

Operating Systems

Prepared by Dr. Alaa Eldeen Ahmed

IT263 / CYB510
CPU Scheduling

Lecture 4



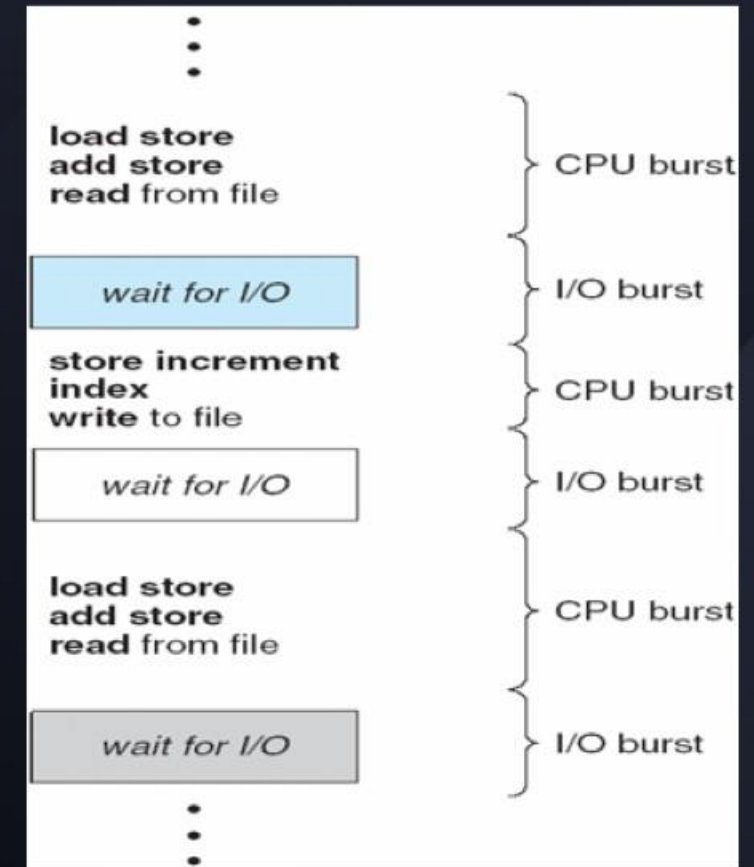
- Topics To Be Covered:
 - To introduce CPU scheduling, which is the basis for multiprogram on operating systems
 - To describe various CPU-scheduling algorithms
 - To discuss evaluation criteria for selecting a CPU-scheduling algorithm for a particular system

- Topics To Be Covered:
 - To introduce CPU scheduling, which is the basis for multiprogram on operating systems
 - To describe various CPU-scheduling algorithms
 - To discuss evaluation criteria for selecting a CPU-scheduling algorithm for a particular system

- The Objective is to maximize CPU utilization obtained with multiprogramming
- CPU utilization is defined as the amount of load handled by individual processor cores to run various programs on a computer.
- A process is executed until it must wait, typically for the completion of some I/O request.
- In a simple computer system, the CPU then sits idle. All this is wasted: no useful work is accomplished.

- Several processes are kept in memory at one time.
- When one process has to wait, the operating system takes the CPU away from that process and gives the CPU to another process and this pattern continues.

- CPU-I/O Burst Cycle ,Process execution consists of a cycle of **CPU execution** and **I/O wait**.
- Processes alternate between those two states
- CPU Burst is the time when the process is being executed in CPU.
- I/O burst is the time when a process waits for I/O to complete.

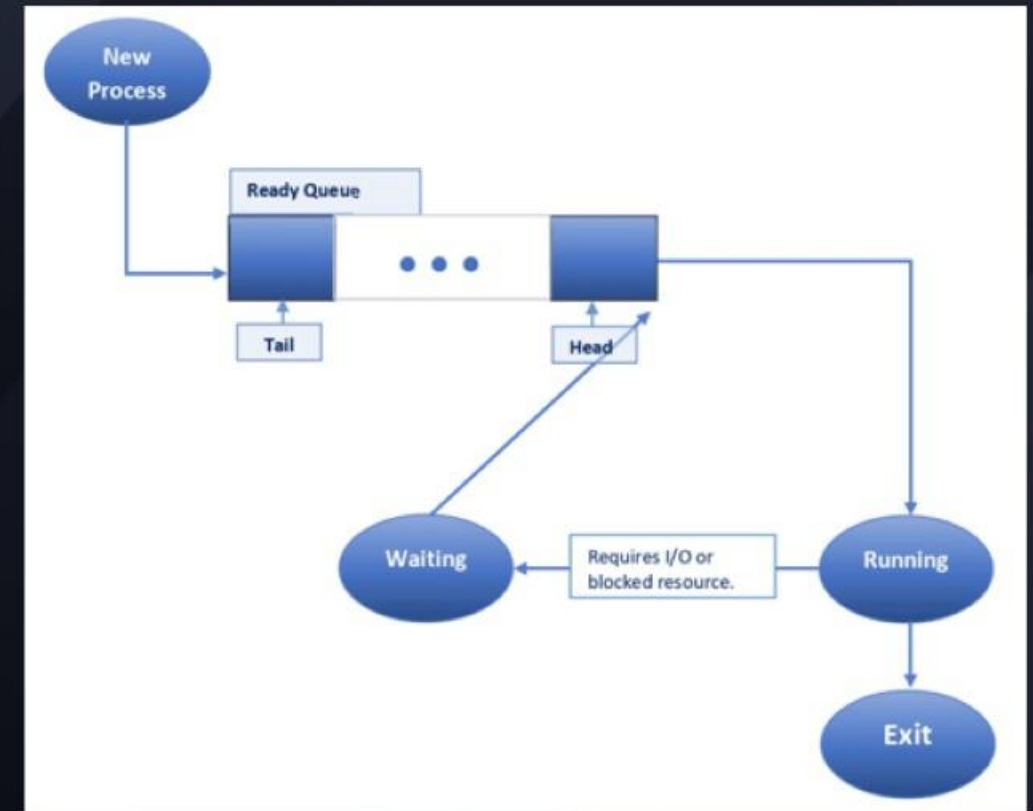


- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them
- CPU scheduling decisions may take place when a process:
 - Switches from running to waiting state
 - Switches from running to ready state
 - Switches from waiting to ready
 - Terminates

Lecture 04

Preemptive and Non Preemptive Scheduling

- For situations 1 and 4, is called **non-preemptive** scheduling
- Once a process is allocated the CPU, it does **not** leave unless: it has to wait, e.g., for I/O request or for a child to terminate or terminates
- We can't interrupt it while it's running.



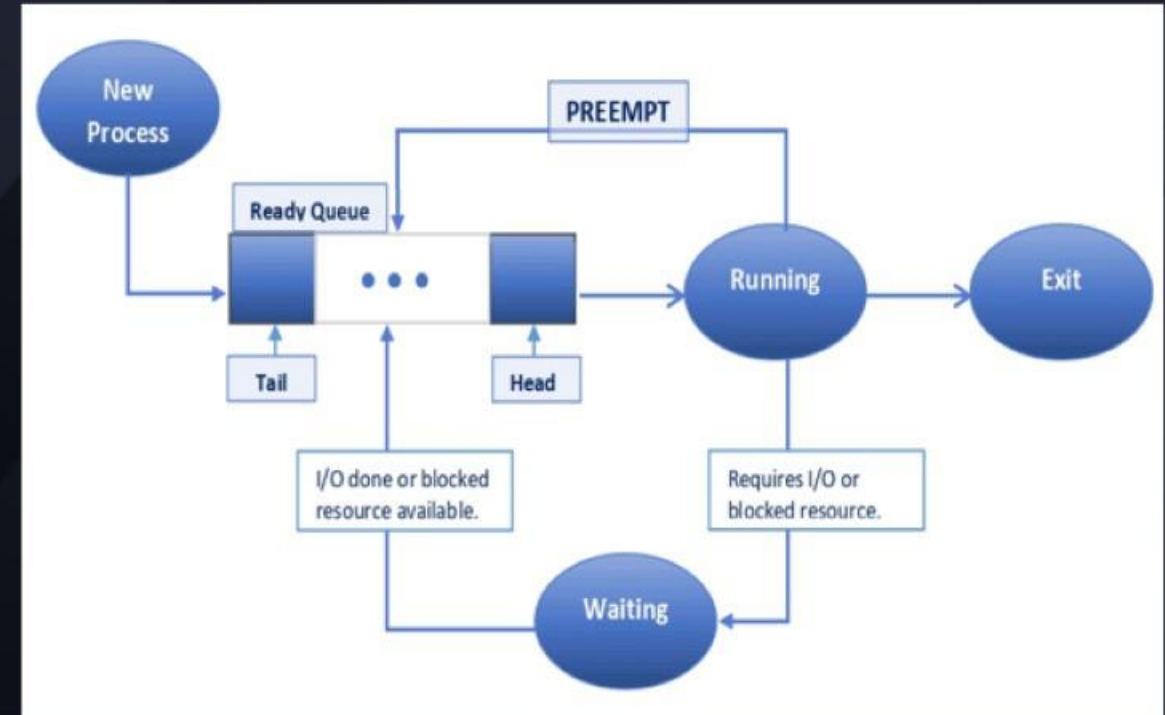
- Advantages:
- 1- Non-preemptive scheduling has minimal computational overhead and is very simple to understand and implement.
- 2- it has higher throughput (i.e., the volume of processes that the CPU executes in a certain length of time).

- Disadvantages:
 - 1- The average response time (time between arrival and first scheduled execution) in non-preemptive scheduling is definitely poor.
 - 2- The difficulty in handling priority scheduling. This is so because a process cannot be preempted.
 - 3- Non-preemption is one of the necessary preconditions for deadlock. If process "A" holds the resource that process "B" needs to continue and vice-versa, and preemption isn't possible, then deadlock is bound to happen.

Lecture 04

Preemptive and Non Preemptive Scheduling

- For situations 2 and 3 , is called **preemptive** scheduling
- OS can force (preempt) a process from CPU at **anytime**
- We can Say, to allocate CPU to another higher-priority process



- **Advantages:**

- 1- It ensures no process can monopolize the CPU
- 2- Improves average response time
- 3- Allow the opportunity to implement priority scheduling.
- 4- Avoids deadlocks

- **Disadvantages:**

- 1- Extra time is required for interrupting a running process, switching between contexts, and scheduling newly arriving processes
- **2- Process starvation** can occur. That's a situation where a low-priority process waits indefinitely or for a long time due to the continuous arrival of higher-priority processes

- Which is harder to implement? and why?
 - Preemptive is harder: Need to maintain **consistency** of data shared between processes, and more importantly, kernel data structures (e.g., I/O queues)
 - Think of a preemption while kernel is executing a sys call on behalf of a process (many OSs, wait for sys call to finish)

- Dispatcher 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
 - Maximize
- **Throughput** – # of processes that complete their execution per time unit
 - Maximize
- **Turnaround time** – amount of time to execute a particular process (time from submission to termination)
- **Turn Around Time) $TAT = Completion\ time(CT) - Arrival\ Time(AT)$**
 - Minimize

- **Waiting time** – amount of time a process has been waiting in **ready queue**
 - Waiting time (WT) = TAT – Burst time (BT)
 - **Minimize**
- **Response time** – amount of time it takes from when a request is submitted until the first response is produced, not output (for time-sharing environment)
 - **Minimize**

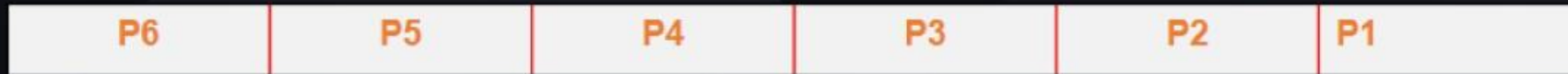
Lecture 04

Scheduling Algorithms

- First Come, First Served
- Shortest Job First
- Priority
- Round Robin

First-Come, First-Served (FCFS) Scheduling

- The simplest form of CPU scheduling Algorithms. The process that request the CPU first is allocated the CPU first
- The FCFS can be implemented using FIFO queue.



- The average waiting time is quite long.
- FCFS is always works a non-Preemptive scheduling algorithm (i.e., once the CPU is allocated a process; the process keeps use the processor until it either terminated or requested I/O request)

Lecture 04

First-Come, First-Served (FCFS) Scheduling

Process	Burst Time	CT	TAT	WT
P_1	24	24	24	0
P_2	3	27	27	24
P_3	3	30	30	27

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



- Average WT: $(0 + 24 + 27)/3 = 17$
- Average TAT = $(24+27+30)/3 = 27$

First-Come, First-Served (FCFS) Scheduling

Suppose that the processes arrive in the order

$$P_2, P_3, P_1$$

$$3, 3, 24$$

- 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

First-Come, First-Served (FCFS) Scheduling

- FCFS is fair in the formal sense or human sense of fairness but it is unfair in the sense that long jobs take priority over short jobs and unimportant jobs make important jobs wait
- The code for FCFS scheduling is simple to write and understand.
- One of the major drawbacks of this scheme is that the waiting time and the average turnaround time is often quite long
- FCFS Scheme is not useful in scheduling interactive users because it cannot guarantee good response time.

First-Come, First-Served (FCFS) Scheduling

- Example:

PID	Arrival time (AT)	Burst Time (BT)	CT	TAT	WT
P1	0	4	4	4	0
P2	1	3	7	6	3
P3	2	1	8	6	5
P4	3	2	10	7	5
P5	4	5	15	11	6



- Average WT = $(0+3+5+5+6)/5 = 3.8$
- Average TAT = $(4+6+6+7+11)/5 = 6.8$

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
- SJF is optimal – gives minimum average waiting time for a given set of processes
 - The difficulty is knowing the length of the next CPU request
 - Could ask the user

Shortest-Job-First (SJF) Scheduling

- 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 known as the Shortest-Remaining-Time-First (SRTF)

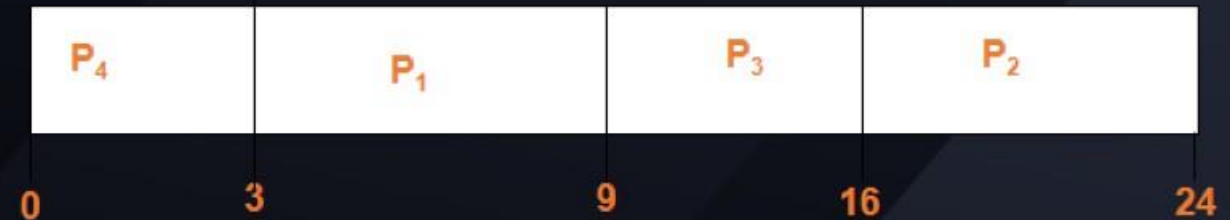
Lecture 04

Shortest-Job-First (SJF) Scheduling (non preemptive)

Arrival time is assumed 0 for all processes.

- SJF scheduling chart

Process	Burst Time	CT	TAT	WT
P_1	6	9	9	3
P_2	8	24	24	16
P_3	7	16	16	9
P_4	3	3	3	0



- Average WT = $(3 + 16 + 9 + 0) / 4 = 7$
- Average TAT = $(9+24+16+3)/4 = 13$

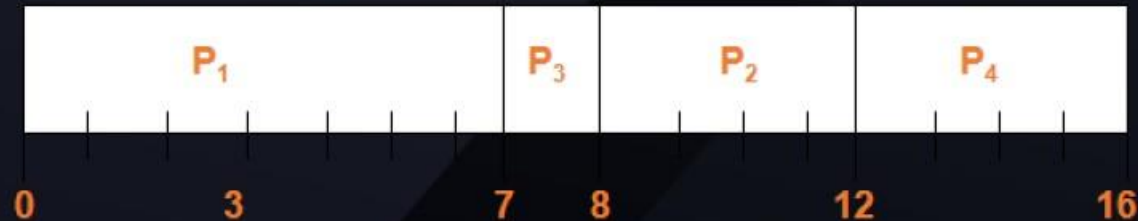
Lecture 04

Example of Non-Preemptive SJF

Process	Arrival Time	Burst Time	CT	TAT	WT
P_1	0.0	7	7	7	0
P_2	2.0	4	12	10	6
P_3	4.0	1	8	4	3
P_4	5.0	4	16	11	7

- In this example, every process
- Arrives at different time

- The Gantt chart for non-preemptive SJF is:



$$\text{Average WT} = (0 + 6 + 3 + 7)/4 = 4$$

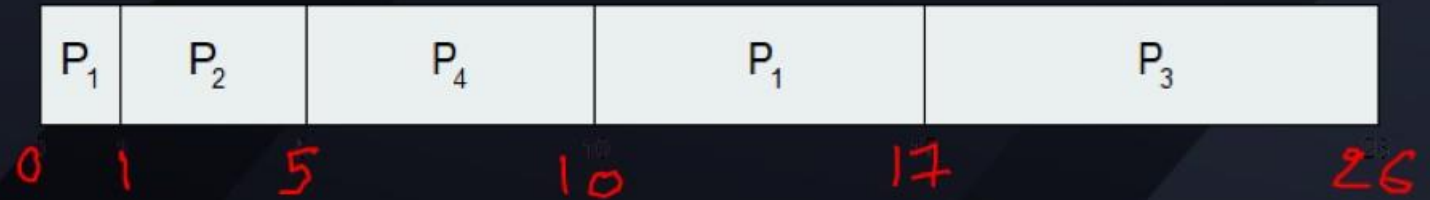
$$\text{Average TAT} = (7 + 10 + 4 + 11)/4 = 32/4 = 8$$

Lecture 04

Shortest-Job-First (SJF) Scheduling (preemptive)

This algorithm is also called:
Shortest-remaining-time-first

Process	Arrival Time	Burst Time	CT	TAT	WT
P_1	0	8	17	17	9
P_2	1	4	5	4	0
P_3	2	9	26	24	15
P_4	3	5	10	7	2



Average WT = $[(9+0+15+2)]/4 = 26/4 = 6.5$ msec

Average TAT = $[(9+4+24+7)]/4 = 44/4 = 11$ msec

Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority
- (smallest integer $\equiv$ highest priority)
 - **Preemptive**
 - **Non-preemptive**
- SJF is a priority scheduling where priority is the predicted next CPU burst time
- Problem $\equiv$ **Starvation** – low priority processes may never execute
- Solution $\equiv$ **Aging** – as time progresses increase the priority of the process

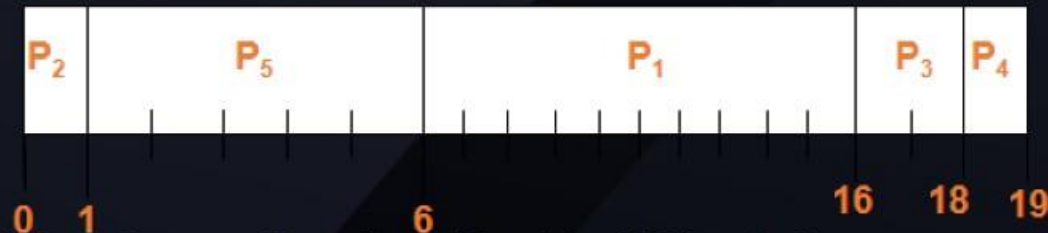
Lecture 04

Example of Priority Scheduling (non-preemptive)

All Arrival time = 0

Process	Burst Time	Priority	CT	TAT	WT
P_1	10	3	16	16	6
P_2	1	1	1	1	0
P_3	2	4	18	18	16
P_4	1	5	19	19	18
P_5	5	2	6	6	1

- The Gantt chart is (all processes arrive at time 0):



- Average waiting time = $(6 + 0 + 16 + 18 + 1)/5 = 8.2$
- Average TAT = $(16 + 1 + 18 + 19 + 6)/5 = 12$

Lecture 04

Example of Priority Scheduling preemptive

Here processes arrives
At different times

Process	Arrival Time	Burst Time	Priority	CT	TAT	WT
P_1	0	10	3	16	16	6
P_2	1	1	1	2	1	0
P_3	2	2	4	18	16	14
P_4	3	1	5	19	16	15
P_5	4	5	2	9	5	0

- The Gantt chart is:



- Average WT = $(6 + 0 + 14 + 15 + 0)/5 = 7$
- Average TAT = $(16 + 1 + 16 + 16 + 5)/5 = 10.8$

- The round robin RR scheduling algorithm is designed especially for timesharing systems.
- It is similar to the FCFS scheduling, but preempting is added to switch between processes.
- 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.

- The CPU scheduler goes around the ready queue , allocating the CPU to each process for a time interval of up to 1 time quantum.
- 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
 - *If q is large compared to the processes burst time $\Rightarrow$ the round robin algorithm becomes FCFS*
 - *IF q is small $\Rightarrow q$ must be large with respect to context switch, otherwise overhead is too high*

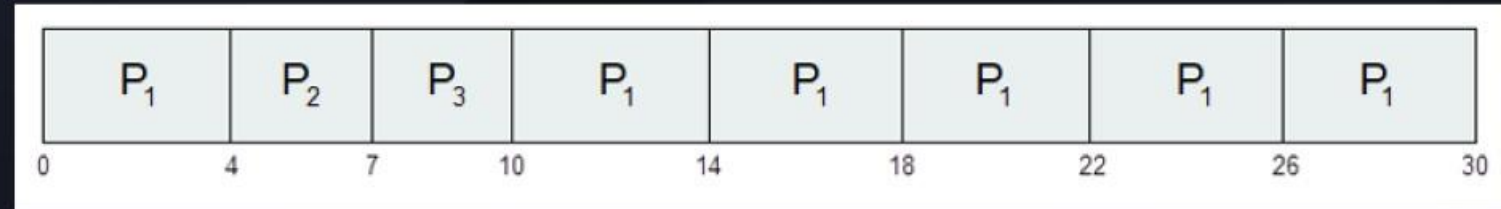
Lecture 04

Example of RR with Time Quantum = 4

Assume the arrival time = 0 for All processes.

Process	Burst Time	CT	TAT	WT
P_1	24	30	30	6
P_2	3	7	7	4
P_3	3	10	10	7

- The Gantt chart is:



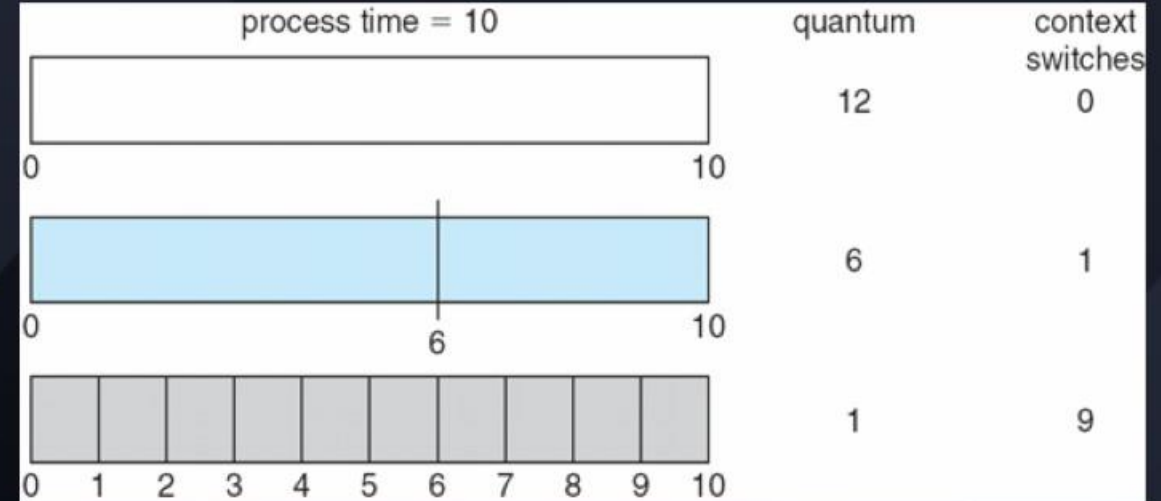
- Average WT = $(6+4+7)/3 = 5.7$
- Average TAT = $(30+7+10)/3 = 15.7$
- Typically, higher average turnaround than SJF, but better *response*

Lecture 04

Time Quantum and Context Switch Time

Assume, for example, that we have only one process of 10 time units.

1. If the quantum is 12 time units, the process finishes in less than 1 time quantum, with no overhead.
2. If the quantum is 6 time units, however, the process requires 2 quanta, resulting in a context switch.
3. If the time quantum is 1 time unit, then nine context switches will occur, slowing the execution of the process accordingly



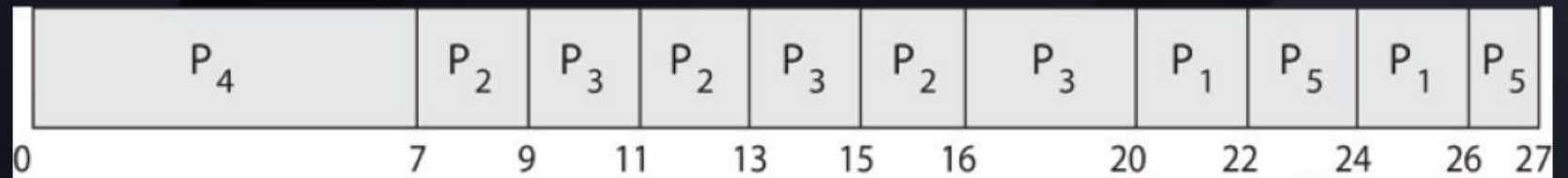
Lecture 04

Priority Scheduling w/ Round-Robin

Process	Burst Time	Priority	CT	TAT	WT
P_1	4	3	26	26	22
P_2	5	2	16	16	11
P_3	8	2	20	20	12
P_4	7	1	7	7	0
P_5	3	3	27	27	24

Run the process with the highest priority. Processes with the same priority run round-robin

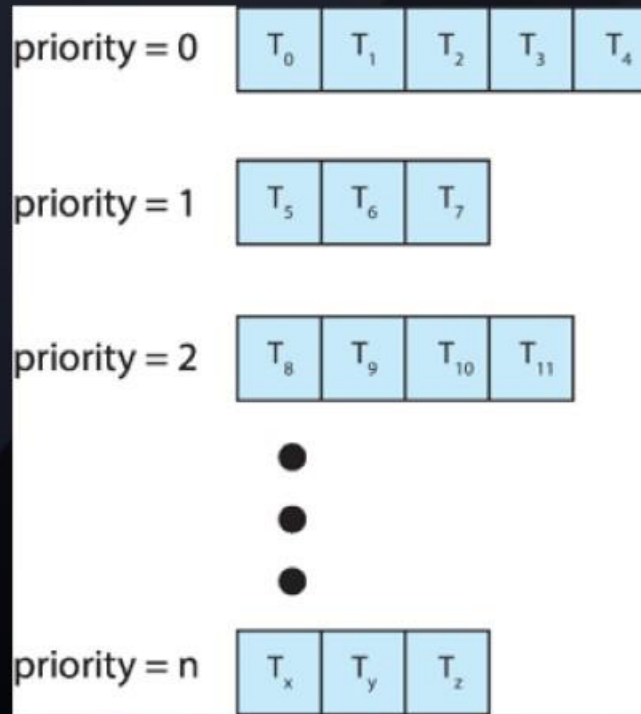
Gantt Chart wit 2 ms time quantum



Lecture 04

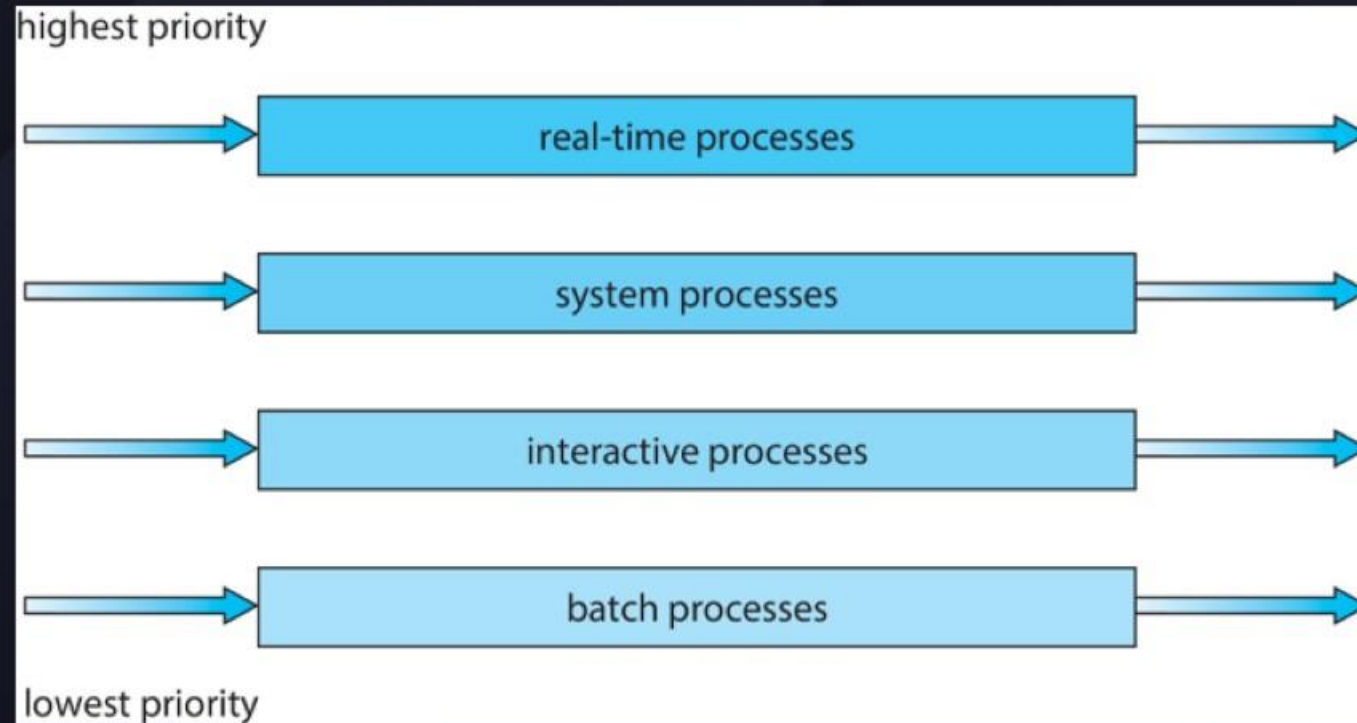
Multilevel Queue

- With priority scheduling, have separate queues for each priority.
- Schedule the process in the highest-priority queue!



Lecture 04

Multilevel Queue

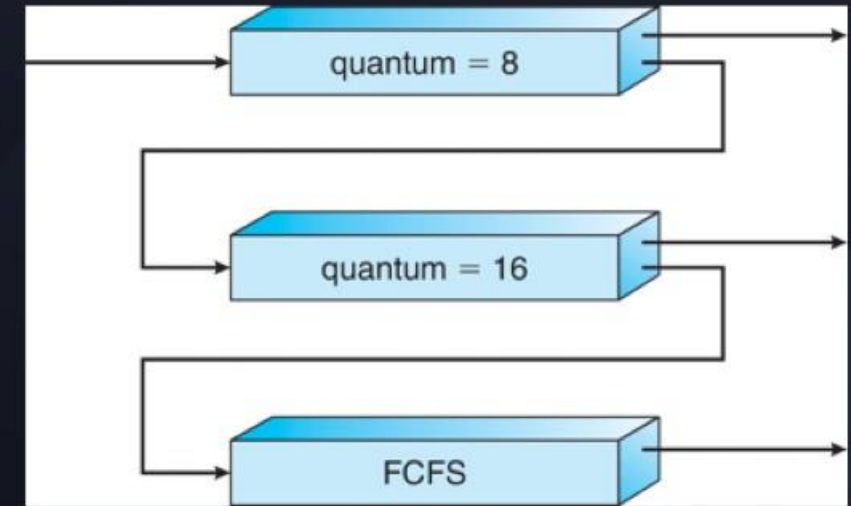


Multilevel Feedback Queue

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

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
 - 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





Thank You