Chapter 4

CPU Scheduling (Algorithms)

Chapter 4: CPU Scheduling

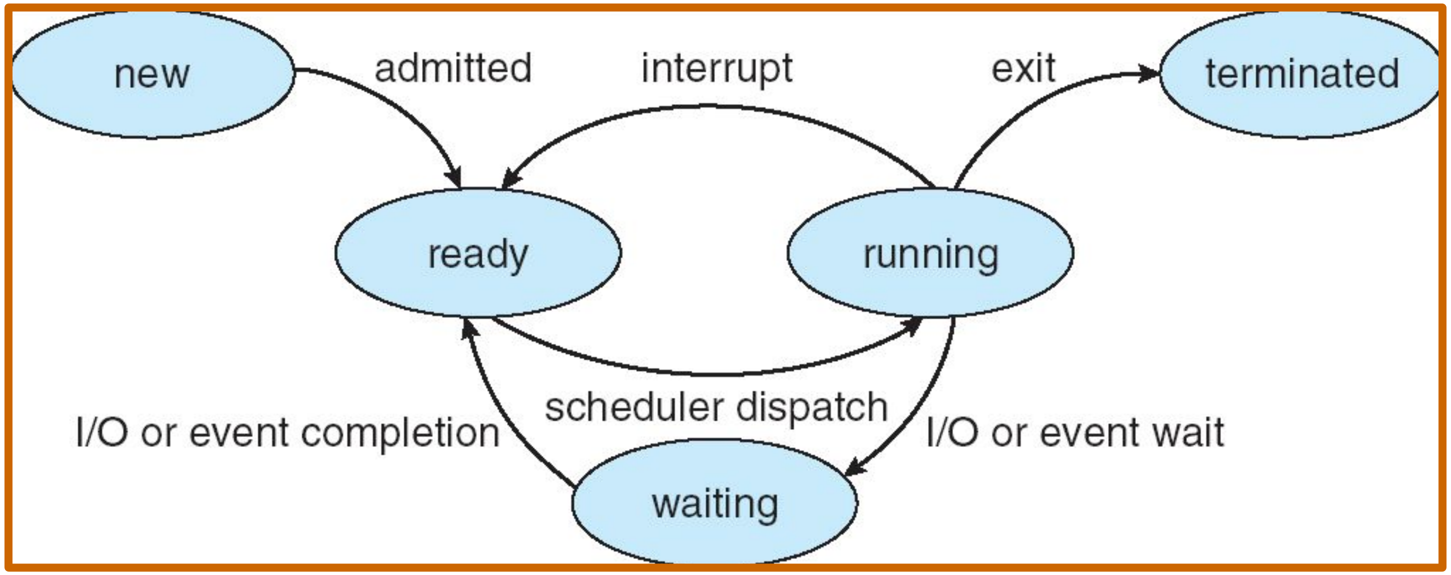
4.1 Basic Concepts

Basic Concepts

- Maximum CPU utilization is obtained with multiprogramming
  - Several processes are kept in memory at one time
  - Every time a running process has to wait, another process can take over use of the CPU
- Scheduling of the CPU is fundamental to operating system design
- Process execution consists of a *cycle* of a **CPU time burst** and an **I/O time burst** (i.e. wait) as shown on the next slide
  - Processes alternate between these two states (i.e., CPU burst and I/O burst)
  - Eventually, the final CPU burst ends with a system request to terminate execution

CPU Scheduler

- The CPU scheduler selects from among the processes in memory that are ready to execute and allocates the CPU to one of them
- CPU scheduling is affected by the following set of circumstances:
  1. (N) A process switches from **running** to **waiting** state
  2. (P) A process switches from **running** to **ready** state
  3. (P) A process switches from **waiting** to **ready** state
  4. (N) A processes switches from **running** to **terminated** state
- Circumstances 1 and 4 are **non-preemptive**; they offer no schedule choice
- Circumstances 2 and 3 are **pre-emptive**; they can be scheduled



4.2 Scheduling Criteria

Scheduling Criteria

- Different CPU scheduling algorithms have different properties
- The choice of a particular algorithm may favor one class of processes over another
- In choosing which algorithm to use, the properties of the various algorithms should be considered
- Criteria for comparing CPU scheduling algorithms may include the following
  - **CPU utilization** – percent of time that the CPU is busy executing a process
  - **Throughput** – number of processes that are completed per time unit
  - **Response time** – amount of time it takes from when a request was submitted until the first response occurs (but not the time it takes to output the entire response)
  - **Waiting time** – the amount of time before a process starts after first entering the ready queue (or the sum of the amount of time a process has spent waiting in the ready queue)
  - **Turnaround time** – amount of time to execute a particular process from the time of submission through the time of completion

Optimization Criteria

- It is desirable to
  - Maximize CPU utilization
  - Maximize throughput
  - Minimize turnaround time
  - Minimize start time
  - Minimize waiting time
  - Minimize response time
- In most cases, we strive to optimize the average measure of each metric
- In other cases, it is more important to optimize the minimum or maximum values rather than the average

4.3 Single Processor Scheduling Algorithms

Single Processor Scheduling Algorithms

- First Come, First Served (FCFS)
- Shortest Job First (SJF)
- Priority
- Round Robin (RR)

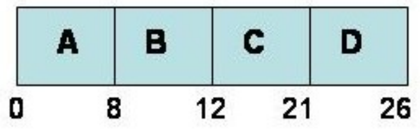
First Come, First Served (FCFS) Scheduling

FCFS Scheduling Example

- suppose the scheduler is given 4 tasks, A, B, C and D. Each task requires a certain number of time units to complete.

Task	Time units
A	8
B	4
C	9
D	5

- The FCFS scheduler's Gantt chart for these tasks would be:



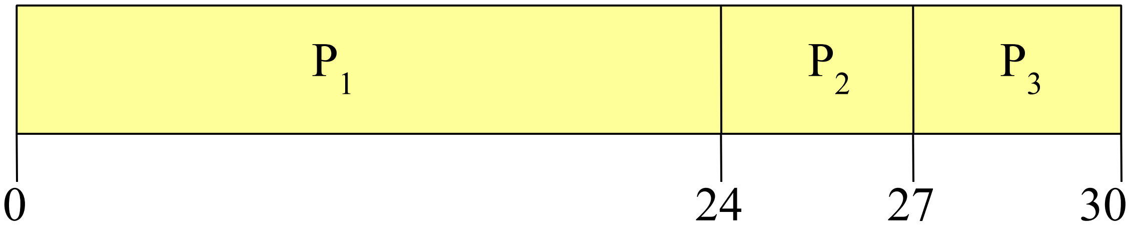
- The OS incurs some overhead each time it switches between processes due to **context switching**. We will call this overhead "**cs**".
- *CPU Utilization* -26/(26+3**cs**)
- *Turn around time* - (8+12+21+26+6**cs**)/4 = 16.5 ignoring **cs**
- *Waiting-(0+8+12+21+6cs)/4 = 10.25 ignoring cs*
- *Throughput-4/(26 + 3cs)*
- *Response-(0+8+cs+12+2cs+21+3cs)/4 = 10.25 ignoring **cs***

First-Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

- With FCFS, the process that requests the CPU first is allocated the CPU first
- Case #1: Suppose that the processes arrive in the order: P_1 , P_2 , P_3

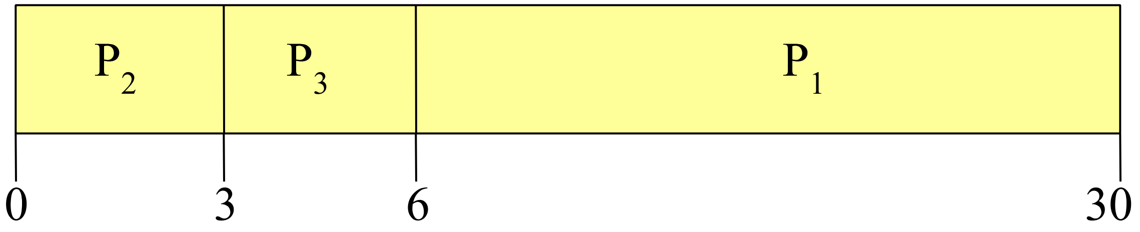
  The Gantt Chart for the schedule is:



- Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Average waiting time: $(0 + 24 + 27)/3 = 17$
- Average turn-around time: $(24 + 27 + 30)/3 = 27$

FCFS Scheduling (Cont.)

- Case #2: 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 Case #1)
- Average turn-around time: $(3 + 6 + 30)/3 = 13$
- Case #1 is an example of the **convoy effect;** all the other processes wait for one long-running process to finish using the CPU
  - This problem results in lower CPU and device utilization; Case #2 shows that higher utilization might be possible if the short processes were allowed to run first
- The FCFS scheduling algorithm is **non-preemptive**
  - Once the CPU has been allocated to a process, that process **keeps the CPU** until it releases it either by terminating or by requesting I/O
  - It is a **troublesome algorithm for time-sharing systems**

Shortest Job First (SJF) Scheduling

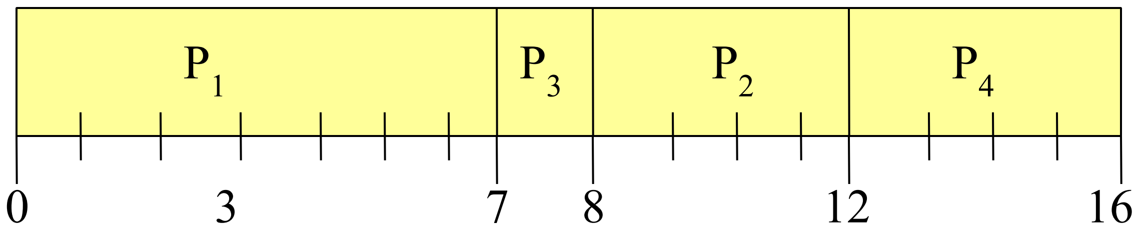
Shortest-Job-First (SJF) Scheduling

- The SJF algorithm associates with each process the length of its next CPU burst
- When the CPU becomes available, it is assigned to the process that has the smallest next CPU burst (in the case of matching bursts, FCFS is used)
- Two schemes:
  - **Nonpreemptive** – once the CPU is given to the process, it cannot be preempted until it completes its CPU burst
  - **Preemptive** – if a new process arrives with a CPU burst length less than the remaining time of the current executing process, preempt. This scheme is know as the **Shortest-Remaining-Time-First (SRTF**)

Example #1: 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, varied arrival times)



- Average waiting time

  = ((0 – 0) + (8 – 2) + (7 – 4) + (12 – 5))/4

  = (0 + 6 + 3 + 7)/4 = 4

- Average turn-around time:

  = ((7 – 0) + (12 – 2) + (8 - 4) + (16 – 5))/4

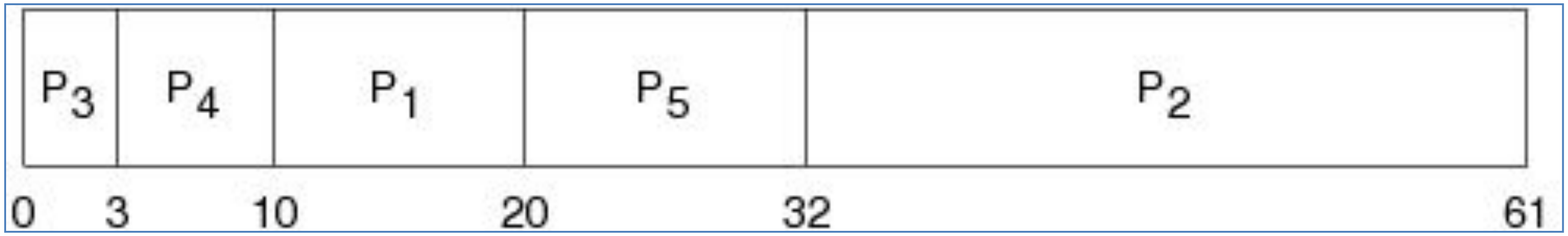
  = (7 + 10 + 4 + 11)/4 = 8

Waiting time : sum of time that a process has spent waiting in the ready queue

Ex:2

Process	Burst Time	Arrival	Start	Wait	Finish	TAT
P1	10	0	10	10	20	20
P2	29	0	32	32	61	61
P3	3	0	0	0	3	3
P4	7	0	3	3	10	10
P5	12	0	20	20	32	32

Gantt chart:

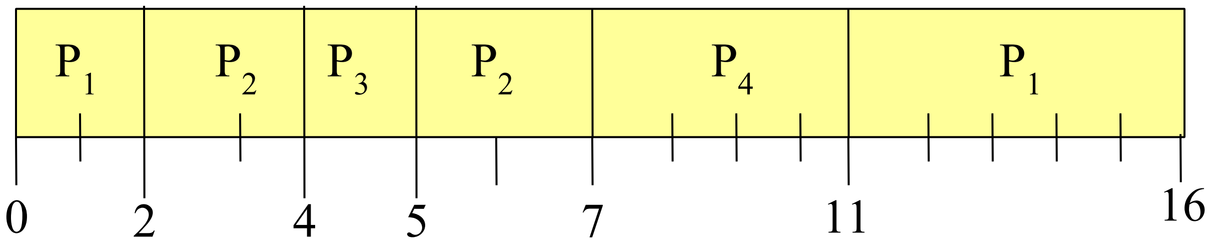


Average waiting time: (10+32+0+3+20)/5 = 13
Average turnaround time: (10+39+42+49+61)/5 = 25.2

Example #3: Preemptive SJF (Shortest-remaining-time-first)

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, varied arrival times)



- Average waiting time

  = ([(0 – 0) + (11 - 2)] + [(2 – 2) + (5 – 4)] + (4 - 4) + (7 – 5))/4

  = (9 + 1 + 0 + 2)/4 = 3

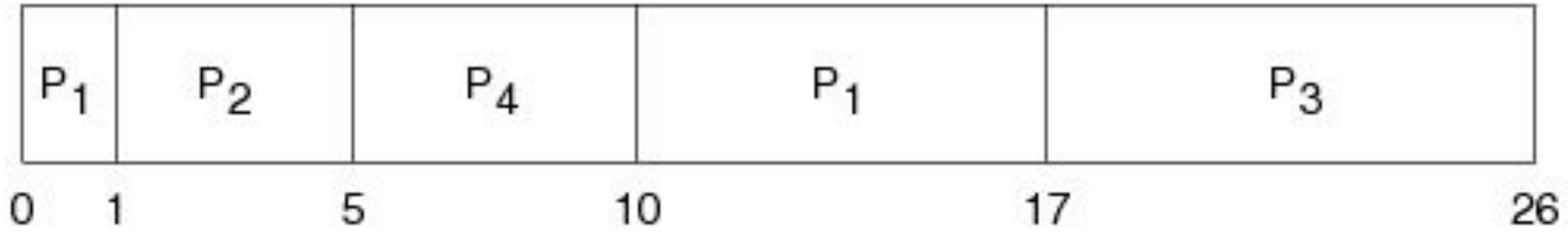
  Average turn-around time = (16-0) + (7-2) + (5-4) +(11-5)/4 = 7

Waiting time : sum of time that a process has spent waiting in the ready queue

Ex4

Process	Burst Time	Arrival	Start	Wait	Finish	TA
P1	8	0	0	9	17	17
P2	4	1	1	0	5	4
P3	9	2	17	15	26	24
P4	5	3	5	2	10	7

Gantt chart:



Average waiting time: [(0-0)+(10-1)]+(1-1)+(17-2)+(5-3))/4 = 6.5

Average turnaround time: (17-0)+(5-1)+(26-2)+(10-3)/4 = 13

Exponential Averaging

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

- τ_{n+1} : predicted length of next CPU burst
- t_n : actual length of last CPU burst
- τ_n : previous prediction

- α **= 0** implies make no use of recent history

$$(\tau_{n+1} = \tau_n)$$

- α **= 1** implies $\tau_{n+1} = t_n$ (past prediction not used).
- α **= 1/2** implies weighted (older bursts get less and less weight).

Priority Scheduling

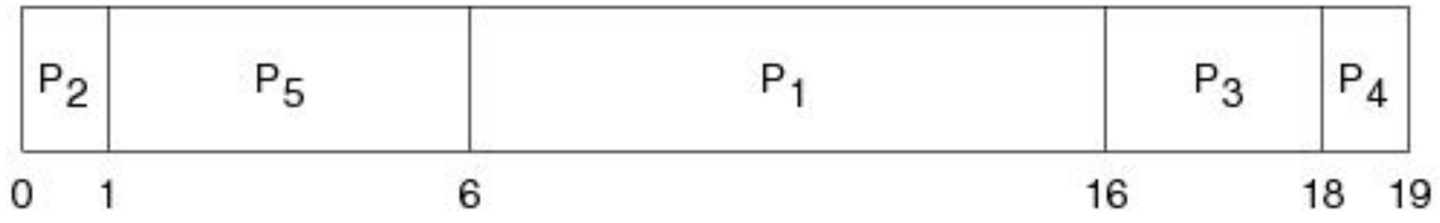
Priority Scheduling

- The **SJF algorithm** is a special case of the general priority scheduling algorithm
- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer = highest priority)
- Priority scheduling can be either preemptive or non-preemptive
  - A **preemptive** approach will preempt the CPU if the priority of the newly-arrived process is higher than the priority of the currently running process
  - A **non-preemptive** approach will simply put the new process (with the highest priority) at the head of the ready queue
- SJF is a priority scheduling algorithm where priority is the predicted next CPU burst time
- The main problem with priority scheduling is **starvation**, that is, low priority processes may never execute
- A solution is **aging**; as time progresses, the priority of a process in the ready queue is increased

Priority Scheduling example

Process	Burst Time	Priority	Arrival	Start	Wait	Finish	TA
P1	10	3	0	6	6	16	16
P2	1	1	0	0	0	1	1
P3	2	4	0	16	16	18	18
P4	1	5	0	18	18	19	19
P5	5	2	0	1	1	6	6

Gantt chart:



Average waiting time: (6+0+16+18+1)/5 = 8.2

Average turnaround time: (16+1+18+19+6)/5 = 12

Round Robin (RR) Scheduling

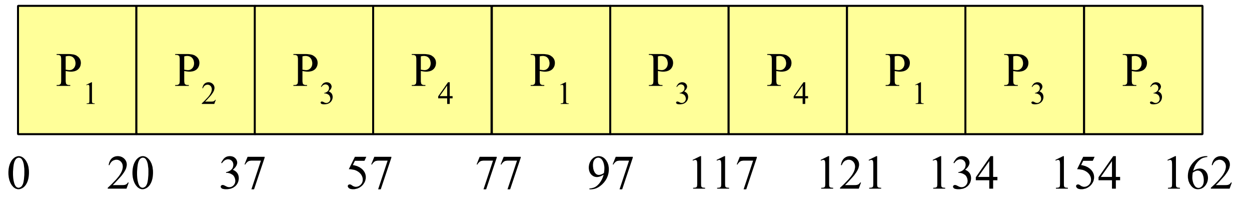
Round Robin (RR) Scheduling

- In the round robin algorithm, each process gets a small unit of CPU time (a *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 of the round robin algorithm
  - q large $\Rightarrow$ FCFS
  - q small $\Rightarrow$ q must be greater than the <u>context switch</u> time; otherwise, the overhead is too high
- One rule of thumb is that 80% of the CPU bursts should be shorter than the time quantum

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:

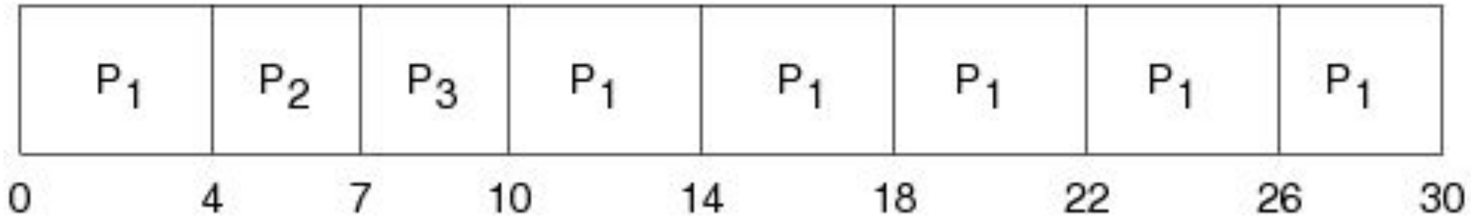


- Typically, higher average turnaround than SJF, but better *response time*
- Average waiting time
  = ([(0 – 0) + (77 - 20) + (121 – 97)] + (20 – 0) + [(37 – 0) + (97 - 57) + (134 – 117)] + [(57 – 0) + (117 – 77)]) / 4
  = (0 + 57 + 24) + 20 + (37 + 40 + 17) + (57 + 40)) / 4
  = (81 + 20 + 94 + 97)/4
  = 292 / 4 = 73
- Average turn-around time = 134 + 37 + 162 + 121) / 4 = 113.5

Example 2 (q = 4):

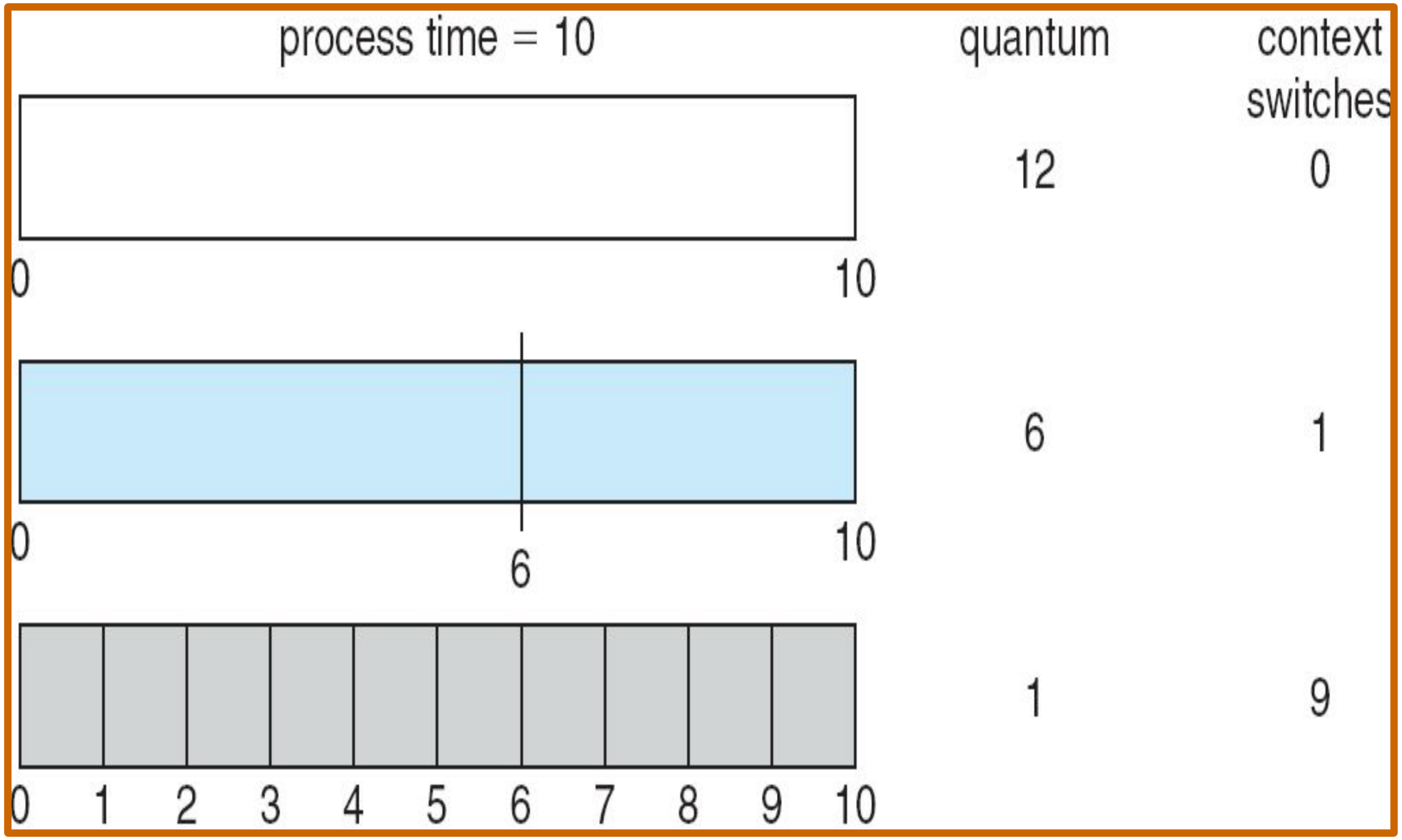
Process	Burst Time	Arrival	Start	Wait	Finish	TA
1	24	0	0	6	30	30
2	3	0	4	4	7	7
3	3	0	7	7	10	10

Gantt chart:

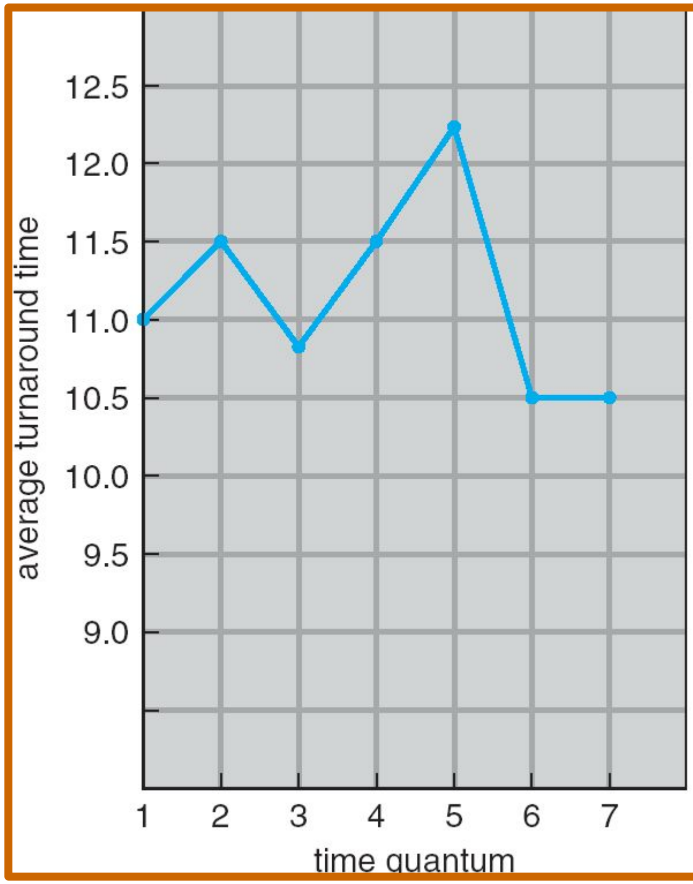


Average waiting time: (6+4+7)/3 = 5.67
Average turnaround time: (30+7+10)/3 = 15.67

Time Quantum and Context Switches



Turnaround Time Varies With The Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

As can be seen from this graph, the average turnaround time of a set of processes does not necessarily improve as the time quantum size increases. In general, the average turnaround time can be improved if most processes finish their next CPU burst in a single time quantum.

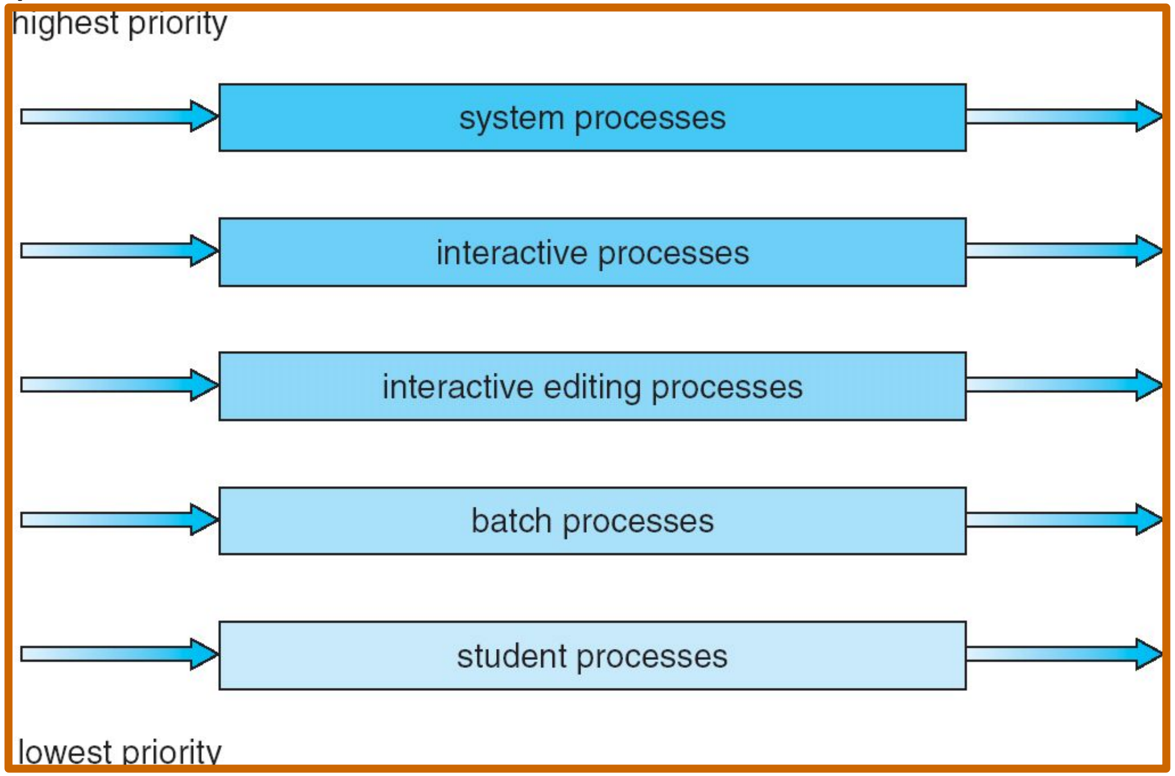
4.3b Multi-level Queue Scheduling

Multi-level Queue Scheduling

- Multi-level queue scheduling is used when processes can be classified into groups
- For example, **foreground** (interactive) processes and **background** (batch) processes
  - The two types of processes have different response-time requirements and so may have different scheduling needs
  - Also, foreground processes may have priority (externally defined) over background processes
- A multi-level queue scheduling algorithm partitions the ready queue into several separate queues
- The processes are permanently assigned to one queue, generally based on some property of the process such as memory size, process priority, or process type
- Each queue has its own scheduling algorithm
  - The foreground queue might be scheduled using an RR algorithm
  - The background queue might be scheduled using an FCFS algorithm
- In addition, there needs to be scheduling among the queues, which is commonly implemented as fixed-priority pre-emptive scheduling
  - The foreground queue may have absolute priority over the background queue

Multi-level Queue Scheduling

- One example of a multi-level queue are the five queues shown below
- Each queue has absolute priority over lower priority queues
- For example, no process in the batch queue can run unless the queues above it are empty
- However, this can result in **starvation** for the processes in the lower priority queues



Multilevel Queue Scheduling

- Another possibility is to time slice among the queues
- Each queue gets a certain portion of the CPU time, which it can then schedule among its various processes
  - The foreground queue can be given 80% of the CPU time for RR scheduling
  - The background queue can be given 20% of the CPU time for FCFS scheduling

Multilevel Queue Scheduling

Assume there are 2 queues:- Q1(using RR scheduling with quantum =8) for foreground processes and Q2(using FCFS scheduling) for background processes. Consider following processes arriving in the system

Process	Burst time	Type
P1	12	FG
P2	8	BG
P3	20	FG
P4	7	BG

Calculate average waiting time assuming that processes in Q1 will be executed first.(Fixed priority scheduling)

P1	P3	P1	P3	P3	P2	P4
Q1	Q1	Q1	Q1	Q1	Q2	Q2

0 8 16 20 28 32 40 47

- **Waiting time of P1=(20-12) =8**
- **Waiting time of P2 = (40-8)=32**
- **Waiting time of P4 =(47-7)=40**
- **Waiting time of P3 = (32-20)= 12**
- **Average waiting time = (8 + 32 + 40 + 12)/4=23**

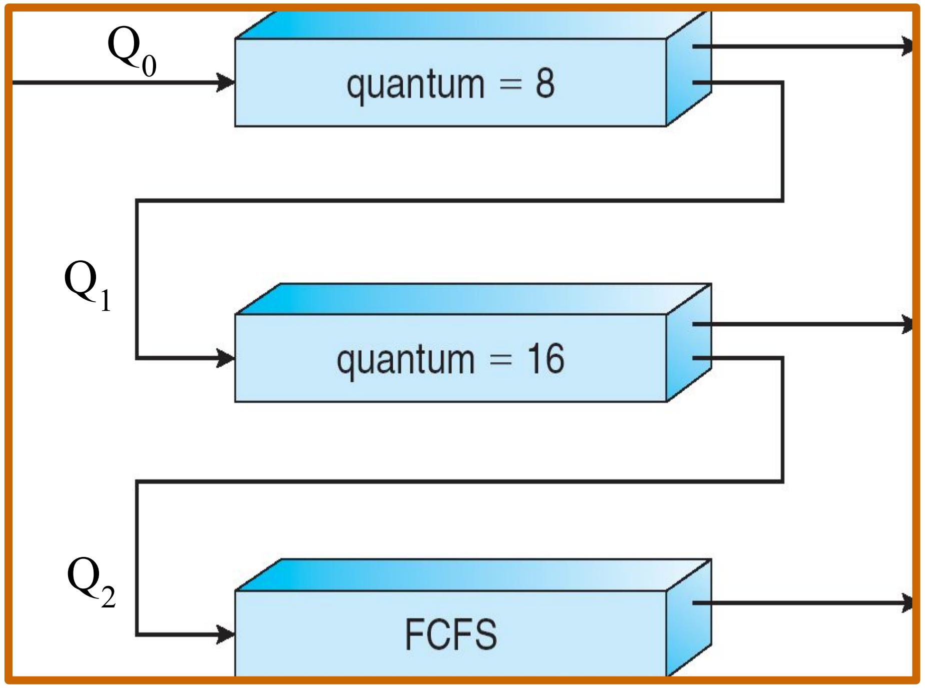
4.3 Multi-level Feedback Queue Scheduling

Multilevel Feedback Queue Scheduling

- In multi-level feedback queue scheduling, a process can move between the various queues; **aging** can be implemented this way.
- A multilevel-feedback-queue scheduler is defined by the following parameters:
  - Number of queues
  - Scheduling algorithms for each queue
  - Method used to determine when to <u>promote</u> a process
  - Method used to determine when to <u>demote</u> a process
  - Method used to determine which queue a process will enter when that process needs service

Example of Multilevel Feedback Queue

- Scheduling
  - A new job enters queue Q_0 *(RR)* and is placed at the end. When it gains the CPU, the job receives 8 milliseconds. If it does not finish in 8 milliseconds, the job is moved to the end of queue Q_1.
  - A Q_1 (RR) job receives 16 milliseconds. If it still does not complete, it is preempted and moved to queue Q_2 (FCFS).



Multilevel Feedback Queues

- Consider the following set of processes:

Processes	Arrival time	Burst time	Waiting Time
P1	0	17	
P2	12	25	
P3	28	8	
P4	36	32	
P5	46	18	

4.4 Multiple-Processor Scheduling

Multiple-Processor Scheduling

- If multiple CPUs are available, load sharing among them becomes possible; the scheduling problem becomes more complex
- We concentrate in this discussion on systems in which the processors are identical (homogeneous) in terms of their functionality
  - We can use any available processor to run any process in the queue
- Two approaches: **Asymmetric** processing and **symmetric** processing (see next slide)

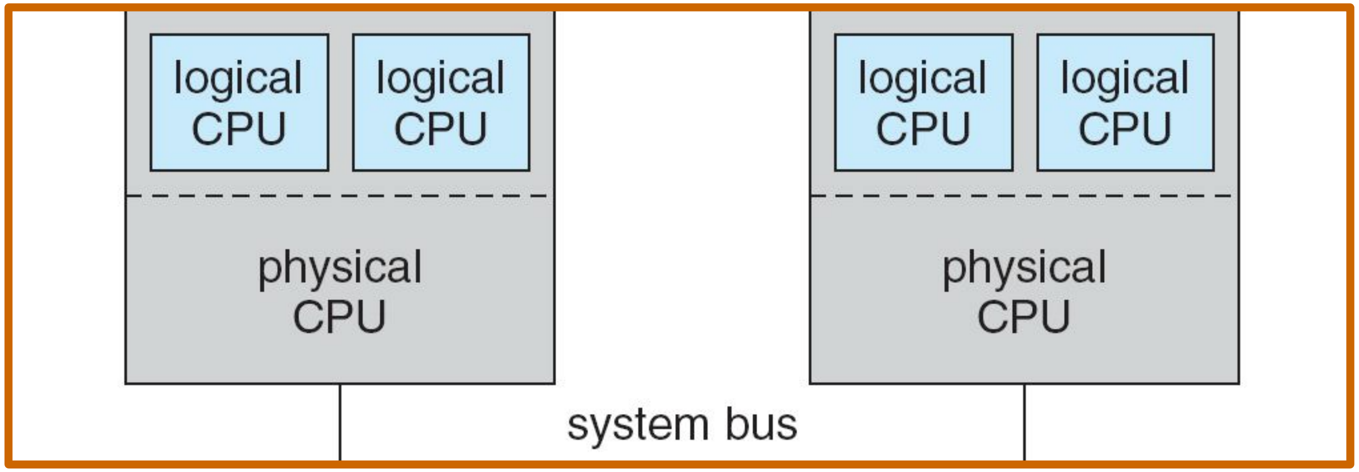
Multiple-Processor Scheduling

- ***Asymmetric multiprocessing (ASMP)***
  - One processor handles all scheduling decisions, I/O processing, and other system activities
  - The other processors execute only user code
  - Because only one processor accesses the system data structures, the need for data sharing is reduced
- **Symmetric multiprocessing (SMP)**
  - Each processor schedules itself
  - All processes may be in a common ready queue or each processor may have its own ready queue
  - Either way, each processor examines the ready queue and selects a process to execute
  - Efficient use of the CPUs requires load balancing to keep the workload evenly distributed
    - In a **Push** migration approach, a specific task regularly checks the processor loads and redistributes the waiting processes as needed
    - In a **Pull** migration approach, an idle processor pulls a waiting job from the queue of a busy processor
  - Virtually all modern operating systems support SMP, including Windows XP, Solaris, Linux, and Mac OS X

Symmetric Multithreading

- Symmetric multiprocessing systems allow several threads to run concurrently by providing multiple physical processors
- An alternative approach is to provide multiple **logical** rather than **physical** processors
- Such a strategy is known as **symmetric multithreading** (SMT)
  - This is also known **as hyperthreading technology**
- The idea behind SMT is to create multiple logical processors on the same physical processor
  - This presents a view of several **logical processors** to the operating system, even on a system with a single physical processor
  - Each logical processor has its own **architecture state**, which includes general-purpose and machine-state registers
  - Each logical processor is responsible for its **own interrupt handling**
  - However, each logical processor **shares** the resources of its physical processor, such as **cache memory and buses.**
- **SMT is a feature** provided in the **hardware**, not the software
  - The hardware must provide the representation of the architecture state for each logical processor, as well as interrupt handling (see next slide)

A typical SMT architecture



SMT = Symmetric Multi-threading