OPERATING SYSTEMS

Subject code : CSC 404

Subject In-charge

Nidhi Gaur

Assistant Professor

email: nidhigaur@sfit.ac.in

CPU Scheduling

Chapter : CPU Scheduling

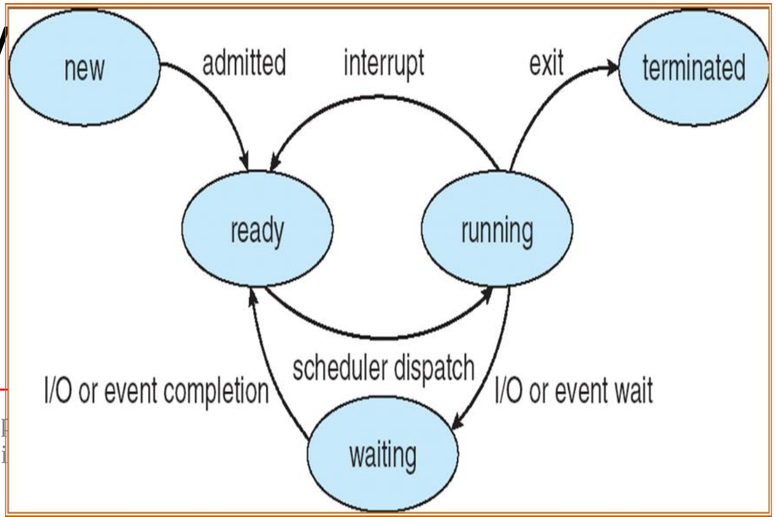
- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms

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 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
- 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** – time elapsed between the submission of a job and its termination
- **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

CPU Scheduling

- TAT = Process completion – Arrival time

  OR = Execution time(CPU Burst) + Waiting time
- Waiting time = Process start time – Arrival time

  OR = TAT – Execution time(CPU Burst)

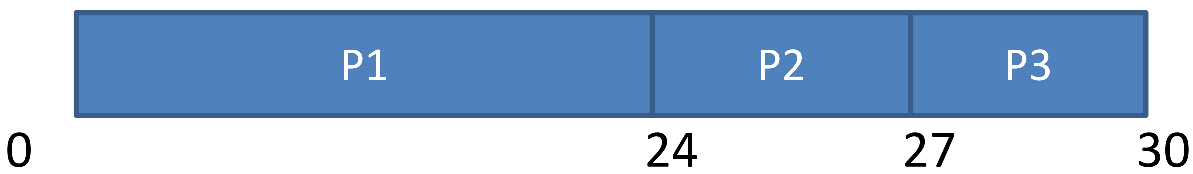
FCFS(First Come First Serve)

- **Non pre-emptive policy**
- Maintains FIFO list of jobs as they arrive.
- Allocates CPU to job at head of the list.
- **Burst time**: for how much time the CPU is allocated to that process.

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 Arrival time is 0ms for all the processes.**
- **The Gantt Chart for the schedule is:**

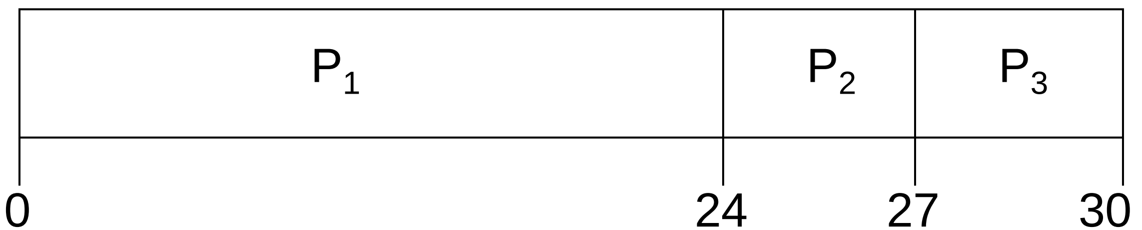


- **Calculate turn around time for each process.**
- **Calculate waiting time for each process.**
- **Calculate average TAT and average waiting time.**

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 Arrival time is 0ms for all the processes.
- **The Gantt Chart for the schedule is:**



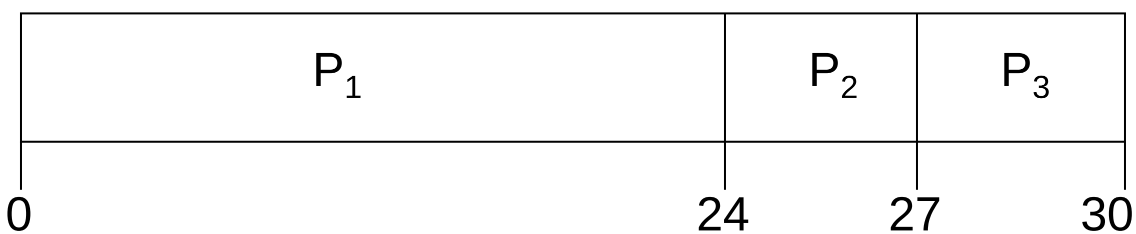
TAT = Process completion – Arrival time

- Turn around time for P1: 24-0 = 24ms
- Turn around time for P2: 27- 0 = 27ms
- Turn around time for P3: 30- 0 = 30 ms. Average TAT??

First-Come, First-Served (FCFS) Scheduling

Process	**Burst Time**
P$_1$	**24**
P$_2$	**3**
P$_3$	**3**

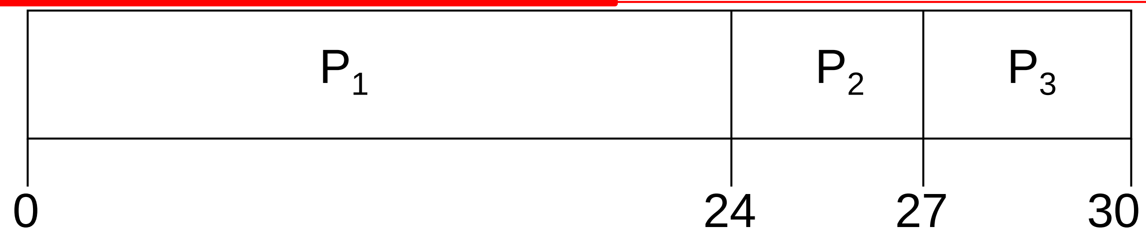
- **Suppose that the processes arrive in the order: P_1 , P_2 , P_3**
  The Arrival time is 0ms for all the processes.
- **The Gantt Chart for the schedule is:**



Waiting time = Process start time – Arrival time

- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17ms

FCFS



PROCESS	CPU TIME/BURST TIME/EXECUTION TIME (in ms)	ARRIVAL TIME(AT)	TURN AROUND TIME(in ms)(completion time -AT)	WAITING TIME(in ms)(Start time- AT)
P1	24	0	24-0 = 24ms	0-0 = 0ms
P2	3	0	27-0 = 27ms	24-0 = 24ms
P3	3	0	30-0 = 30ms	27-0 = 27ms

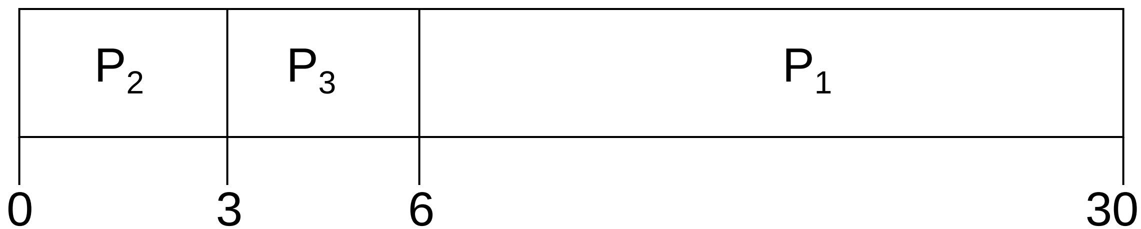
Average TAT = 27ms **Average WT = 17ms**

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

FCFS

Convoy effect in FCFS

P_1	P_2	P_3
0 – 24	24 – 27	27 – 30

There is a **convoy effect** as all the other processes wait for the one big process to get off the CPU. A long CPU-bound job may take the CPU and may force shorter (or I/O-bound) jobs to wait prolonged

FCFS

- Exercise

Process	Burst Time	Arrival Time
P1	10	0
P2	1	0
P3	2	0
P4	1	0
P5	5	0

a) Draw a Gantt chart for the FCFS Scheduling Algorithm

b) Calculate the **turnaround time** for the FCFS

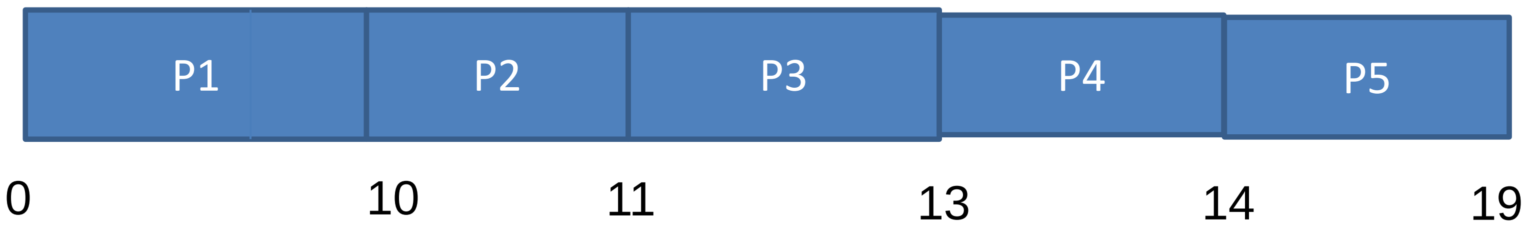
c) Calculate the **average turnaround time**

d) Calculate **waiting time**.

e) Calculate **average waiting time.**

FCFS

- GANTT CHART



PROCESS	ARRIVAL TIME(in ms)	TAT TIME (in ms)	WAITING TIME(in ms)
P1	0	10	0
P2	0	11	10
P3	0	13	11
P4	0	14	13
P5	0	19	14

FCFS

ANS: ATAT: 13.4ms

AWT: 9.6ms

Shortest-Job-First (SJF) 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 it completes its CPU burst
- **Pre-emptive** – 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

Other name of this algorithm is **Shortest-Process-Next (SPN).**

SJF

When the CPU is available, it is assigned to the process that has the smallest next CPU burst. **If the next CPU bursts of two processes are the same, FCFS scheduling is used.**

P_1	P_2	P_3
0 – 24	24 – 27	27 – 30

The SJF scheduling algorithm gives the minimum average waiting time for a given set of processes.

- Moving a short process before a long one decreases the waiting time of the short process more than it increases the waiting time of the long process.

Consequently, the average waiting time decreases

P_2	P_3	P_1
0 – 3	3 – 6	6 – 30

SJF

- Exercise

PROCESS	ARRIVAL TIME	EXECUTION TIME
P1	0	5
P2	2	4
P3	3	7
P4	5	6

a) **Draw a Gantt chart** for the the SJF Scheduling Algorithm

b) Calculate the **turnaround time** for the SJF

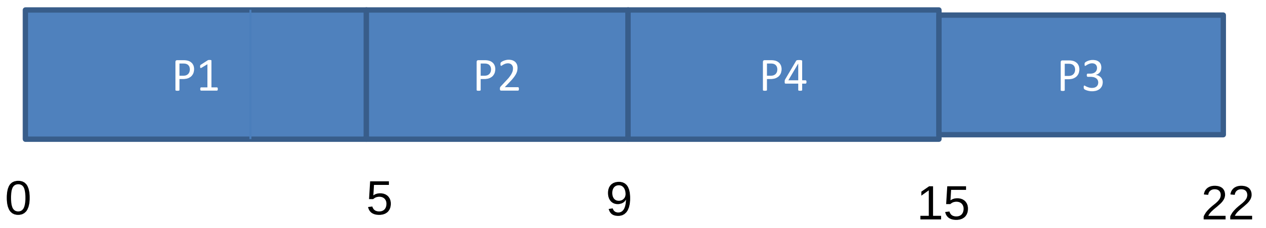
c) Calculate the **average turnaround time**

d) Calculate **waiting time**.

e) Calculate **average waiting time.**

SJF(non-preemptive)

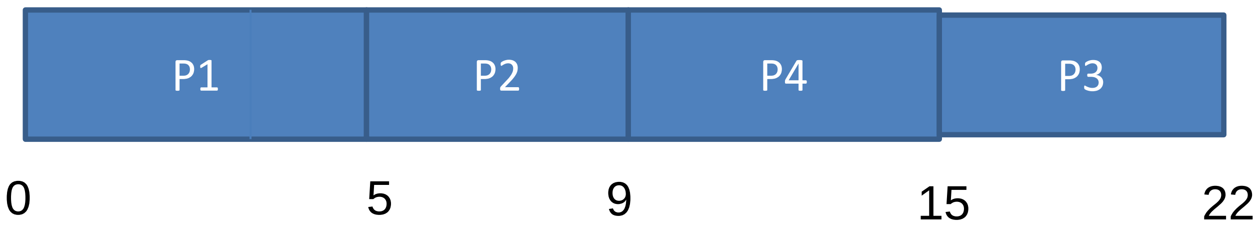
- GANTT CHART



PROCESS	ARRIVAL TIME	EXECUTION TIME
P1	**0**	**5**
P2	**2**	**4**
P3	**3**	**7**
P4	**5**	**6**

SJF

- GANTT CHART



PROCESS	ARRIVAL TIME	EXECUTION TIME	TAT (Completion -AT)	WAITING TIME(Start-AT)
P1	0	5	5-0 =5	0-0 = 0
P2	2	4	9-2= 7	5-2 = 3
P3	3	7	22-3 = 19	15-3 = 12
P4	5	6	15-5 = 10	9-5 = 4

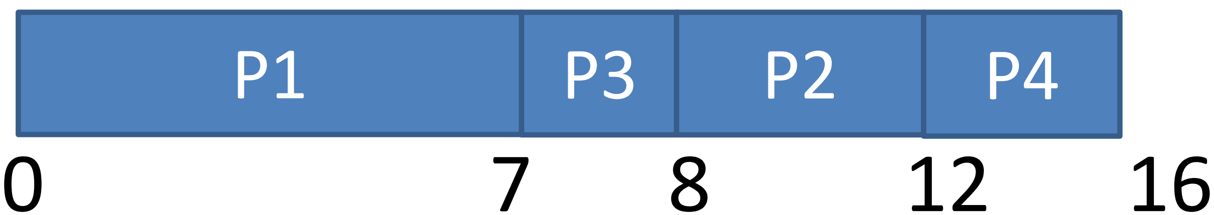
SJF

ATAT= (5+7+19+10)/ 4 = 10.25 ms

AWT= (0+3+12+4)/4 = 4.75 ms

Solve for Non-Preemptive SJF

PROCESS	ARRIVAL TIME	BURST TIME
P1	0	7
P2	2	4
P3	4	1
P4	5	4



- Draw Gantt chart.
- Calculate Average waiting time and turnaround time?
- 4ms and 8 ms respectively

SJF

- Exercise

Process	Burst Time	Arrival Time
P1	10	0
P2	1	0
P3	2	0
P4	1	0
P5	5	0

a) Draw a Gantt chart for the SJF Scheduling Algorithm

b) Calculate the **turnaround time** for the SJF

c) Calculate the **average turnaround time**

d) Calculate **waiting time**.

e) Calculate **average waiting time.**

Ans : ATAT: 7ms ; AWT: 3.2ms

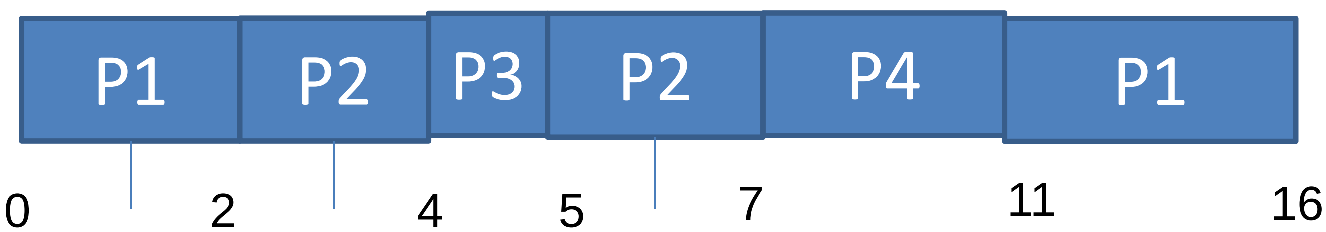
Solve for Preemptive SJF(SRTF)

PROCESS	ARRIVAL TIME	BURST TIME
P1	0	7
P2	2	4
P3	4	1
P4	5	4

- Draw Gantt chart.
- Calculate Average waiting time and turnaround time?

Solve for Preemptive SJF(SRTF)

PROCESS	ARRIVAL TIME	BURST TIME
P1	0	7
P2	2	4
P3	4	1
P4	5	4



Solve for Preemptive SJF(SRTF)

P1	P2	P3	P2	P4	P1

0 2 4 5 7 11 16

PROCESS	ARRIVAL TIME	BURST TIME/EXECUTION TIME	TAT= (COMPLETION TIME-AT)	WT=(TAT-EXECUTION TIME)
P1	0	7	16-0 = 16	9
P2	2	4	7-2 = 5	1
P3	4	1	5-4 = 1	0
P4	5	4	11- 5 = 6	2

ATAT = 7ms AWT = 3 ms

Solve for SRTF

- EXERCISE

Process	Burst Time	Arrival Time(all in ms)
P1	9	0
P2	5	1
P3	3	2
P4	4	3

a) Draw a Gantt chart

b) Calculate **average TAT**

c) Calculate **average waiting time.**

Solve for Preemptive SJF(SRTF)

P1	P2	P3	P2	P4	P1

0 1 2 5 9 13 21

PROCESS	ARRIVAL TIME	BURST TIME/EXECUTION TIME	TAT=(COMPLETION TIME-AT)	WT=(TAT-EXECUTION TIME)
P1	0	9	21	12
P2	1	5	8	3
P3	2	3	3	0
P4	3	4	10	6

ATAT = 10.5ms AWT = 5.25 ms

SJF

ADVANTAGES

- provably optimal for minimizing average waiting time.
- helps in keeping I/O devices busy.

DISADVANTAGES

- Not practical as we can’t predict future CPU burst time.
- Long jobs may never be scheduled.

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

Priority Scheduling

- Exercise

Process	Burst Time	Priority	Arrival Time
P1	10	3	0
P2	1	1	0
P3	2	4	0
P4	1	2	0
P5	5	2	0

a) Draw a Gantt chart
b) Calculate **waiting time**.
c) Calculate **average waiting time.**

Solve it for non premptive priority scheduling algorithm

Solve for PRIORTY scheduling(non pre emptive)

P2	P4	P5	P1	P3

0 1 2 7 17 19

PROCESS	ARRIVAL TIME	BURST TIME/EXECUTION TIME	PRIORITY NUMBER	TAT= (COMPLETION TIME-AT)	WT=(TAT-EXECUTION TIME)
P1	0	10	3	17	7
P2	0	1	1	1	0
P3	0	2	4	19	17
P4	0	1	2	2	1
P5	0	5	2	7	2

ATAT = 9.2ms AWT = 5.4 ms

Solve for Priority scheduling(pre emptive)

PROCESS	ARRIVAL TIME	BURST TIME	PRIORITY
P1	0	5	2
P2	2	4	1
P3	3	7	3
P4	5	6	4

- Draw Gantt chart.
- Calculate Average waiting time and turnaround time?

Solve for PRIORITY scheduling(pre emptive)

P1	P2	P1	P3	P4

0 2 6 9 16 22

PROCESS	ARRIVAL TIME	BURST TIME	PRIORITY
P1	0	5	2
P2	2	4	1
P3	3	7	3
P4	5	6	4

Solve for PRIORITY scheduling(pre emptive)

P1	P2	P1	P3	P4

0 2 6 9 16 22

PROCESS	ARRIVAL TIME	BURST TIME/EXECUTION TIME	PRIORITY NUMBER	TAT=(COMPLETION TIME-AT)	WT=(TAT-EXECUTION TIME)
P1	0	5	2	9	4
P2	2	4	1	4	0
P3	3	7	3	13	6
P4	5	6	4	17	11

ATAT = 10.75ms AWT = 5.25 ms

Round Robin

- The requirement of scheduling is different for multi-user time sharing systems.
- **Response time** is the most important objective.
- If each process gets same processor time, the response will be equally good for all the processes and neither the short nor long process will suffer from starvation.

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 $\Rightarrow$ MORE context switching
  - *q* must be large with respect to context switch, otherwise overhead is too high

Round Robin

- Sometimes a process may finish its execution before the time slice expires. Will the timer complete its full time slice and then send the interrupt signal??
- No, it does not happen that way.
- There is time wastage in this design if a timer completes its time slice, even when a process has finished earlier than the time slice.
- Therefore, the design is such that **whenever a process finishes before the time slice expires, the timer will stop and send the interrupt signal, so that the next process can be scheduled.**

ROUND ROBIN

- Designed for time-sharing systems.
- Similar to FCFS but pre-emption is added
- In this, assign a small unit of time quantum(or time slice) for each process.
- Ready queue is treated as circular queue.
- Arriving jobs are placed at the end.
- Dispatcher selects first job in queue & runs until completion of CPU burst or until time quantum expires.
- If quantum expires job is again placed at the end.

Solve for Round Robin

- EXERCISE

Process	**Burst Time**	**Arrival Time(all in ms)**
P1	**9**	**0**
P2	**5**	**1**
P3	**3**	**2**
P4	**4**	**3**

time quantum q =2 ms

a) Draw Gantt Chart

b) Calculate **average TAT**

c) Calculate **average waiting time.**

RR

P1	P2	P3	P4	P1	P2	P3	P4	P1	P2	P1	P1

0 2 4 6 8 10 12 13 15 17 18 20 21

PROCESS	ARRIVAL TIME	BURST TIME/EXECUTION TIME	TAT=(COMPLETION TIME-AT)	WT=(TAT-EXECUTION TIME)
P1	0	9	21	12
P2	1	5	17	12
P3	2	3	11	8
P4	3	4	12	8

ATAT= 15.25ms AWT = 10 ms

Solve for Round Robin

- EXERCISE

Process	**Burst Time**	**Arrival Time(all in ms)**
P1	**9**	**0**
P2	**5**	**1**
P3	**3**	**2**
P4	**4**	**3**

q = 5 ms

a) Draw Gantt Chart

b) Calculate **average TAT**

c) Calculate **average waiting time.**

RR

P1	P2	P3	P4	P1

0 5 10 13 17 21

ATAT = 13.75 ms
AWT = 8.5 ms

ROUND ROBIN

- If there is n number of processes in a system and q is the time quantum , what is the maximum time a process needs to wait for its execution?

W =(n-1)* q

ROUND ROBIN

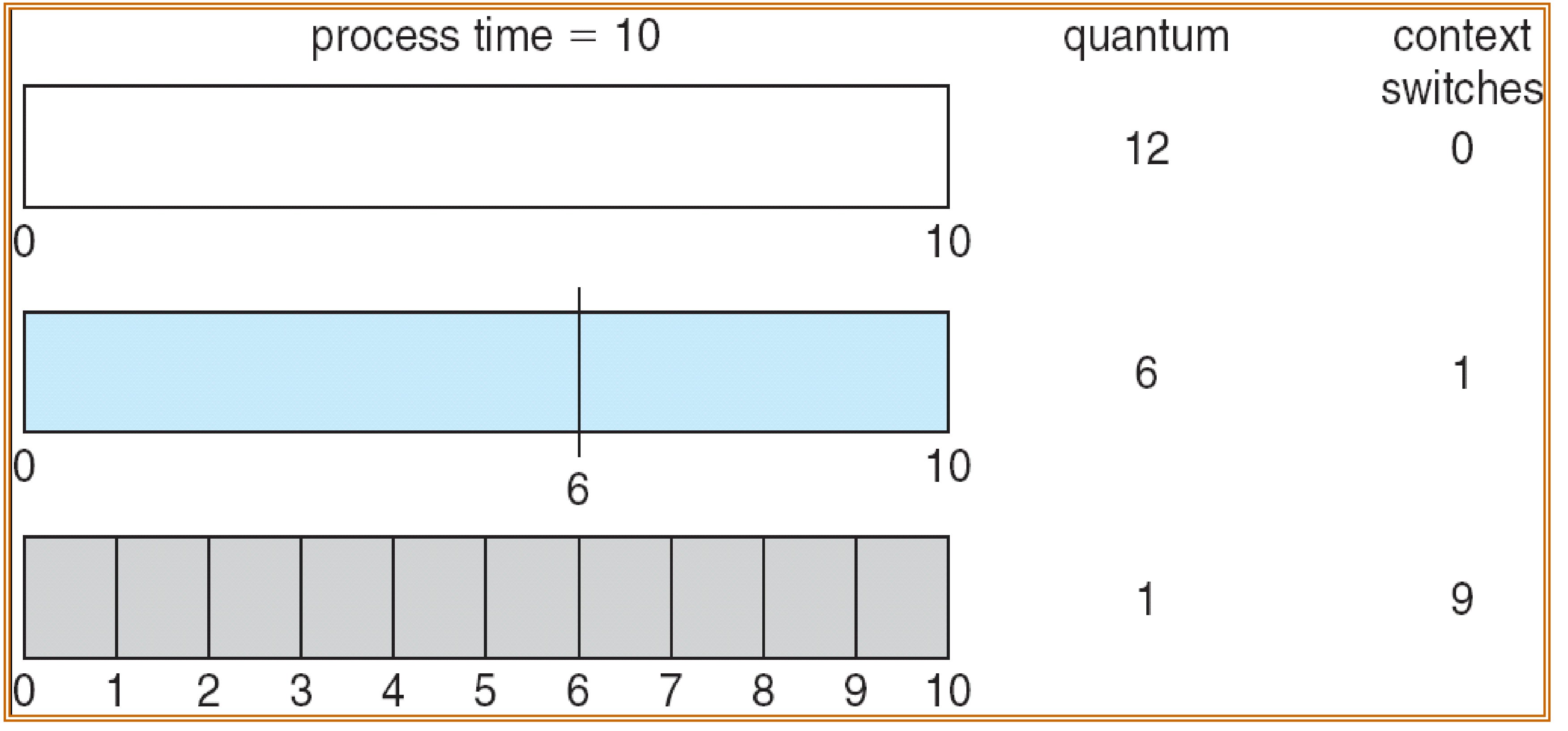
ADVANTAGE:

- Jobs get fair share to CPU.
- Shortest jobs finish relatively quickly.

DISADVANTAGE:

- Poor average waiting time with similar job length.e.g.10 jobs, each requiring 10 time slice, all complete after 100 time slices.
- Performance depends on length of time slice.

Time Quantum and Context Switch Time

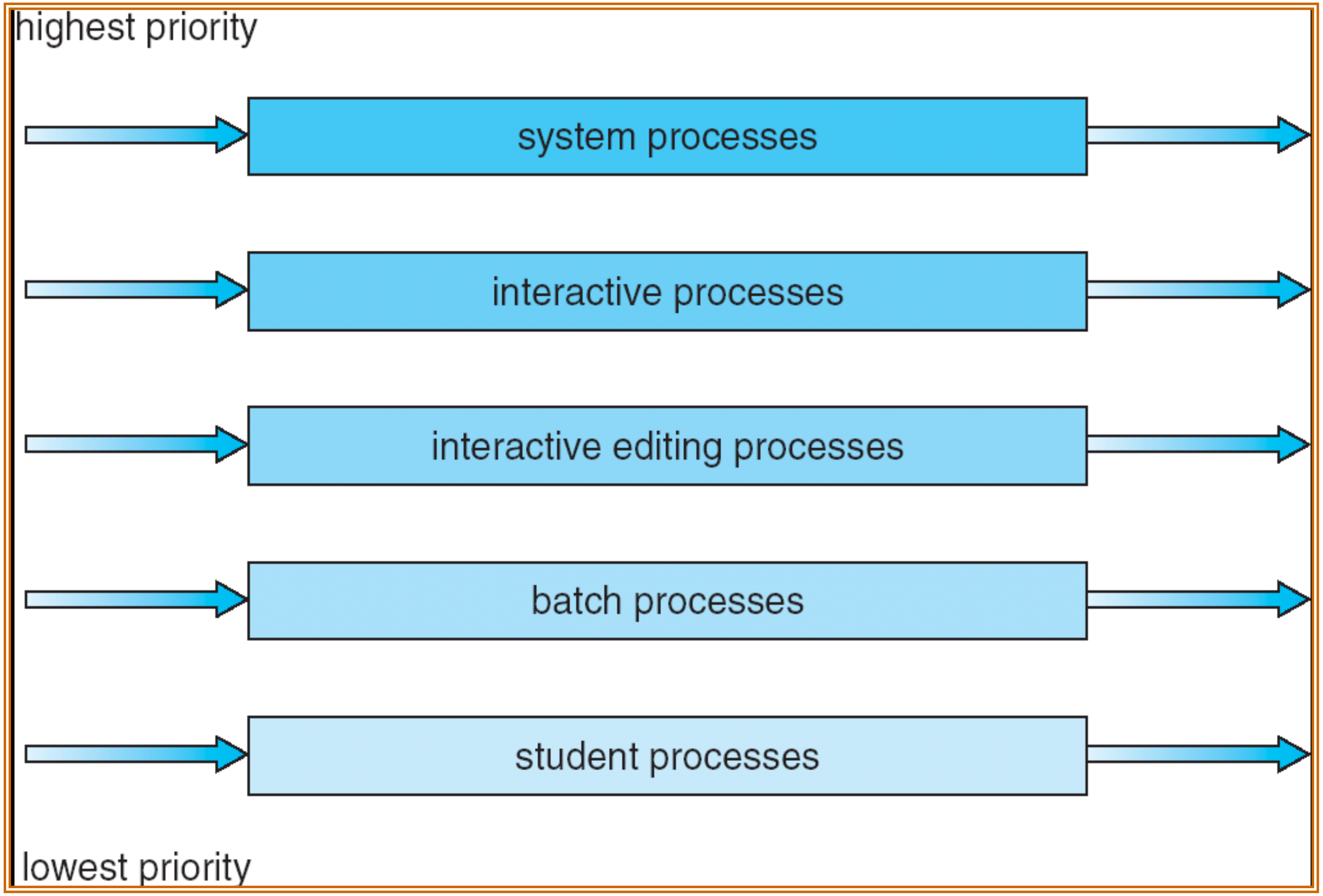


End of Chapter

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



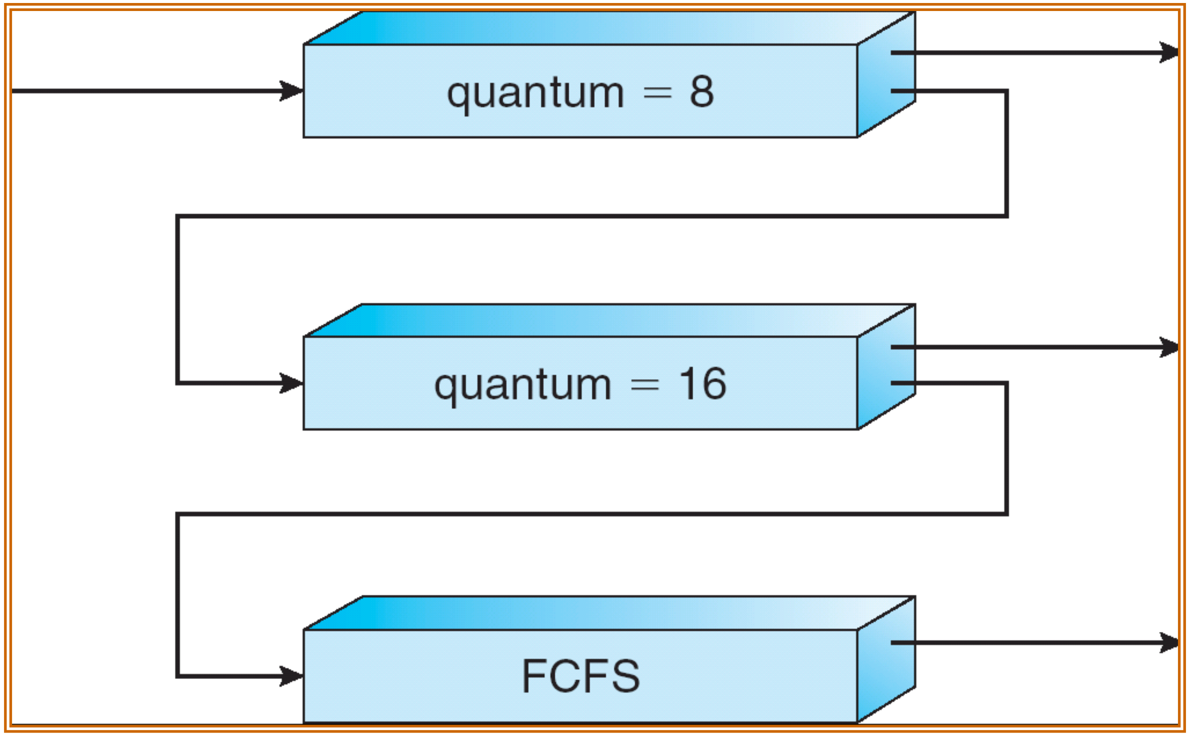
Multilevel Feedback Queue

- 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

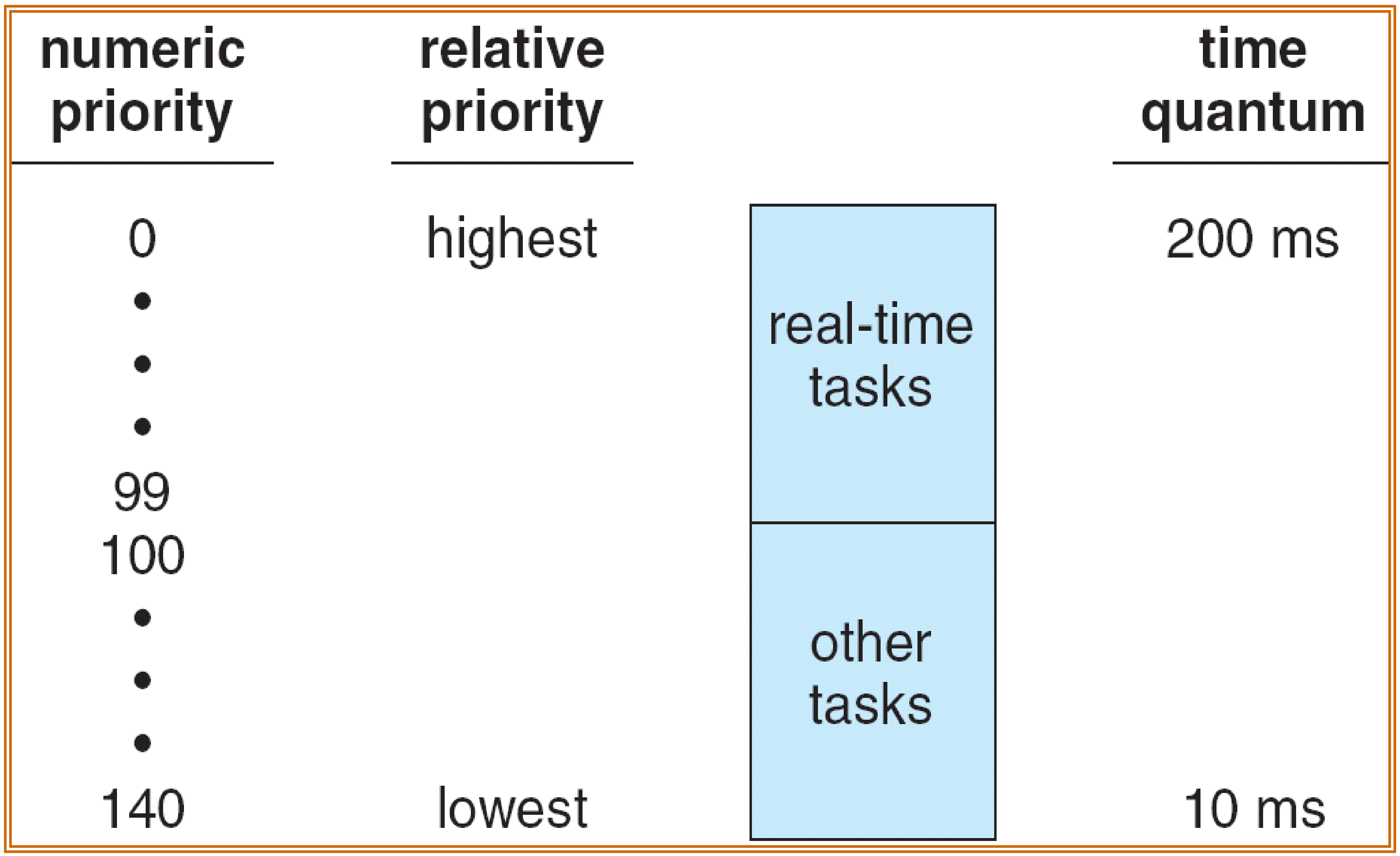
Example of Multilevel Feedback Queue

- 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 RR. 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 RR and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q_2.

Multilevel Feedback Queues



The Relationship Between Priorities and Time-slice length



Thread Priorities

Priority	**Comment**
Thread.MIN_PRIORITY Thread Priority	Minimum
Thread.MAX_PRIORITY Thread Priority	Maximum
Thread.NORM_PRIORITY Thread Priority	Default

Priorities May Be Set Using setPriority() method:

setPriority(Thread.NORM_PRIORITY + 2);