst time are given below-

Process Id	Arrival time	Burst time
P1	0	3
P2	1	2
P3	2	1
P4	3	4
P5	4	5
P6	5	2

If the CPU scheduling policy is FCFS and there is 1 unit of overhead in scheduling the processes, find the efficiency of the algorithm.

FCFS



Gantt Chart

Now,

- Useless time / Wasted time = 6 x δ = 6 x 1 = 6 unit
- Total time = 23 unit
- Useful time = 23 unit – 6 unit = 17 unit

Efficiency (η)

= Useful time / Total Total

= 17 unit / 23 unit

= 0.7391

SJF

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time

- Two Schemes:
  - Non Preemptive – Once CPU given to the process it cannot be preempted until completes its CPU burst

  - Preemptive – If a New Process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the Shortest-Remaining-Time-First (SRTF)

- SJF is Optimal – Gives minimum average waiting time for a given set of processes

SJF (Non Preemptive)

Process	Burst time	Exit time	TAT	WT
P_1	6			
P_2	8			
P_3	7			
P_4	3			

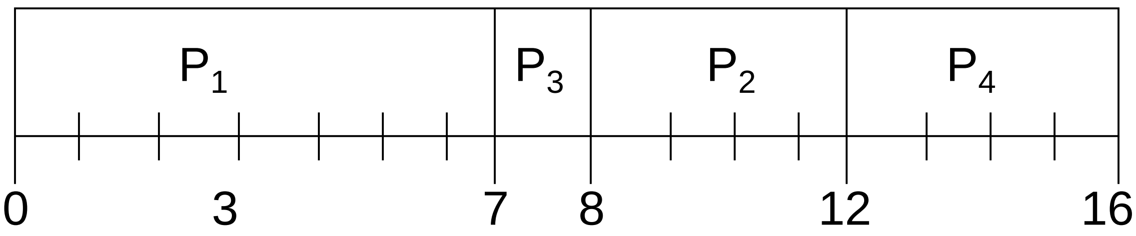
SJF (Non Preemptive)

Process	Arrival time	Burst time	Exit time	TAT	WT	RT
P_1	0	7				
P_2	2	4				
P_3	4	1				
P_4	5	4				

SJF (Non Preemptive)

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (Non-Preemptive)

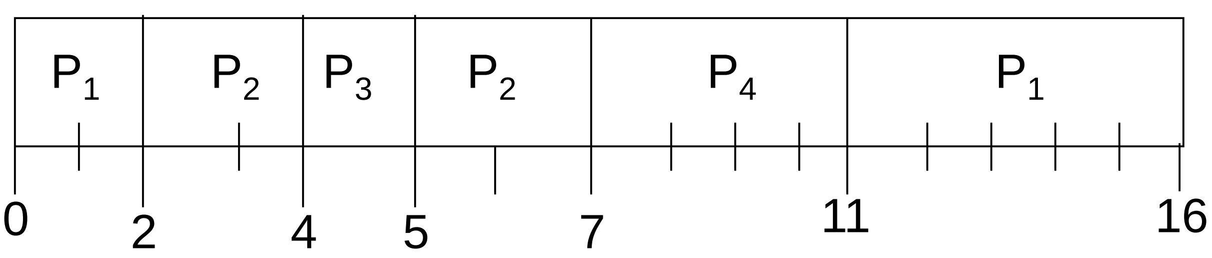


SJF (Preemptive)/SRTF

Process	Arrival time	Burst time	Exit time	TAT	WT	RT
P_1	0	7				
P_2	2	4				
P_3	4	1				
P_4	5	4				

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (Preemptive)



SRTF

Process	Arrival time	Burst time	Exit time	TAT	WT	RT
P_1	0	8				
P_2	1	4				
P_3	2	9				
P_4	3	5				

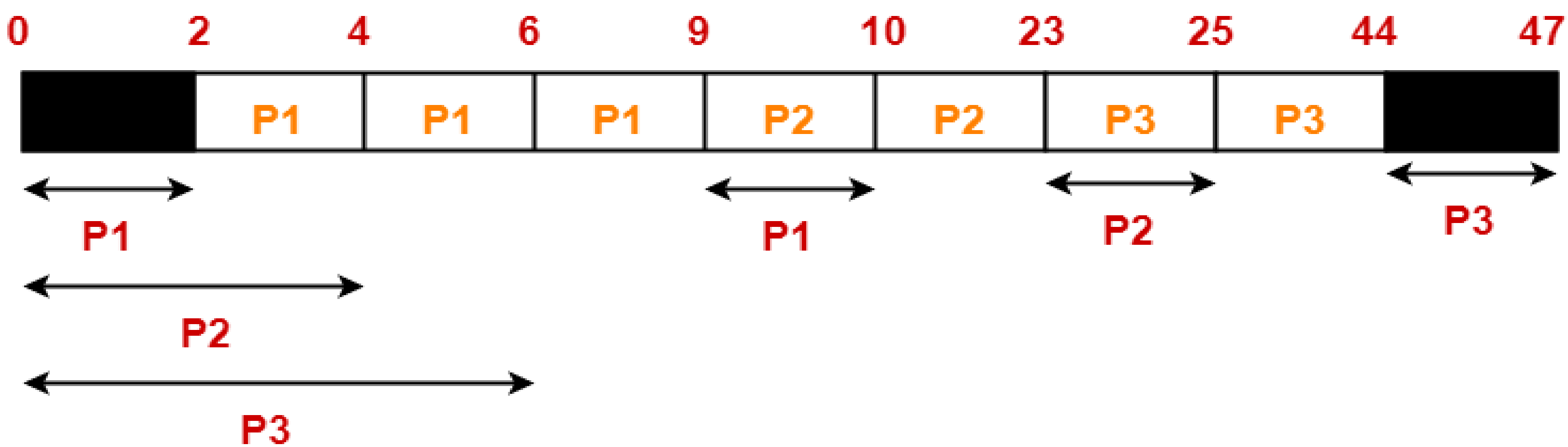
SRTF

Process	Arrival time	Burst time	Exit time	TAT	WT	RT
P_1	0	10				
P_2	6	5				
P_3	7	2				
P_4	8	3				

Question

Consider three processes, all arriving at time zero, with total execution time (CPU and I/O) of 10, 20 and 30 units, respectively. Each process spends the first 20% of execution time doing I/O, the next 70% of time doing computation, and the last 10% of time doing I/O again. The multiprogramming operating system uses a SJF scheduling algorithm. Assume that all I/O operations can be overlapped as much as possible. For what percentage of time does the CPU remain idle?

Process	A.T	Burst time	i/o, cpu time, i/o time
P1	0	10	2, 7, 1
P2	0	20	4, 14 , 2
P3	0	30	6, 21, 3



Gantt Chart

*CPU idle time = (5/47)*100 =10.64%*

*CPU Utilization = (42/47)*100 = 89.36%*

Dynamic method for next CPU burst prediction in SJF

Let t_i be the actual Burst-Time of i^{th} process and T_{n+1} be the predicted Burst-time for $n+1^{th}$ process.

- **Simple average –** Given n processes (P_1, P_2... P_n)

$$T_{n+1} = 1/n(\Sigma^{i=1 \text{ to } n}\, t_i)$$

- **Exponential average (Aging)**

$$T_{n+1} = \alpha t_n + (1 - \alpha)T_n$$

where α = is smoothing factor and $0 <= \alpha <= 1$,

t_n = actual burst time of n^{th} process,

T_n = predicted burst time of n^{th} process.

General term,

$\alpha t_n + (1 - \alpha)\alpha t_{n-1} + (1 - \alpha)^2\alpha t_{n-2} ... + (1 - \alpha)^j\alpha t_{n-j} ... + (1 - \alpha)^{n+1}T_0$

$\mathbf{T_0}$ is a constant or overall system average.

Smoothening factor (α)

- It controls the relative weight of recent and past history in our prediction.
- If $\alpha = 0$, $T^{n+1} = T^{n}$ i.e. no change in value of initial predicted burst time.
- If $\alpha = 1$, $T^{n+1} = t^{n}$ i.e. predicted Burst-Time of new process will always change according to actual Burst-time of n^{th} process.
- If $\alpha = 1/2$, recent and past history are equally weighted.

Example of Exponential Averaging

Calculate the exponential averaging with T1 = 10, $\alpha = 0.5$ and the algorithm is SJF with previous runs as 8, 7, 4, 16. The predicted burst time for process P4 is:

(a) 9

(b) 8

(c) 7.5

(d) None of these

Solution

Initially $T1 = 10$ and $\alpha = 0.5$ and the run times given are 8, 7, 4, 16 as it is shortest job first, So the possible order in which these processes would serve will be 4, 7, 8, 16 since SJF is a non-preemptive technique.

So, using formula: $T2 = \alpha * t1 + (1-\alpha)T1$

so we have,

$T2 = 0.5*4 + 0.5*10 = 7$, here $t1 = 4$ and $T1 = 10$

$T3 = 0.5*7 + 0.5*7 = 7$, here $t2 = 7$ and $T2 = 7$

$T4 = 0.5*8 + 0.5*7 = 7.5$, here $t3 = 8$ and $T3 = 7$

So the future prediction for 4th process will be $T4 = 7.5$ which is the option(c).

Examples of Exponential Averaging

Calculate the predicted burst time using exponential averaging for the fifth process if the predicted burst time for the first process is 10 units and actual burst time of the first four processes is 6, 4, 6 and 4 units respectively. Given α = 0.5.

Examples of Exponential Averaging

Calculate the predicted burst time using exponential averaging for the fifth process if the predicted burst time for the first process is 20 units and actual burst time of the first four processes is 6, 10, 4 and 7 units respectively. Given $\alpha = 0.5$.

In this example, if the predicted burst time for the first process is 20 units and the actual burst time of the first four processes is 6, 10, 4 and 7 units respectively.

So given that, predicted burst time for first process = 20 units.

And actual burst time of the first four processes are as follows = 6, 10, 4, 7.

We can calculate the predicted Burst Time for the second Process by,

= α * Actual burst time of first process + (1-α) * Predicted burst time for first process

= 0.5 * 6 + 0.5 * 20

= 3 + 10

= 13 units

We can calculate predicted Burst Time for third Process by,

= α * Actual burst time of second process + (1-α) * Predicted burst time for second process

= 0.5 * 10 + 0.5 * 13

= 5 + 6.5

= 11.5 units

We can calculate predicted burst time for fourth process by,

= α * Actual burst time of third process + (1-α) * Predicted burst time for third process

= 0.5 * 4 + 0.5 * 11.5

= 2 + 5.75

= 7.75 units

We can calculate predicted Burst Time for fifth Process by,

= α * Actual burst time of fourth process + (1-α) * Predicted burst time for fourth process

= 0.5 * 7 + 0.5 * 7.75

= 3.5 + 3.875

= 7.375 units

SJF/SRTF

Advantages -

- SRTF is optimal and guarantees the minimum average waiting time.
- It provides a standard for other algorithms since no other algorithm performs better than it.

Disadvantages -

- It can not be implemented practically since burst time of the processes can not be known in advance.
- It leads to starvation for processes with larger burst time.
- Priorities can not be set for the processes.
- Processes with larger burst time have poor response time.

Priority Scheduling

- A Priority Number (Integer) is associated with each Process.
- The CPU is allocated to the Process with the Highest Priority
  - Preemptive
  - Non Preemptive
- SJF is a Priority Scheduling where PRIORITY IS THE PREDICTED NEXT CPU BURST TIME
- Problem in Priority scheduling is STARVATION – Low Priority processes may never execute
- Solution for above mentioned Problem is AGING – as time progresses increase the priority of the process

Priority Scheduling

Assume lower number indicates high priority

Process	Priority	Burst Time
P_1	3	10
P_2	1	1
P_3	4	2
P_4	5	1
P_5	2	5

Priority Scheduling

Process	Priority	Burst time	Arrival time
P1	3	10	0
P2	4	6	5
P3	2	3	6
P4	1	4	10

Assume lower number indicates high priority

a) Non-preemptive

b) Preemptive

Solution (Non-Preemptive)

Process	Priority	Burst time	Arrival time	ET	TAT	WT
P_1	3	10	0			
P_2	4	6	5			
P_3	2	3	6			
P_4	1	4	10			

Solution (Preemptive)

Process	Priority	Burst time	Arrival time	ET	TAT	WT
P_1	3	10	0			
P_2	4	6	5			
P_3	2	3	6			
P_4	1	4	10			

Priority Scheduling

Process	Priority	Burst time	Arrival time	ET	TAT	WT
P_1	6	6	0			
P_2	9	3	2			
P_3	4	8	4			
P_4	8	2	5			
P_5	1	7	7			

Assume lower number indicates high priority

a) Non-preemptive

b) Preemptive

Priority Scheduling

Process	Priority	Burst time	Arrival time
A	4	5	0.0000
B	2	4	2.0001
C	6	2	2.0001
D	3	4	4.0001

Assume higher number indicates high priority

a) Non-preemptive

b) Preemptive

Priority Scheduling

Advantages

- It considers the priority of the processes and allows the important processes to run first specially for system processes.
- Priority scheduling in preemptive mode is best suited for real time operating system.

Disadvantages

- Processes with lesser priority may starve for CPU.
- Priority scheduling algorithm can leave some low priority processes waiting indefinitely.

Priority Scheduling

- ***Solution*** $\equiv$ ***Aging***
- Aging ensures that processes with lower priority will eventually complete their execution.
- By gradually increasing the priority of processes that wait in the system for long time.

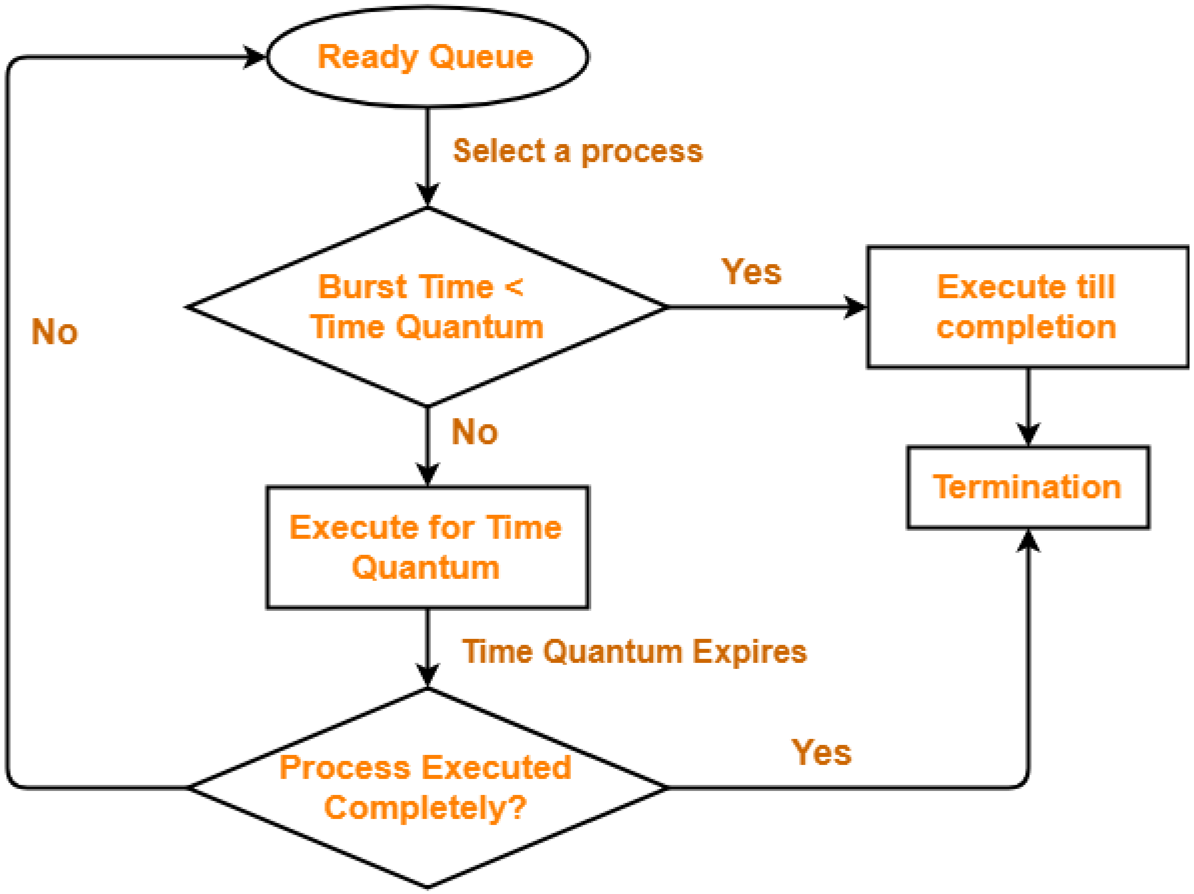
Round Robin

- Each Process gets a Small Unit of CPU time (Time Quantum).
- After this time has elapsed, the PROCESS IS PREEMPTED and added to the end of the Ready Queue.
- If there are n processes in the ready queue and the time quantum is q, then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than $(n-1)q$ time units.
- *PERFORMANCE*

q Large $\Rightarrow$ FIFO

q Small $\Rightarrow$ q must be large with respect to Context Switch, Otherwise overhead is too high

Round Robin



Round Robin Scheduling

Round Robin

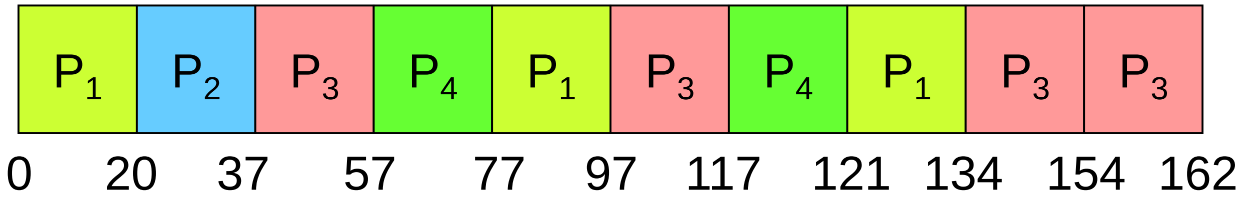
Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- Quantum = 20

Round Robin

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- The Gantt Chart is:



Round Robin

Process	Burst Time
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

- Quantum = 10

Round Robin Quantum = 5

Process	Burst Time	Arrival Time
P_1	10	0
P_2	7	6
P_3	8	16
P_4	9	18

Scheduling Algorithms

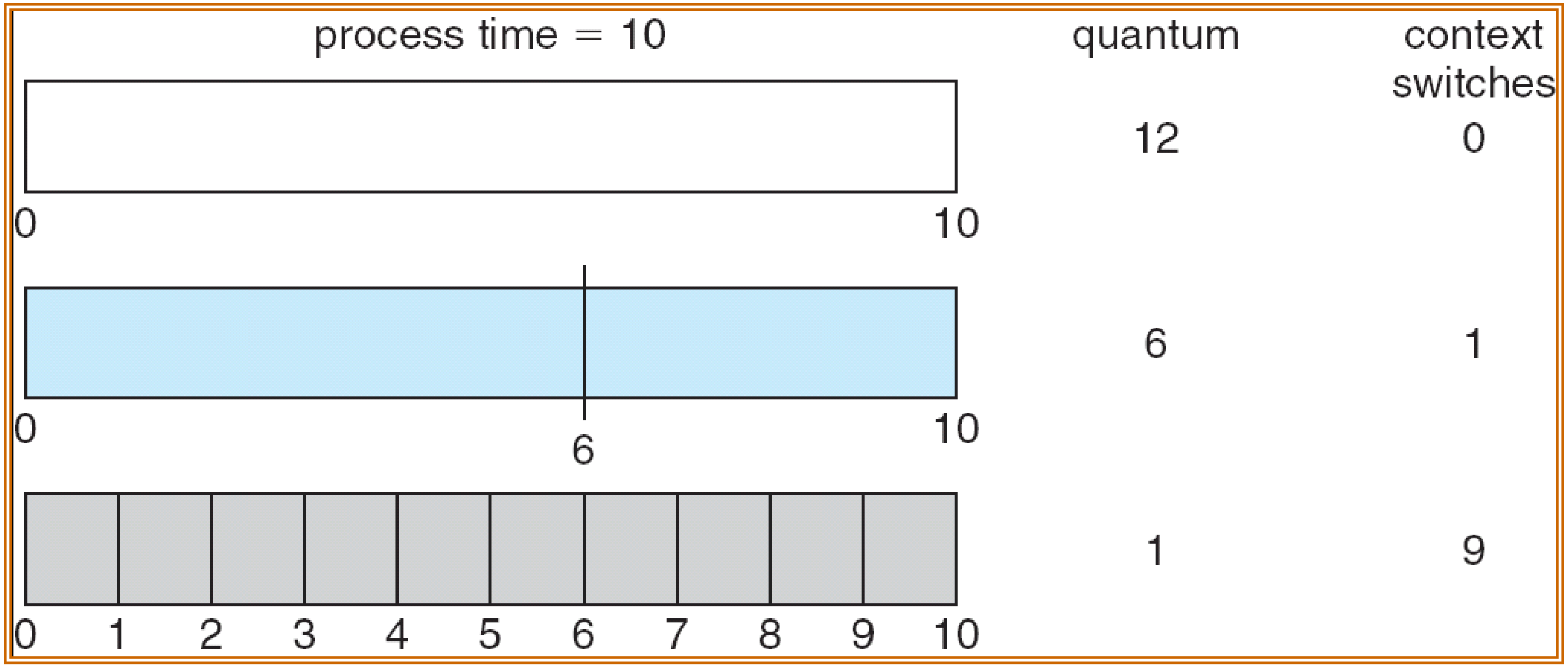
Process	Burst time	Arrival time
A	3	0.000
B	6	1.001
C	4	4.001
D	2	6.001

a) FCFS
b) SJF
c) SRTF
d) RR with q=2
e) RR with q=1

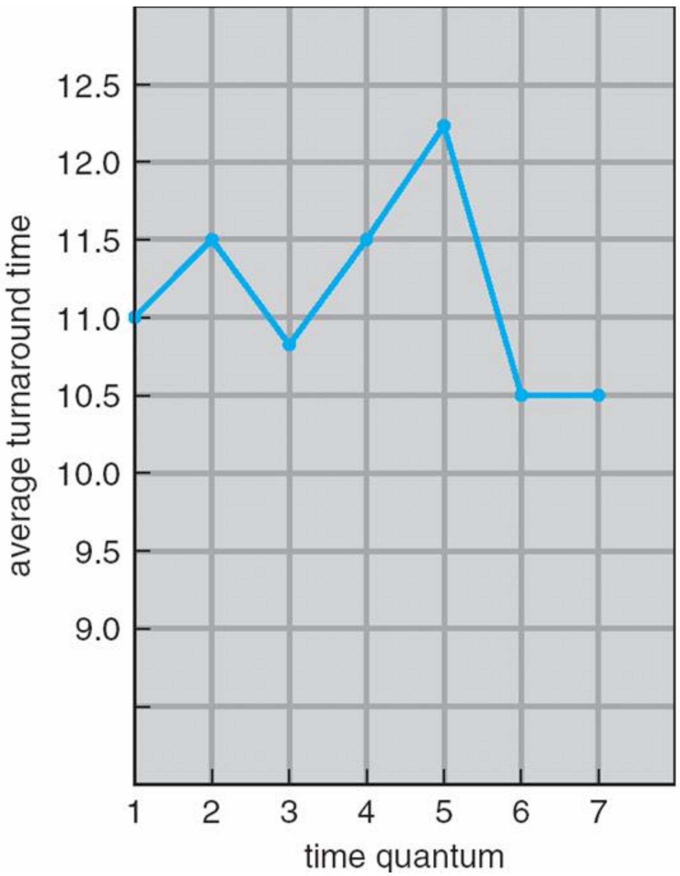
Round Robin Quantum = 1

Process	Burst Time	Arrival Time
P_1	7	0
P_2	4	2
P_3	1	4
P_4	4	5

Time Quantum and Context Switch



Turnaround Time Varies With The Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

80% of CPU bursts should be shorter than q

Multilevel Queue

- Ready Queue is partitioned into separate Queues:
  Foreground
  Background

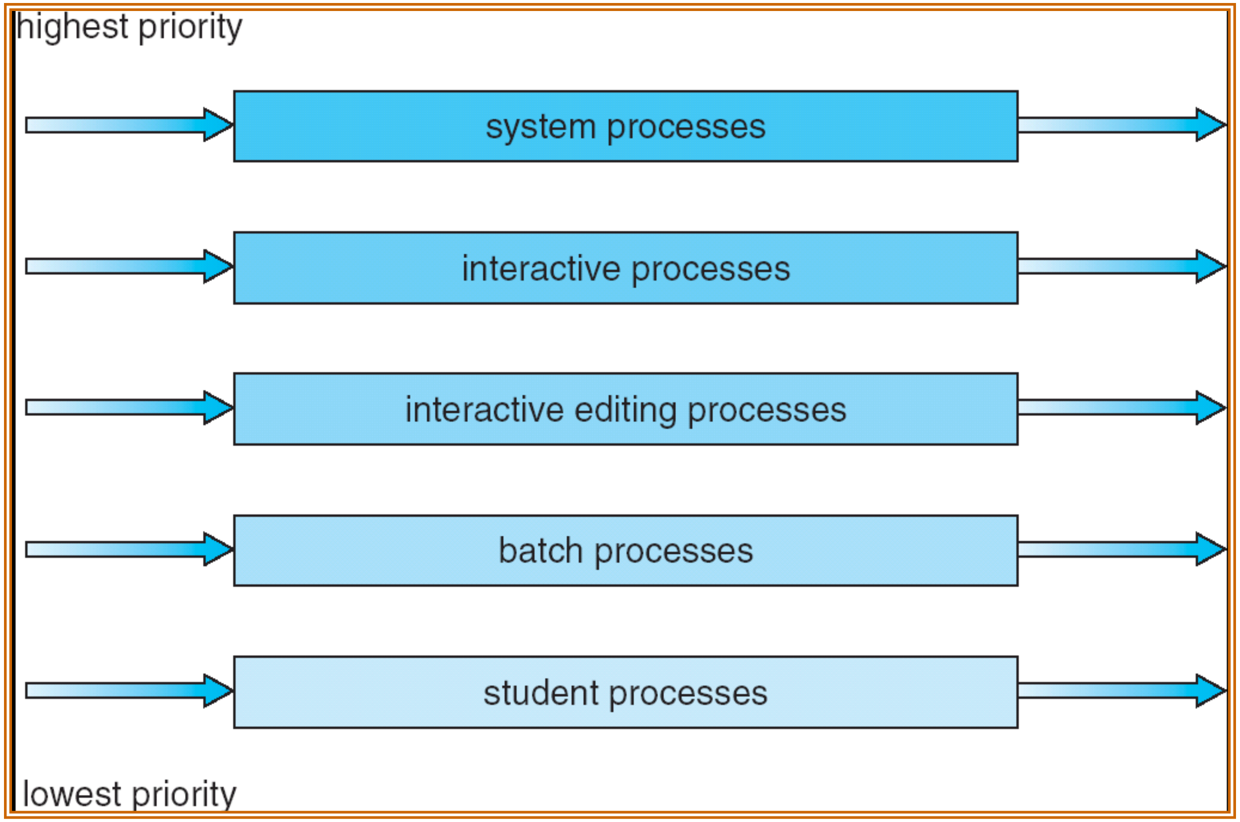
- Each QUEUE has its own Scheduling Algorithm
  Foreground – RR
  Background – FCFS

- Scheduling must be done between the queues

  - Fixed Priority Scheduling - (i.e., Serve all from foreground then from background). Possibility of Starvation.

  - Time Slice – Each Queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR and 20% to background in FCFS

Multilevel Queue Scheduling



Question

Consider below table of four processes under Multilevel queue scheduling. Queue number denotes the queue of the process. Priority of queue 1 is greater than queue 2. Queue 1 uses Round Robin (Time Quantum = 2) and queue 2 uses FCFS. Find out Avg. waiting time and Avg. Turn around time.

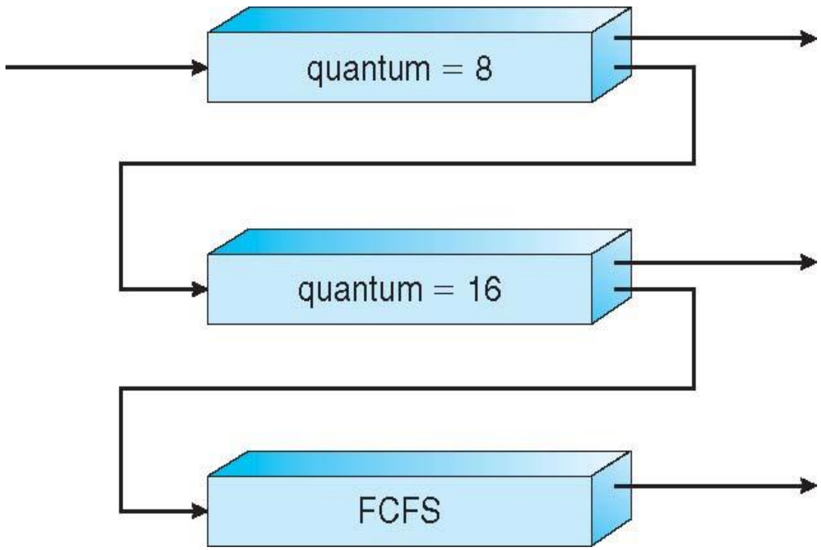
Process	AT	BT	Queue No
P1	0	4	1
P2	0	3	1
P3	0	8	2
P4	10	5	1

Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

Example of Multilevel Feedback Queue

- Three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS
- Scheduling
  - A new job enters queue Q_0 which is served FCFS/RR (in terms of ready queue which is implemented as FCFS. They are given quantum of atmost 8 but in FCFS order. New process enters at the tail and processes are served from head)
    - When it gains CPU, job receives 8 milliseconds
    - If it does not finish in 8 milliseconds, job is moved to queue Q_1
  - At Q_1 job is again served FCFS and receives 16 additional milliseconds
    - If it still does not complete, it is preempted and moved to queue Q_2
  - If a process does not use up its quantum in the current level, it will keep its current queuing level and be put into the end of the queue. Then, it can still get the same amount of quantum (not remaining quantum) next time when it is picked



Questions

- On a system with multilevel feedback queue, CPU bound process requires 50 sec to execute. If first queue uses a quantum of 5 sec and at each level, the time quantum doubles, how many times will job be interrupted and on what queue will it be terminates?
- On a system with multilevel feedback queue, CPU bound process requires 40 sec to execute. If first queue uses a quantum of 2 sec and at each level, the time quantum increases by 5, how many times will job be interrupted and on what queue will it be terminates?

Solution

- On a system with multilevel feedback queue, CPU bound process requires 40 sec to execute. If first queue uses a quantum of 2 sec and at each level, the time quantum increases by 5, how many times will job be interrupted and on what queue will it be terminates?

Q1 = 2   2

Q2 = 7   9

Q3 = 12   12+9=21

Q4 = 17   21+17=38

→ Q5 = 22   2

Question

Consider a multilevel feedback queue scheduler with 3 queues 0 to 2.

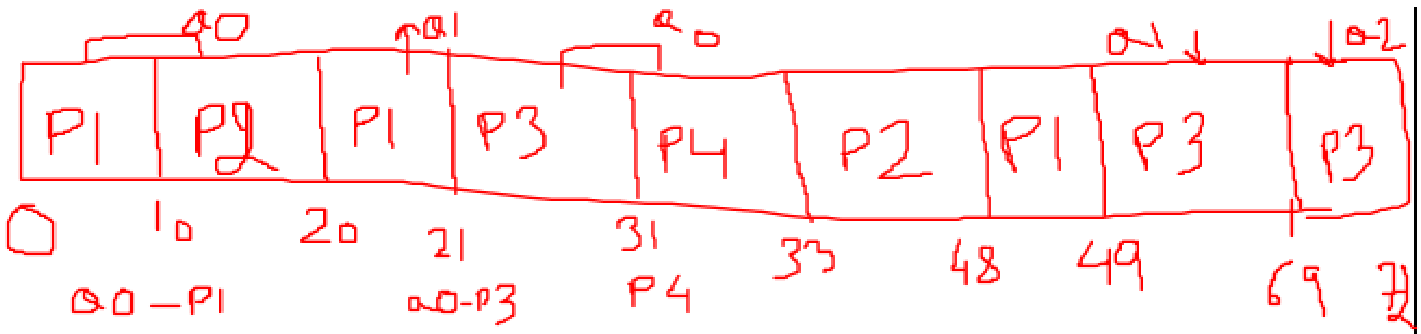
Queue 0 RR scheduling with Quantum 10

Queue 1 RR scheduling with Quantum 20

Queue 2 FCFS only when Queue 0 and Queue 1 are empty

Draw the Gantt chart for the following:

Process	Burst time	Arrival time
P1	12	0
P2	25	8
P3	33	21
P4	2	30



Q0 — P1

Q0-P3 P4

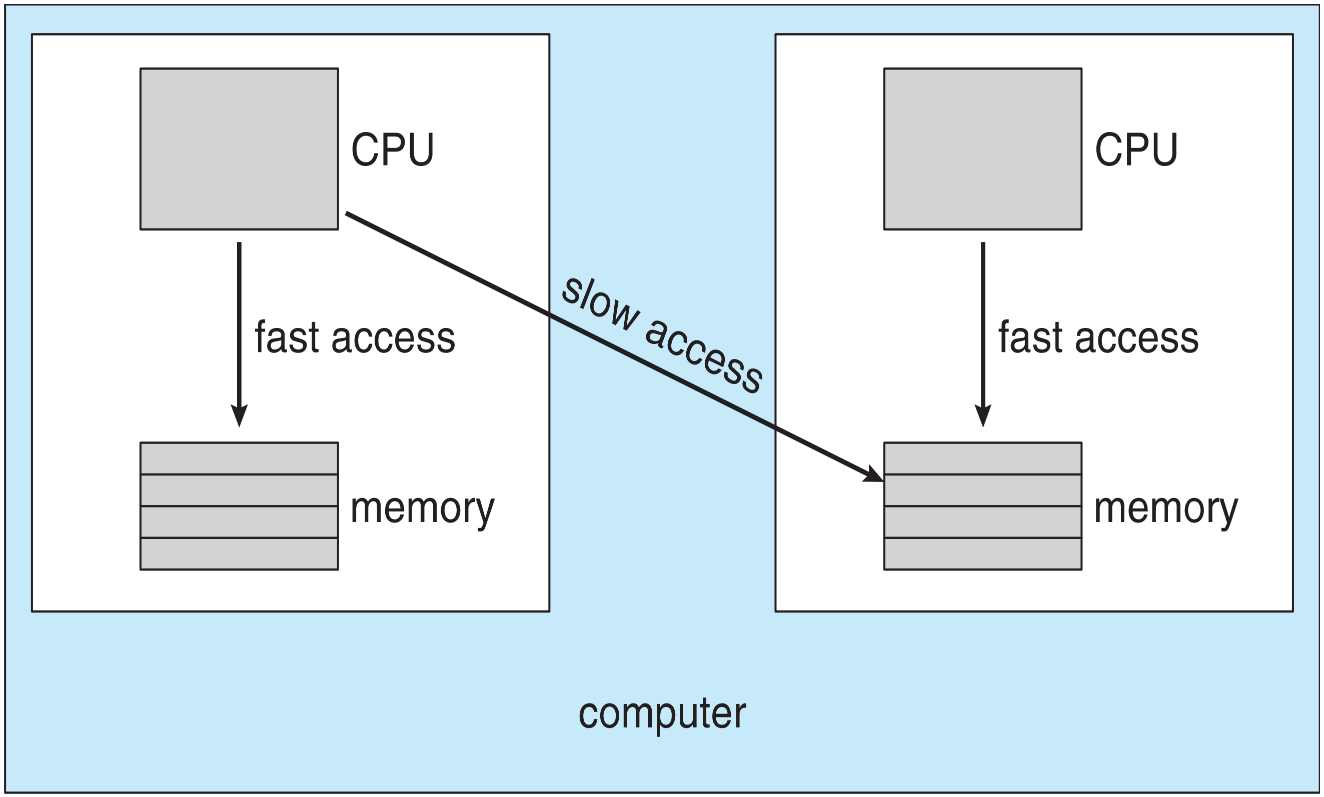
Q1 - ~~P2~~ — P1 — P3

Q1

Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available
- **Homogeneous processors** within a multiprocessor
- **Asymmetric multiprocessing** – only one processor accesses the system data structures, alleviating the need for data sharing
- **Symmetric multiprocessing (SMP)** – each processor is self-scheduling, all processes in common ready queue, or each has its own private queue of ready processes
    - Currently, most common
- **Processor affinity** – process has affinity for processor on which it is currently running
    - **soft affinity**
    - **hard affinity**

NUMA and CPU Scheduling



Multiple-Processor Scheduling – Load Balancing

- If SMP, need to keep all CPUs loaded for efficiency
- **Load balancing** attempts to keep workload evenly distributed
- **Push migration** – periodic task checks load on each processor, and if found pushes task from overloaded CPU to other CPUs
- **Pull migration** – idle processors pulls waiting task from busy processor

Multicore Processors

- Recent trend to place multiple processor cores on same physical chip
- Faster and consumes less power
- Multiple threads per core also growing
  - Takes advantage of memory stall to make progress on another thread while memory retrieve happens

Multithreaded Multicore System