MM PG College Fatehabad

OPERATING SYSTEMS
CPU Scheduling

Class:- B.Sc. (CS)
2^{nd} Year/Sem. 4^{th}

Basic Concepts
Scheduling Criteria
Scheduling Algorithms
Multiple-Processor Scheduling
Real-Time Scheduling
Algorithm Evaluation

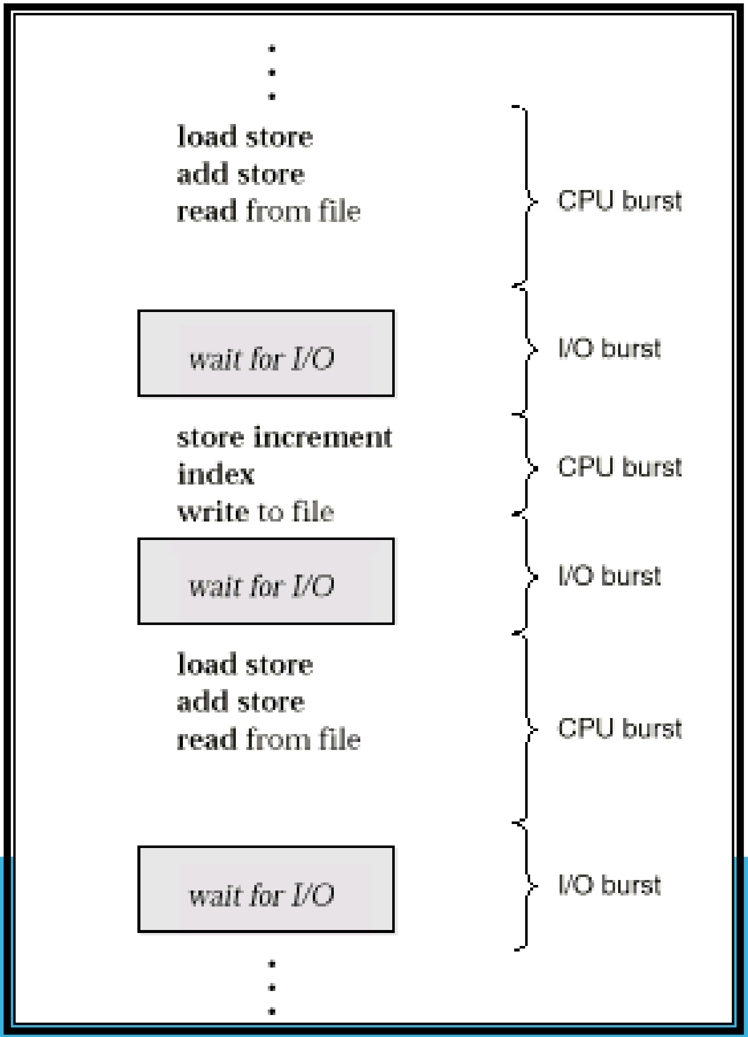
BASIC CONCEPTS

Maximum CPU utilization obtained with multiprogramming

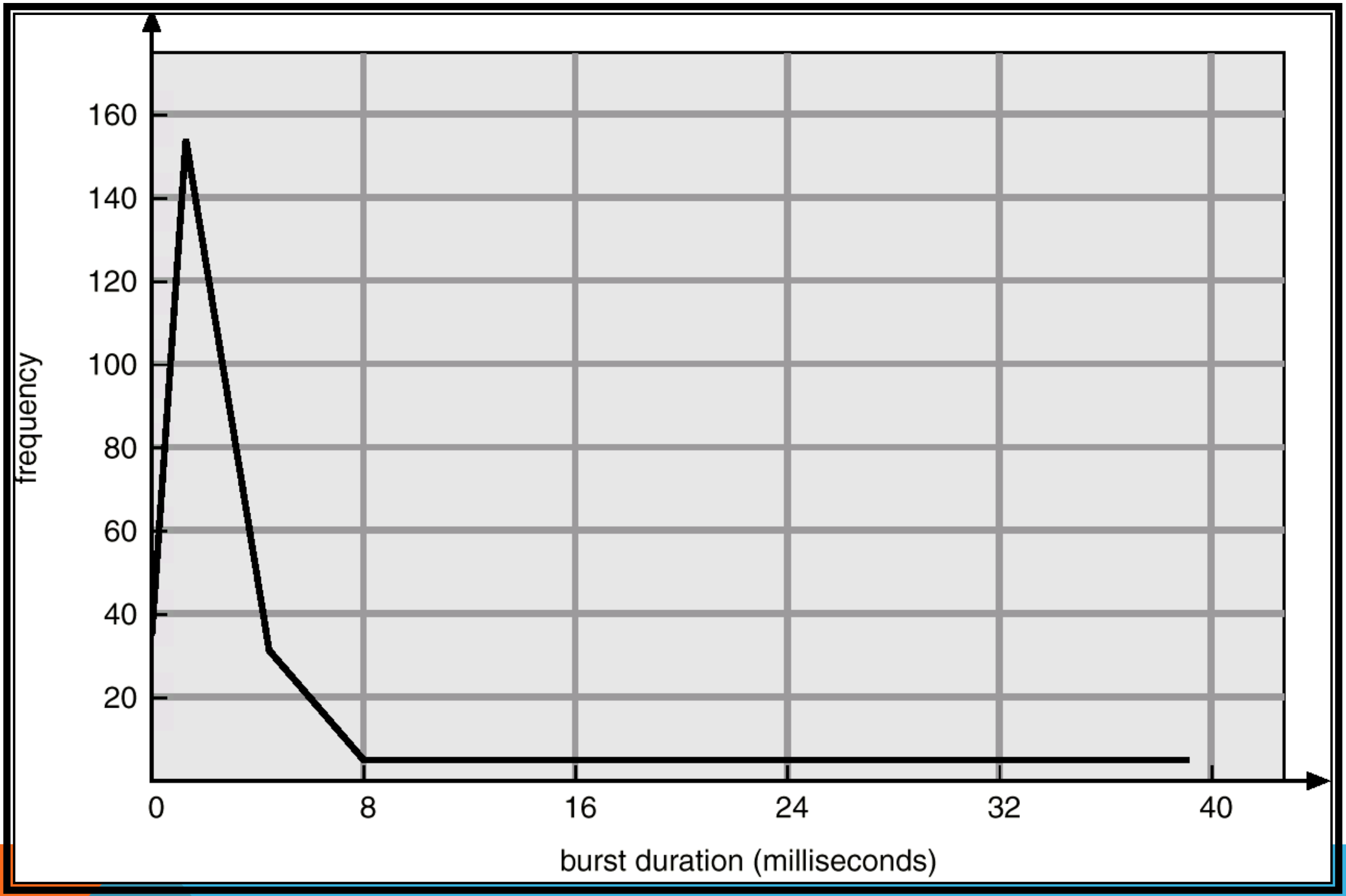
CPU–I/O Burst Cycle – Process execution consists of a *cycle* of CPU execution and I/O wait.

CPU burst distribution

ALTERNATING SEQUENCE OF CPU AND I/O BURSTS



HISTOGRAM OF CPU-BURST TIMES



CPU SCHEDULER

Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.

CPU scheduling decisions may take place when a process:

1. Switches from running to waiting state.
2. Switches from running to ready state.
3. Switches from waiting to ready.
4. Terminates.

Preemptive: allows a process to be interrupted in the midst of its CPU execution, taking the CPU away to another process

Non- Preemptive: ensures that a process relinquishes control of CPU when it finishes with its current CPU burst

Scheduling under 1 and 4 is *non preemptive*.

All other scheduling is *preemptive.*

DISPATCHER

Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:

- switching context
- switching to user mode
- jumping to the 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

- CPU utilization – keep the CPU as busy as possible
- Throughput – # of processes that complete their execution per time unit
- Turnaround time – amount of time to execute a particular process (finishing time – arrival time)
- Waiting time – amount of time a process has been waiting in the ready queue
- Response time – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

OPTIMIZATION CRITERIA

Max CPU utilization

Max throughput

Min turnaround time

Min waiting time

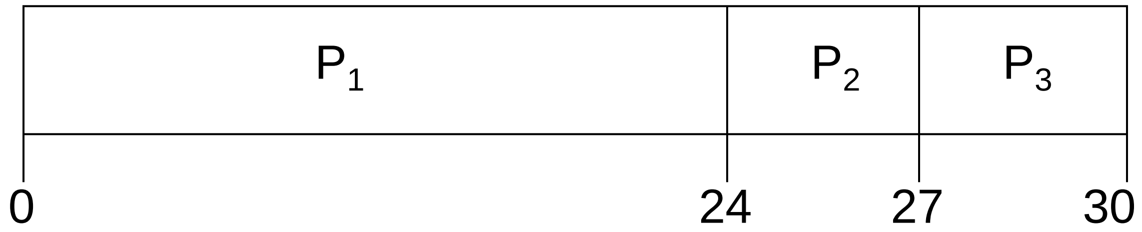
Min response time

FIRST-COME, FIRST-SERVED (FCFS) SCHEDULING

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: $(0 + 24 + 27)/3 = 17$

FCFS SCHEDULING (CONT.)

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

Much better than previous case.

Convoy effect short process behind long process

SHORTEST-JOB-FIRST (SJR) SCHEDULING

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

Two schemes:

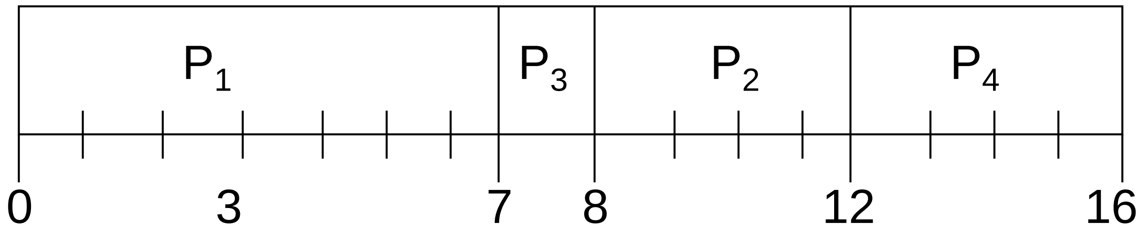
- Non preemptive – once CPU given to the process it cannot be preempted until completes its CPU burst.
- preemptive – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the Shortest-Remaining-Time-First (SRTF).

SJF is optimal – gives minimum average waiting time for a given set of processes.

EXAMPLE OF NON-PREEMPTIVE SJF

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)

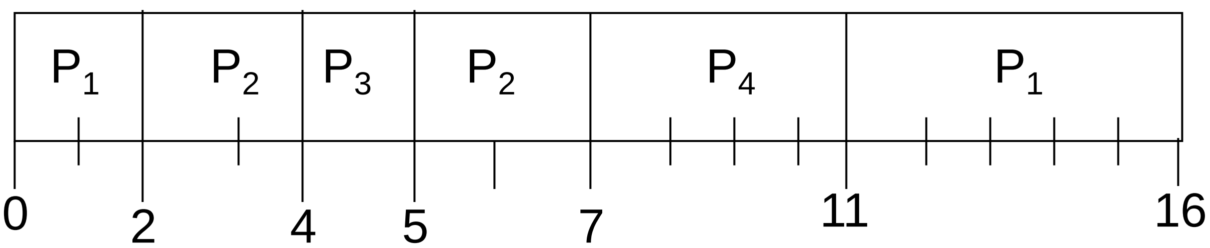


Average waiting time = (0 + 6 + 3 + 7)/4 - 4

EXAMPLE OF PREEMPTIVE SJF

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)



Average waiting time = (9 + 1 + 0 +2)/4 = 3

DETERMINING LENGTH OF NEXT CPU BURST

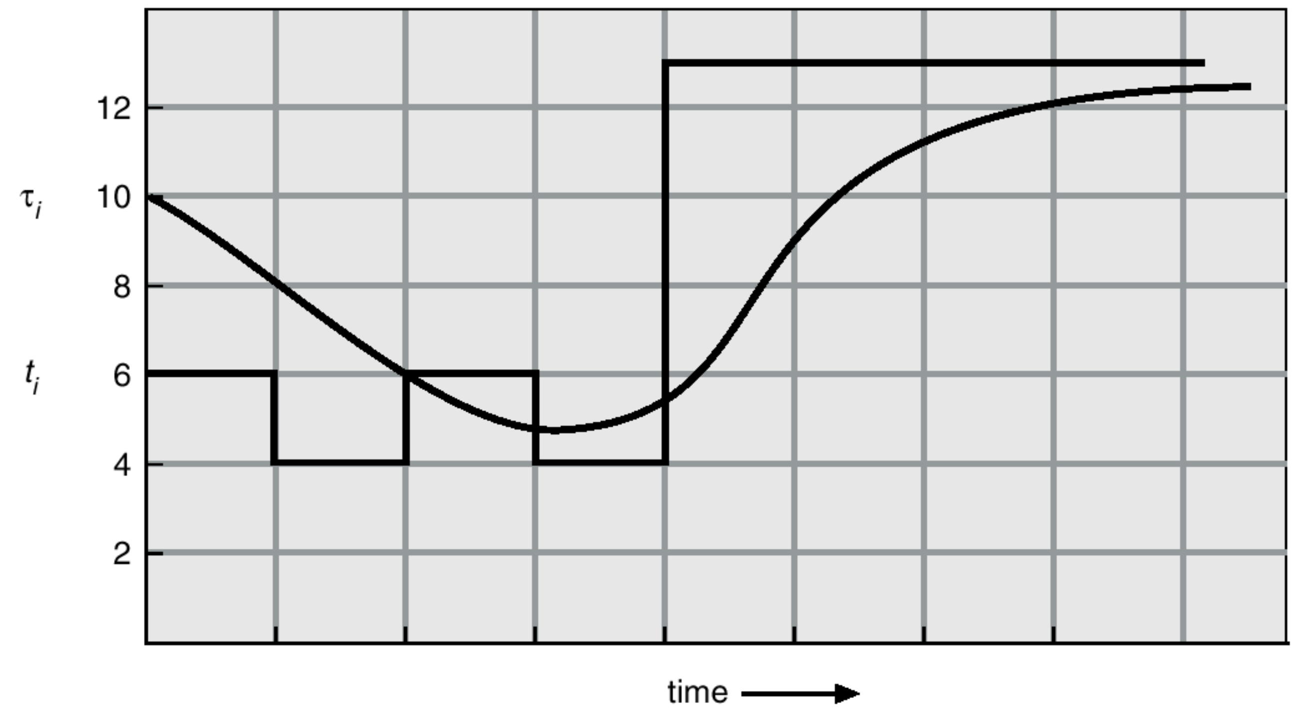
Can only estimate the length.

Can be done by using the length of previous CPU bursts, using exponential averaging.

1. $t_n =$ actual lenght of n^{th} CPU burst
2. $\tau_{n+1} =$ predicted value for the next CPU burst
3. $\alpha, 0 \leq \alpha \leq 1$
4. Define :

$$\tau_{n=1} = \alpha\, t_n + (1 - \alpha)\tau_n .$$

PREDICTION OF THE LENGTH OF THE NEXT CPU BURST



CPU burst (t_i)		6	4	6	4	13	13	13	. . .
"guess" (τ_i)	10	8	6	6	5	9	11	12	. . .

EXAMPLES OF EXPONENTIAL AVERAGING

α =0

- $\tau_{n+1} = \tau_n$
- Recent history does not count.

α =1

- $\tau_{n+1} = t_n$
- Only the actual last CPU burst counts.

If we expand the formula, we get:

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\, \alpha\, t_n - 1 + \ldots$$
$$+ (1 - \alpha)^j\, \alpha\, t_n - 1 + \ldots$$
$$+ (1 - \alpha)^{n=1}\, t_n\, \tau_0$$

Since both α and $(1 - \alpha)$ are less than or equal to 1, each successive term has less weight than its predecessor.

PRIORITY SCHEDULING

A priority number (integer) is associated with each process

The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority).

- Preemptive
- Non preemptive

SJF is a priority scheduling where priority is the predicted next CPU burst time.

Problem ≡ Starvation – low priority processes may never execute.

Solution ≡ Aging – as time progresses increase the priority of the process.

ROUND ROBIN (RR)

Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds. 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.

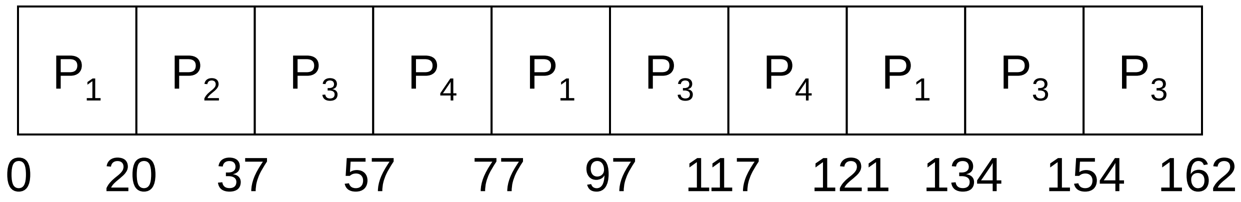
Performance

- q large $\Rightarrow$ FIFO
- q small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high.

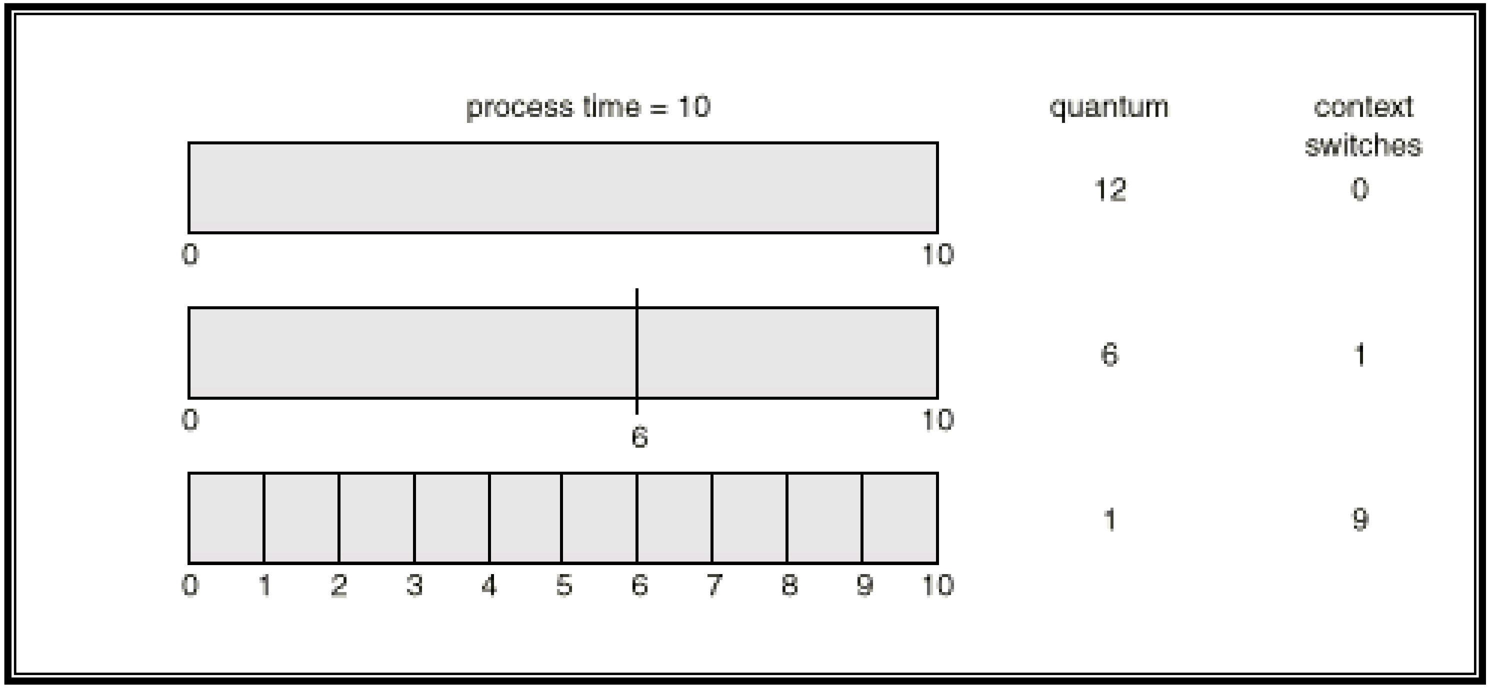
EXAMPLE OF RR WITH TIME QUANTUM = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

The Gantt chart is:



TIME QUANTUM AND CONTEXT SWITCH TIME



SCHEDULING CRITERIA

CPU utilization – keep the CPU as busy as possible

Throughput – # of processes that complete their execution per time unit

Turnaround time – amount of time to execute a particular process (finishing time – arrival time)

Waiting time – amount of time a process has been waiting in the ready queue

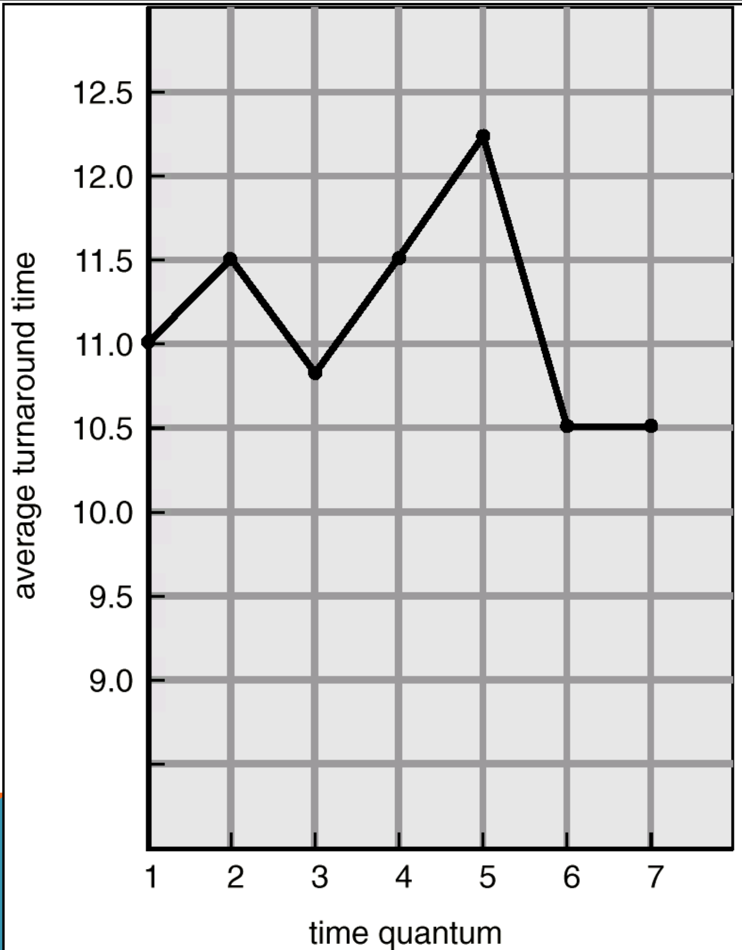
Response time – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

CLASS EXERCISE

Each team works on finding an average turnaround time for a quantum time at 1, 2, 3, 4, 5, 6, 7

Process	Time
p1	6
P2	3
P3	7
p4	1

TURNAROUND TIME VARIES WITH THE TIME QUANTUM



process	time
P_1	6
P_2	3
P_3	1
P_4	7

MULTILEVEL QUEUE

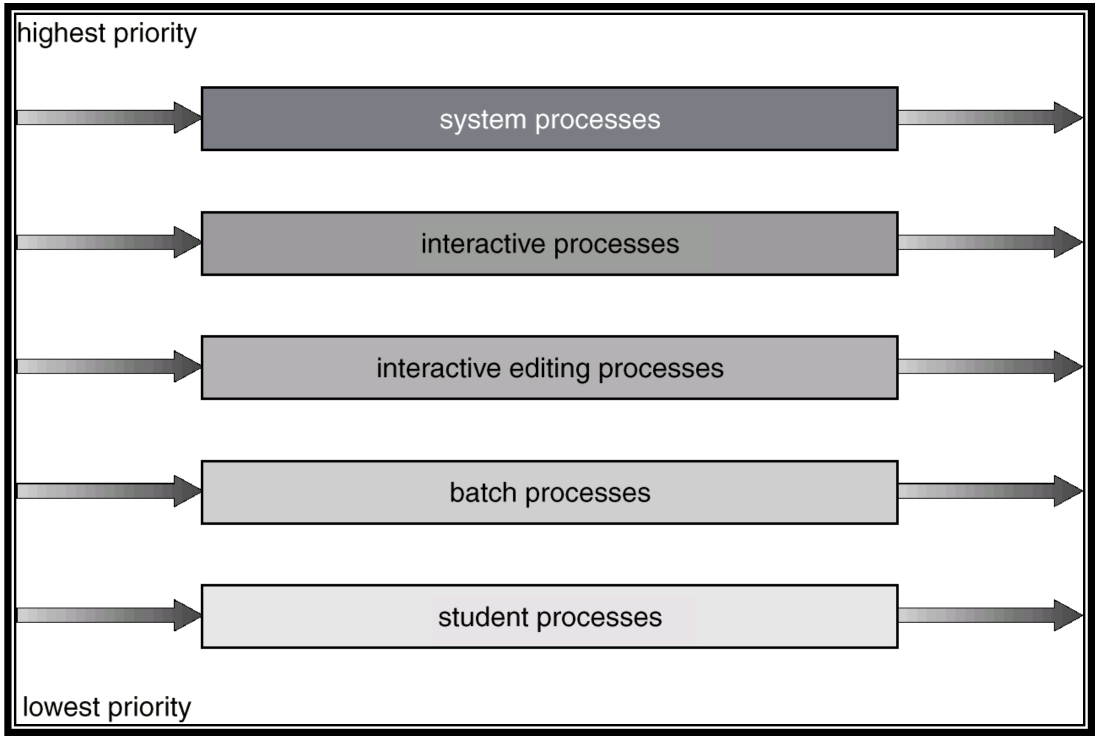
Ready queue is partitioned into separate queues:
foreground (interactive)
background (batch)

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
- 20% to background in FCFS

MULTILEVEL QUEUE SCHEDULING



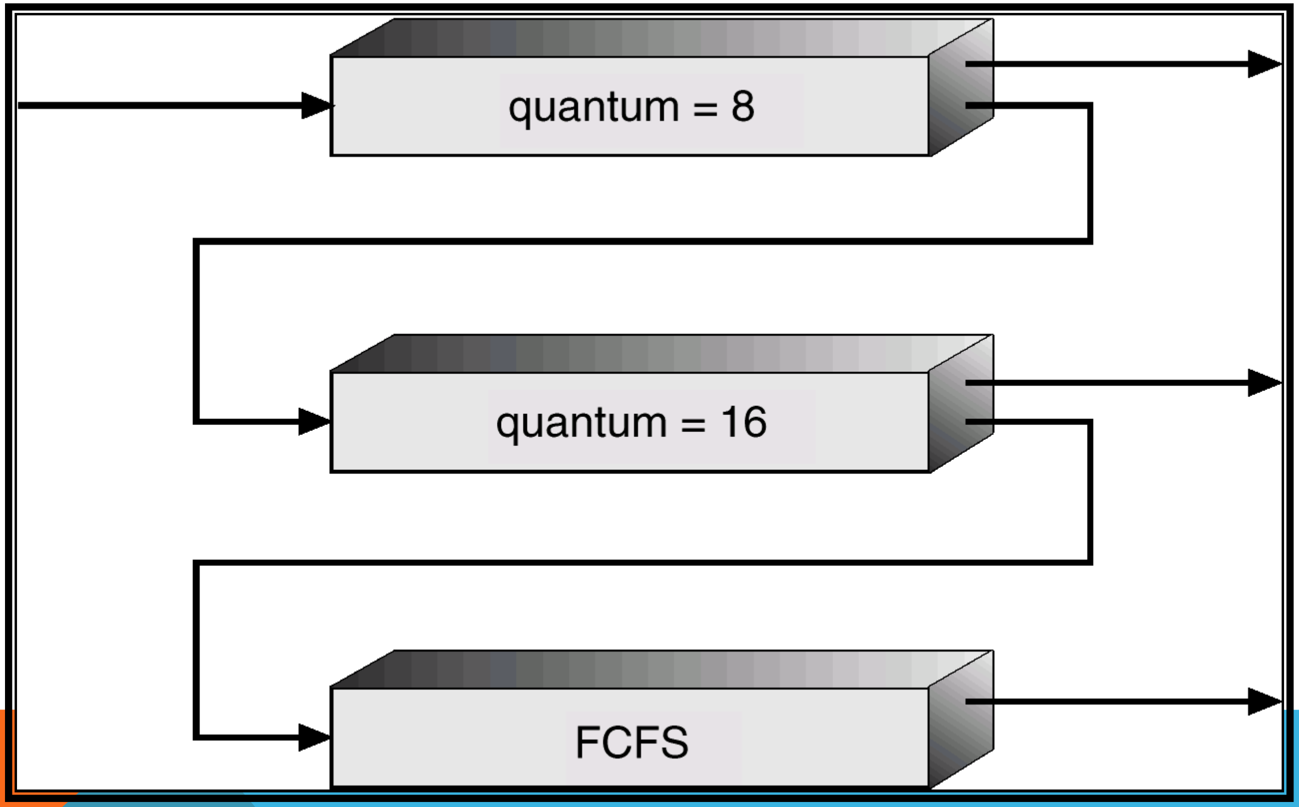
MULTILEVEL FEEDBACK QUEUE

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

MULTILEVEL FEEDBACK QUEUES



EXAMPLE OF MULTILEVEL FEEDBACK QUEUE

Three queues:

- Q_0 – time quantum 8 milliseconds
- Q_1 – time quantum 16 milliseconds
- Q_2 – FCFS

Scheduling

- A new job enters queue Q_0 which is served FCFS. 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.

MULTIPLE-PROCESSOR SCHEDULING

CPU scheduling more complex when multiple CPUs are available.

Homogeneous processors within a multiprocessor.

Load sharing

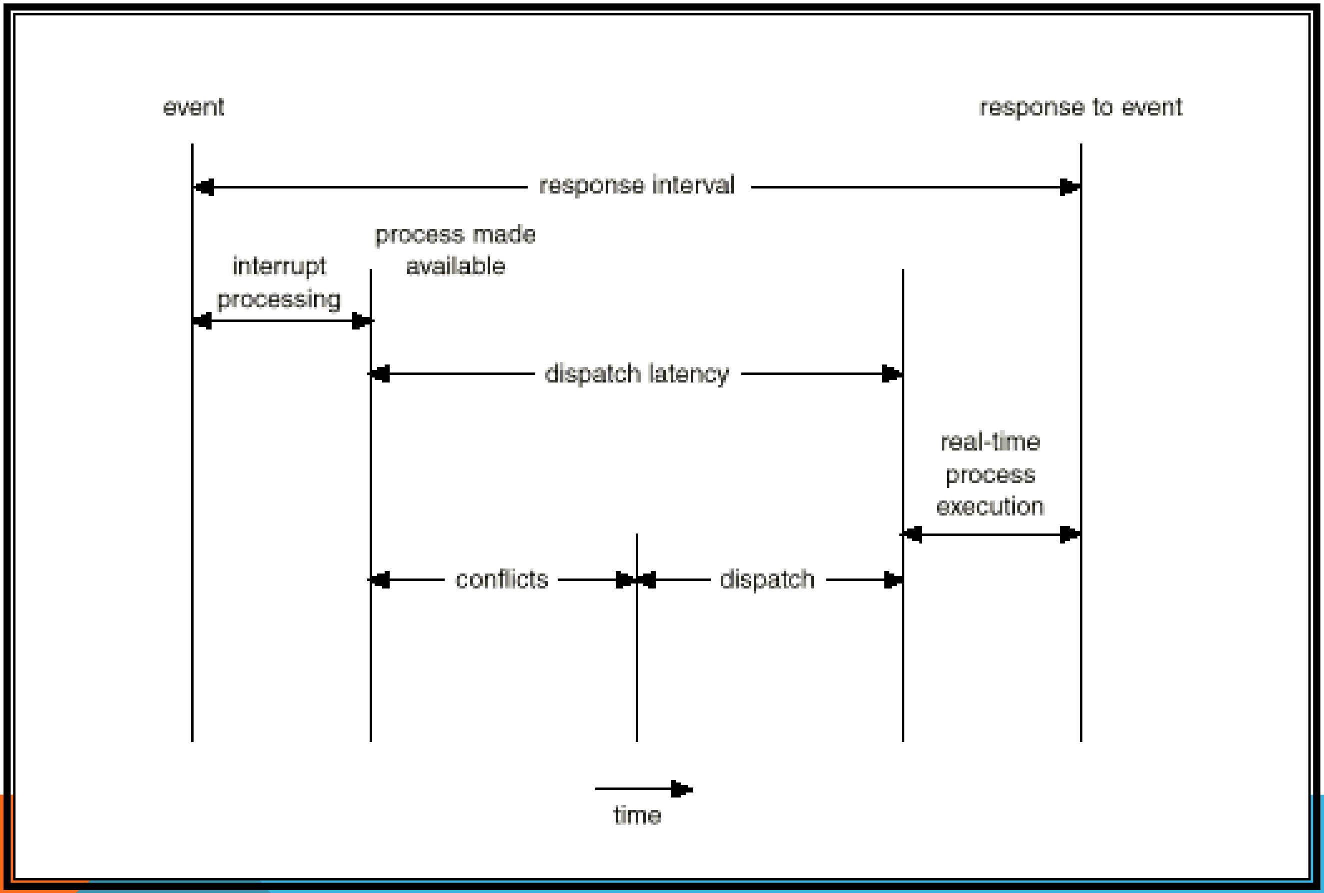
Asymmetric multiprocessing – only one processor accesses the system data structures, alleviating the need for data sharing.

REAL-TIME SCHEDULING

Hard real-time systems – required to complete a critical task within a guaranteed amount of time.

Soft real-time computing – requires that critical processes receive priority over less fortunate ones.

DISPATCH LATENCY



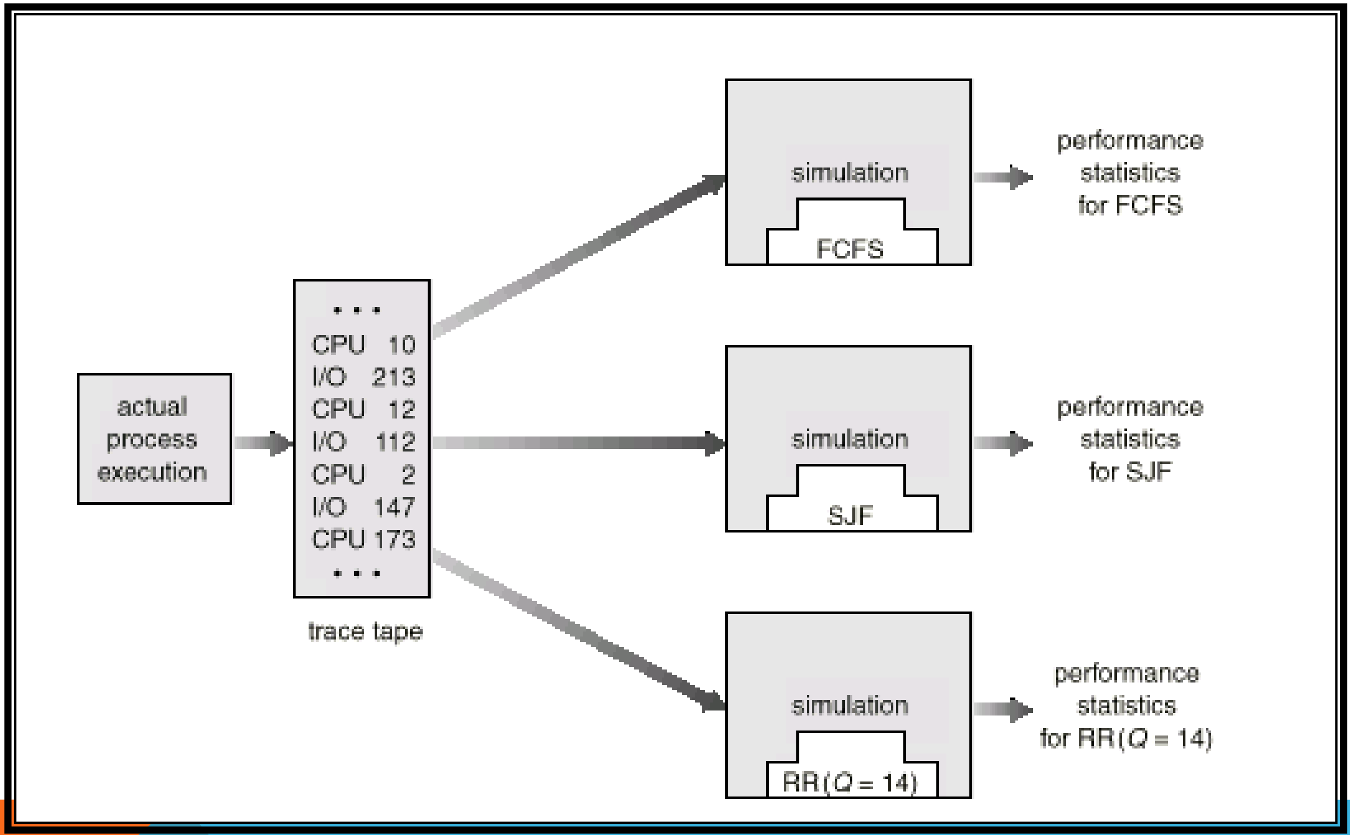
ALGORITHM EVALUATION

Deterministic modeling – takes a particular predetermined workload and defines the performance of each algorithm for that workload.

Queueing models

Implementation

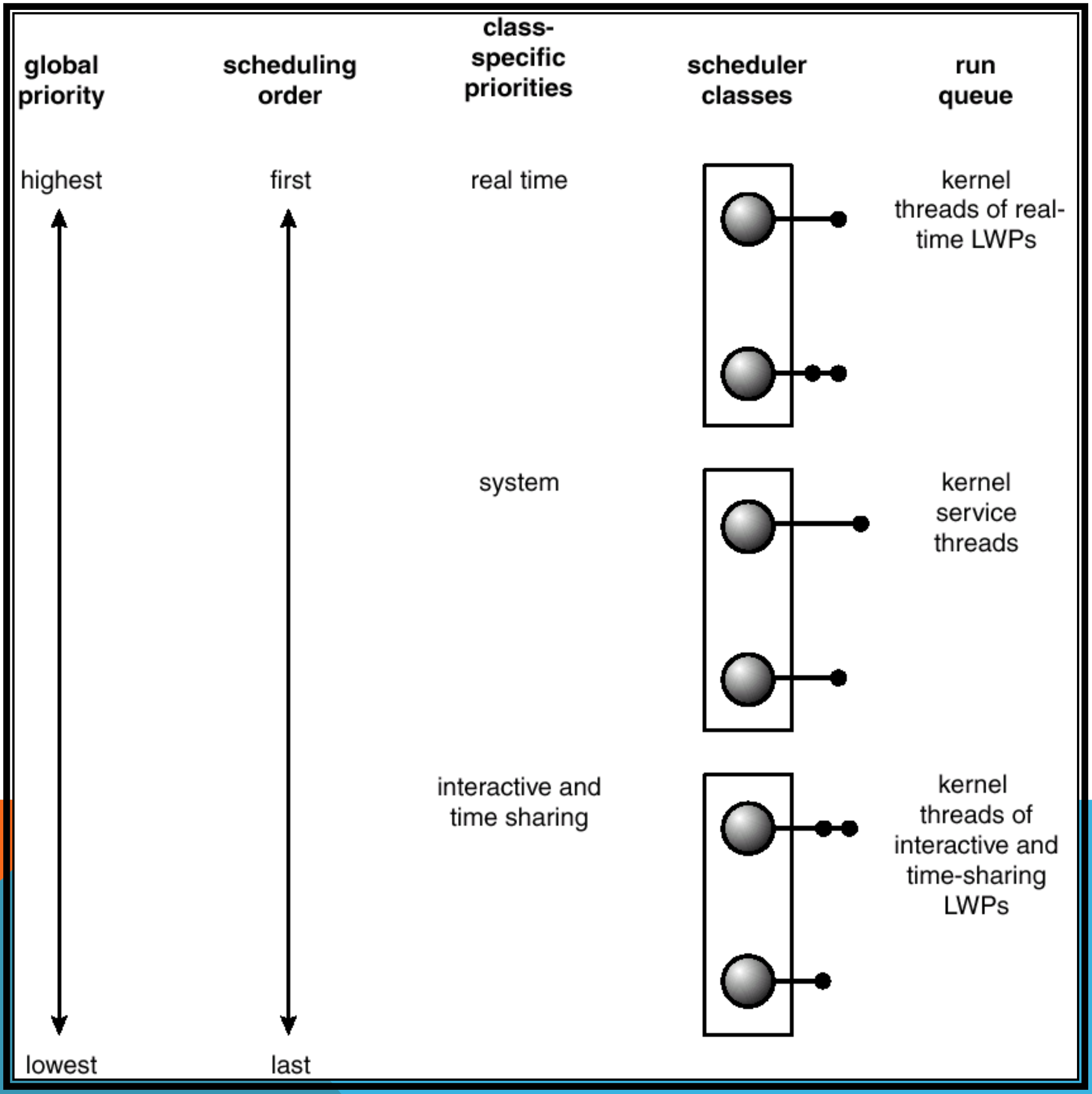
EVALUATION OF CPU SCHEDULERS BY SIMULATION



Linux Process Scheduling

- 2 separate process-scheduling algorithms
  - time-sharing: a prioritized credit-based
  - Soft-real time: FCFS and RR
- only allows processes in a user mode to be preempted.

SOLARIS 2 SCHEDULING



WINDOWS 2000 PRIORITIES

	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1