SNS COLLEGE OF TECHNOLOGY

Coimbatore-35

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

23CST202- OPERATING SYSTEMS

II YEAR AIML B IV SEM

UNIT 2 – PROCESS SCHEDULING AND SYNCHRONIZATION

TOPIC – SCHEDULING ALGORITHM

Preemptive vs. Non-preemptive scheduling

- Non-preemptive scheduling:
  - Each process completes its full CPU burst cycle before the next process is scheduled.
  - No time slicing or CPU stealing occurs.
  - Once a process has control of the CPU, it keeps control until it gives it up (e.g. to wait for I/O or to terminate).
  - Works OK for batch processing systems, but not suitable for time sharing systems.
- Preemptive scheduling:
  - A process may be interrupted during a CPU burst and another process scheduled. (E.g. if the time slice of the first process expires).
  - More expensive implementation due to process switching.
  - Used in all time sharing and real time systems.

Costs of Preemptive Scheduling

- Preemptive scheduling leads to some problems that the OS must deal with:
- Problem 1: inconsistent data:
  - Suppose process 1 is updating data when preempted by process 2.
  - Process 2 may then try to read the data, which is in an inconsistent state.
  - The OS needs mechanisms to coordinate shared data.
- Problem 2: Kernel preemption:
  - Suppose the kernel is preempted while updating data (e.g. I/O queues) used by other kernel functions. This could lead to chaos.
  - UNIX solution: Wait for the system call to complete or have an I/O block take place if in kernel mode.
  - Problem with UNIX solution: Not good for real time computing.

Dispatcher

- Process scheduling determines the order in which processes execute.
- 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
- 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

- The scheduling criteria are optimization problems. We would like to maximize or minimize each.
- Question: Maximize or Minimize?
  - CPU utilization:
  - throughput:
  - turnaround time:
  - waiting time:
  - response time:
- Can all criteria be optimized simultaneously?

- Usually try to optimize average times (although sometimes optimize minimum or maximum)

First-Come, First-Served (FCFS) Scheduling

Process that requests the CPU first is allocated the CPU first.
Easily managed with a FIFO queue.
Often the average waiting time is long.

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:

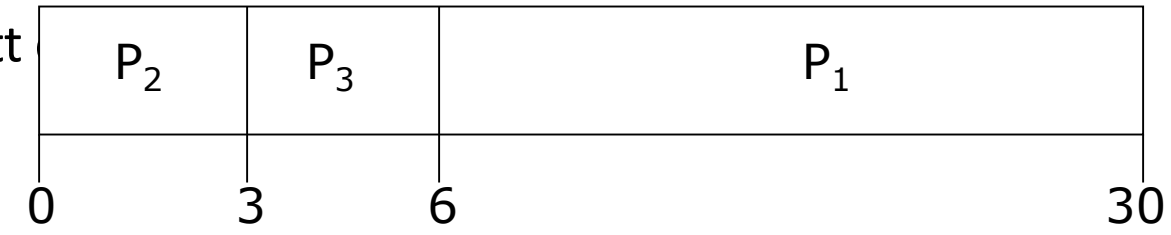
- Waiting time for P_1 = ; P_2 = ; P_3 =
- Average waiting time:

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order

P_2 , P_3 , P_1 .

- The Gantt



- Waiting time for P_1 = ; P_2 = ; P_3 =
- Average waiting time:
- Much better than previous case.
- *Convoy effect:* short processes line up behind long process.
- FCFS is not good for time-sharing systems. (Non-preemptive).

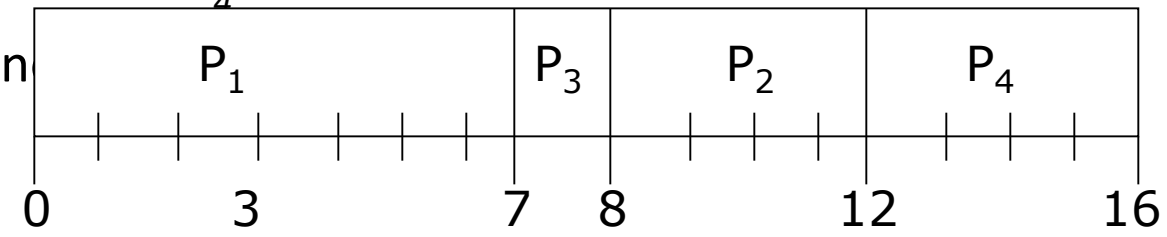
Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- Two schemes:
  - nonpreemptive – 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 (n

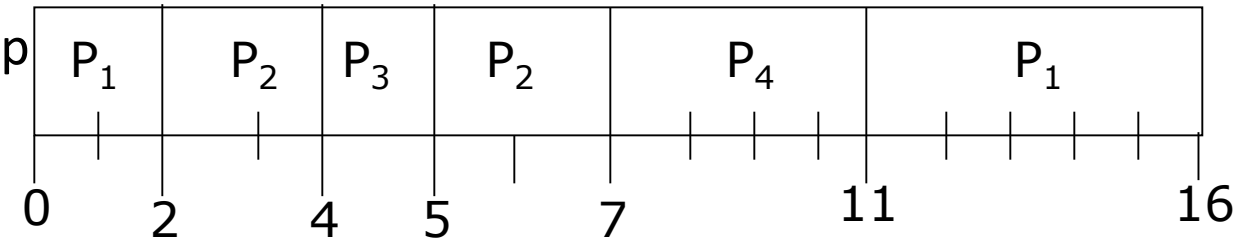


- Average waiting time =

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 (p



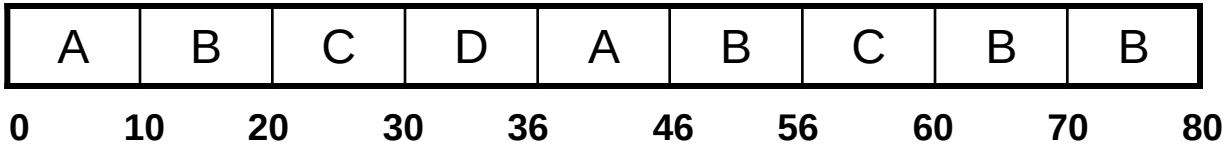
- Average waiting time =

Example: RR with Quantum=10ms

- Consider the following set of process that arrive at time 0 in the order A,B,C,D with the following given CPU burst time. Find the average waiting time with RR of quantum : 10 ms

Process	CPU burst time (ms)
A	20
B	40
C	14
D	6

Gantt Chart:



W (A) = 36-10 =26 ms W (B) = (10-0) + (46-20) + (60-56)= 40 ms

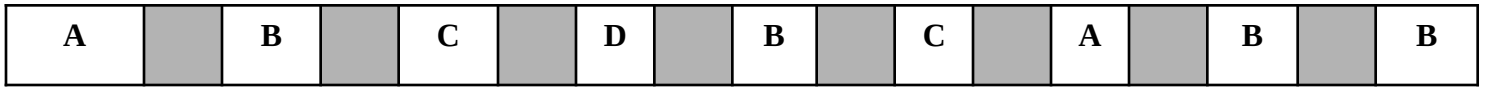
W (C) = (20-0) + (56-30) = 46 ms W (D) = 30-0 = 30 ms

Average waiting time = (26+40+46+30) /4 = 142/4 = 35.5 ms

Example: RR with Quantum = 10ms

- Consider the following set of process that arrive at time 0 in the order A,B,C,D with the following given CPU burst time. Find the average waiting time with RR of quantum = 10 ms and context switch time = 2ms

Process	CPU burst time	I/O	CPU burst time
A	10	40	10
B	40	-	-
C	14	-	-
D	6	-	-



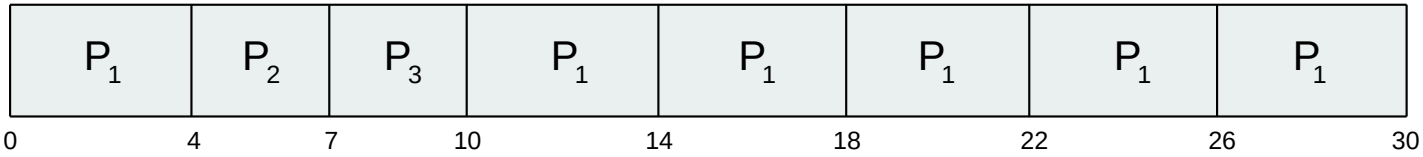
Wait (A) = 62-50 = 12 ms **Wait (B) = 12+22+20+2 = 56 ms**
Wait (C) = 24 + 22 = 46 ms **Wait (D) = 36 ms**

Average waiting time = 150/4 = 37.5 ms

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:



- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 microsecond

Time Quantum and Context Switch Time

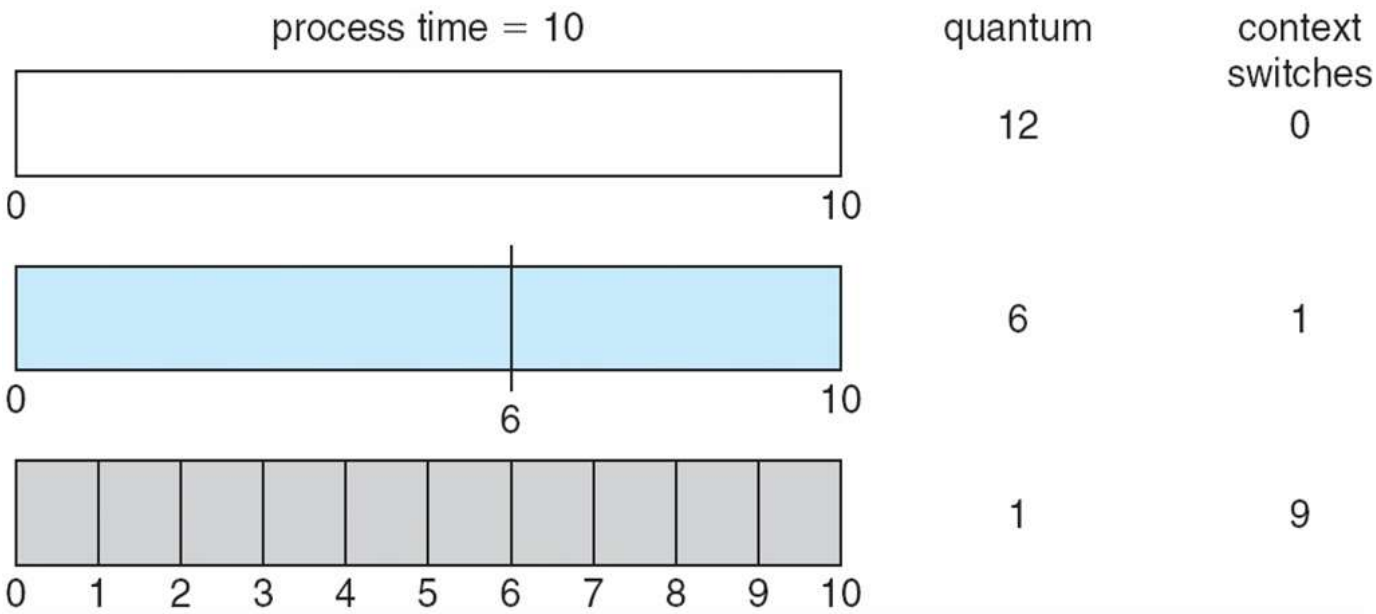


Figure showing how smaller time quantum increases context switches.

Priority Scheduling

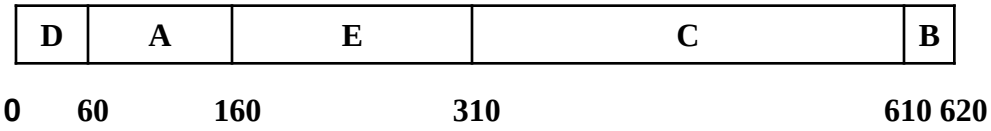
- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority ((**smallest or largest integer value may be defined as the highest priority**)
  - Preemptive
  - Nonpreemptive
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ≡ **Starvation** – low priority processes may never execute
- Solution ≡ **Aging** – as time progresses increase the priority of the process

Example - Priority Scheduling

Process	Arrival time	CPU burst time	Priority
A	0	100	3
B	0	10	1
C	0	300	3
D	0	60	5
E	80	150	4

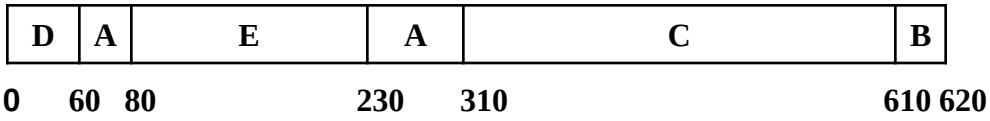
5 highest
1 lowest

Non-preemptive priority Algorithm:



W (A) = 60ms, W (B) = 610ms, W (C) = 310ms, W (D) = 0 ms, W (E) = 80ms

Preemptive priority Algorithm (if a job come with high priority there is a switching):

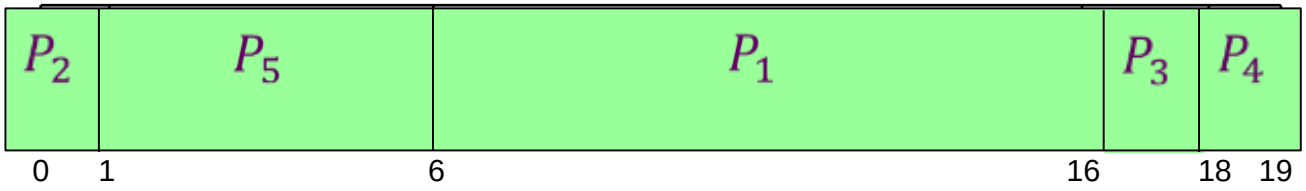


W(A) = 210 ms, W(B) = 610 ms, W(C) = 310 ms, W(D)=0ms, Wait (E) = 0 ms

Example - Priority Scheduling

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

- Priority scheduling Gantt Chart

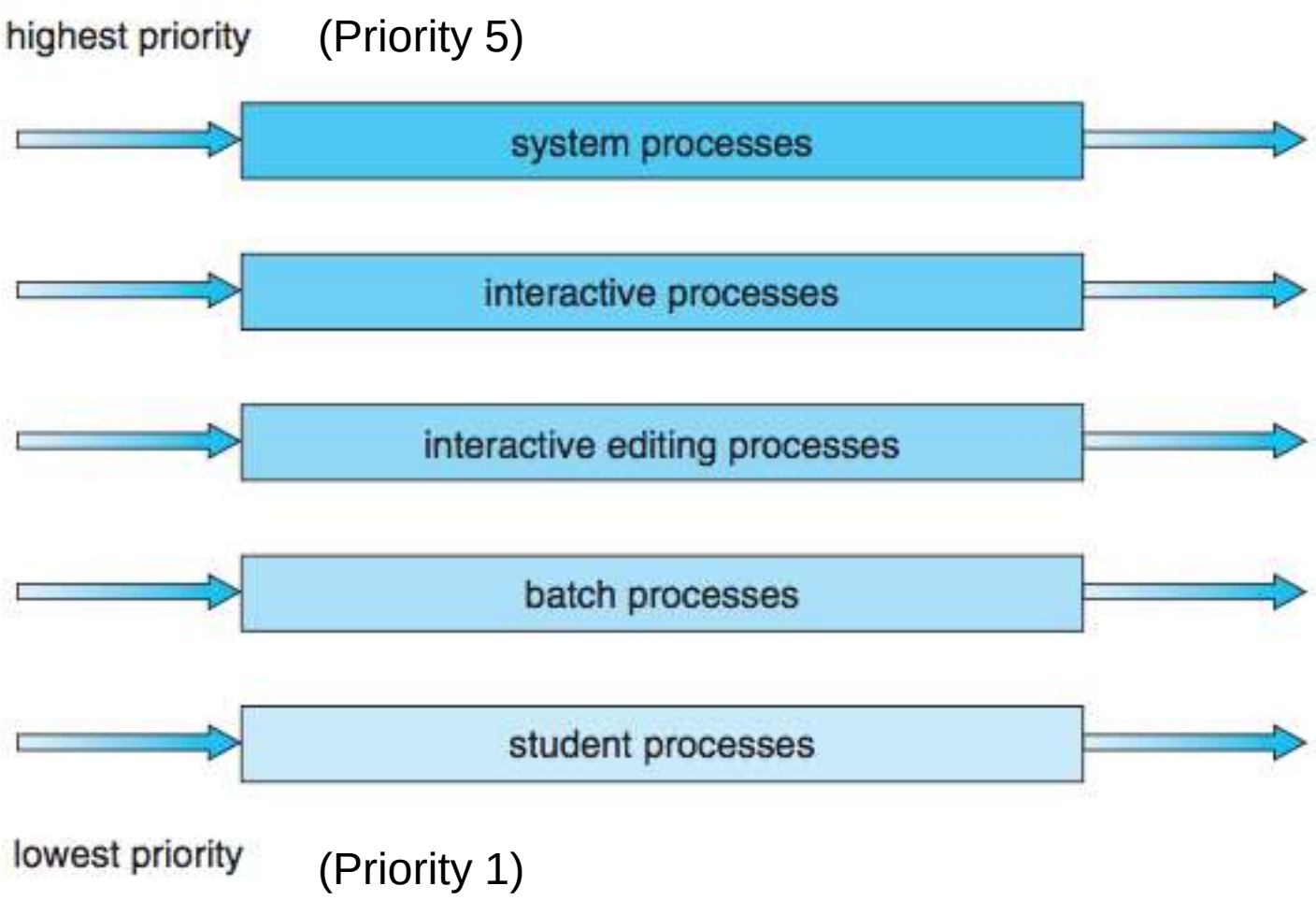


- Average waiting time = 8.2 msec

Multilevel Queue Scheduling

- Another class of scheduling algorithms has been created for situations in which processes are easily classified into different groups(classes).
- Among classes a priority scheduling algorithm is used, inside a class RR is used.
- A ready queue is used for each class.
- The following shows a system with five priority classes where RR is used in each class. As long as there are runnable processes in priority class 5, just run each one for one quantum (i.e. round robin fashion), and never bother with lower priority classes. If priority class 5 in empty , then run the class 4 processes in round robin fashion , and so on. If priorities are not adjusted from time to time, lower priority classes may all starve.
- While a class 3 process is running if a highest queue process arrives, class 3 process would be preempted.
- Time quantum value can be different for classes.
- Each queue may have its own scheduling algorithm.

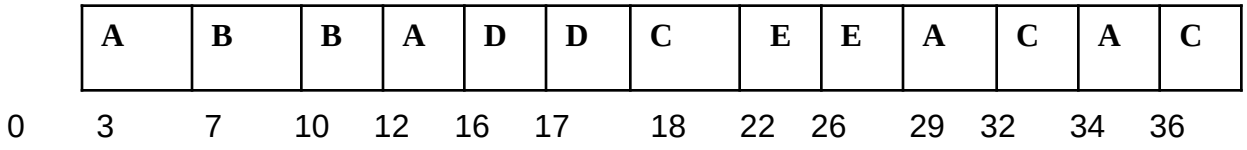
Multilevel Queue Scheduling



Multilevel Queue Scheduling

Process	Arrival time	CPU burst time (ms)	Priority
A	0	10	2
B	3	7	1
C	4	6	2
D	12	5	1
E	18	8	1

Assume that Quantum=4 ms for Queue 1 and Quantum =3 ms for Queue 2. Assume that Queue 1 has higher priority when compared to Queue 2.

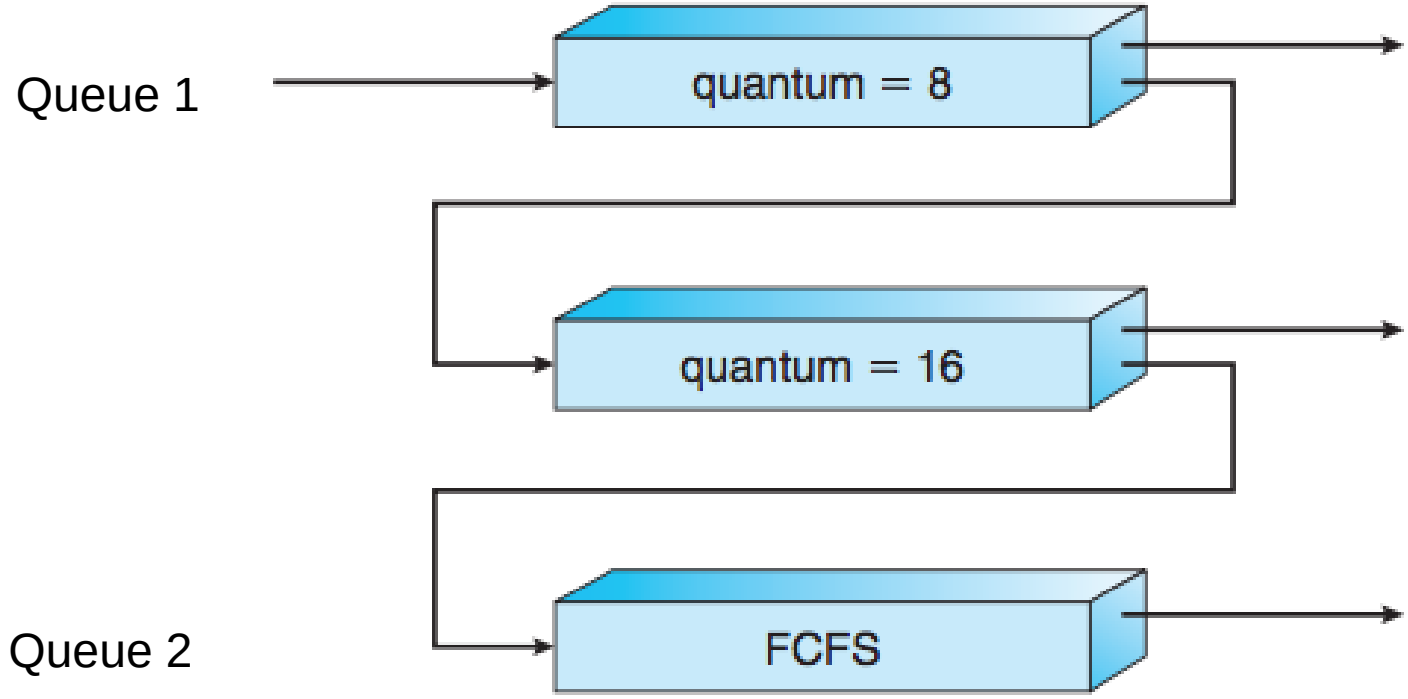


W (A) = 24ms **W (B) = 0ms** **W (C) = 26ms** **W (D) = 0ms**
W (E) = 0ms **W (F) = 10ms**

Multilevel Feedback Queue Scheduling

- Normally, when the multilevel queue scheduling algorithm is used, processes are permanently assigned to a queue when they enter the system.
- The **multilevel feedback queue** scheduling algorithm allows a process to move between queues.
- The scheduler first executes all processes in Queue 1. Only when Queue 1 is empty will it execute processes in Queue 2. Similarly, processes in Queue 3 will be executed only if Queues 1 and 2 are empty. A process that arrives for Queue 1 will preempt a process in Queue 2.
- A process entering the ready queue is put in Queue 1. If it does not finish within one quantum, it is moved to the tail of Queue 2. If it does not complete there in one quantum, it is then preempted and put into Queue 3.

Multilevel Feedback Queue Scheduling



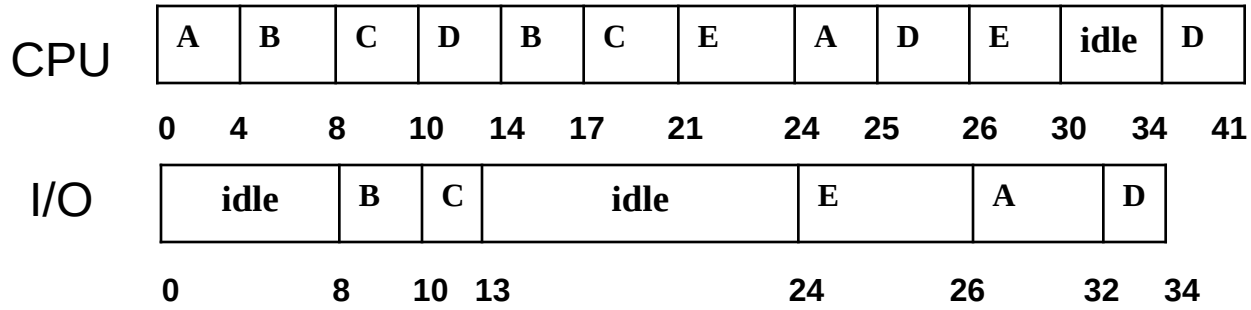
The figure above shows an example for multilevel feedback queue where Queue 1 uses RR with Quantum = 8ms, Queue 2 uses RR with Quantum = 16 and Queue 3 uses FCFS.

Multilevel Feedback Queue Scheduling

Process	Arrival time	CPU burst time	I/O	CPU burst time
A	0	5	6	0
B	3	4	2	3
C	4	2	3	4
D	7	5	2	7
E	14	3	2	4

Assume Queue 1 with Quantum =4 ms and Queue 2 with Quantum = 7ms, both use Round Robin.

Queue_1 : ABCDBCEED
Queue_2 : AD



Algorithm Evaluation

- How to select CPU-scheduling algorithm for an OS?
- Determine criteria, then evaluate algorithms
  - Takes a particular predetermined workload and defines the performance of each algorithm for that workload
- Consider 5 processes arriving at time 0, in the order given, with the length of the CPU burst given in ms.

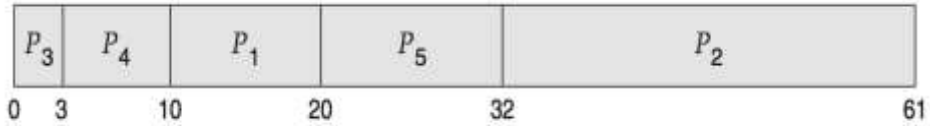
Process	Burst Time
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

- Consider the FCFS, SJF and RR(quantum = 10 ms) scheduling algorithms for this set of processes. Which algorithm would give the minimum average waiting time?

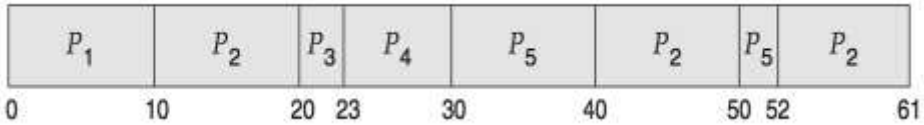
Algorithm Evaluation

- For each algorithm, calculate minimum average waiting time
- Simple and fast, but requires exact numbers for input, applies only to those inputs
  - FCFS is (0+10+39+42+49)/5=28ms:

  - Non-preemptive SJF is (10+32+0+3+20)/5=13ms:



  - RR is (0+32+20+23+40)/5=23ms:



Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available.
- *Homogeneous processors* within a multiprocessor: We concentrate on systems where the processors are identical (or homogeneous) in terms of their functionality; any available processor can then be used to run any processes in the queue.
- *Load sharing* : If several identical processors are available, then load sharing can occur. It would be possible to provide a separate queue for each processor. In this case, however, one processor could be idle, with an empty queue, while another processor was very busy. To prevent this situation, we use a common ready queue. All processes go into one queue and are scheduled onto any available processor.
- *Asymmetric multiprocessing* – only one processor accesses the system data structures, alleviating the need for data sharing. having all scheduling decisions, I/O processing, and other system activities handled by one single processor-the master server. The other processors only execute user code.