ECE437/CS481

M03A: CPU SCHEDULING
SCHEDULING CONCEPT & ALGORITHMS

CHAPTER 6.1-6.3

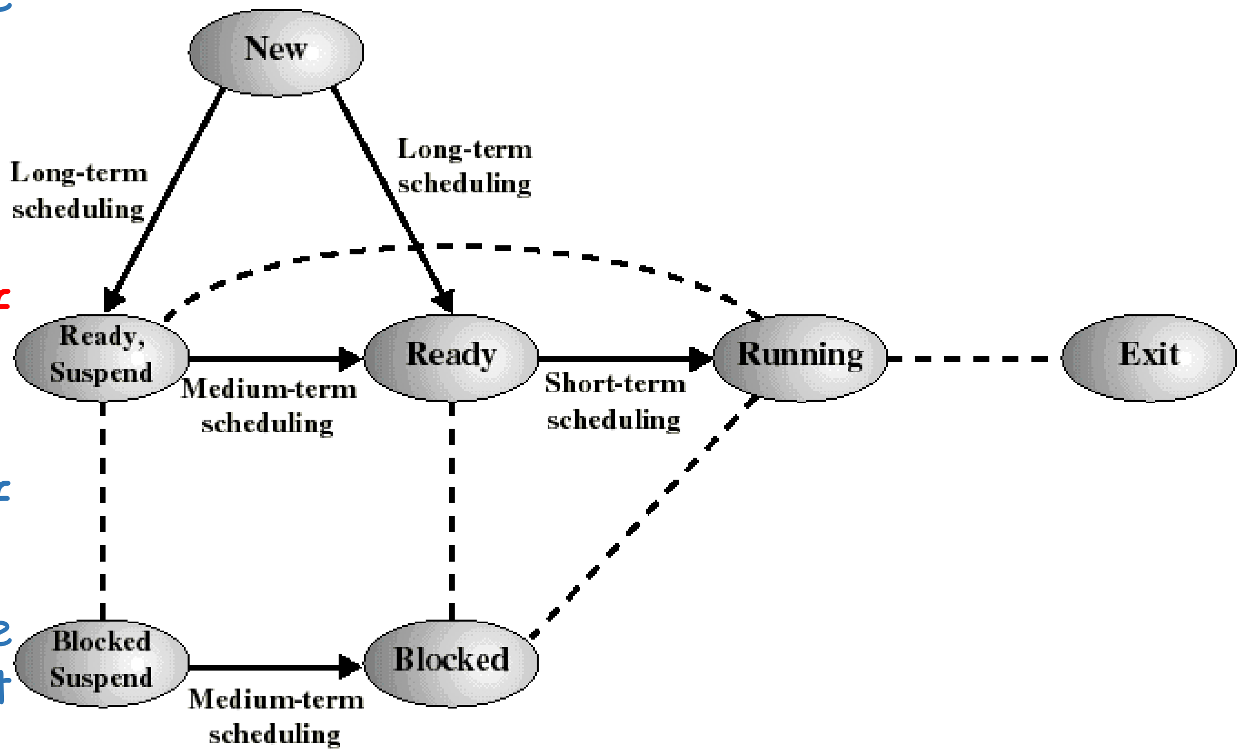
Xiang Sun

The University of New Mexico

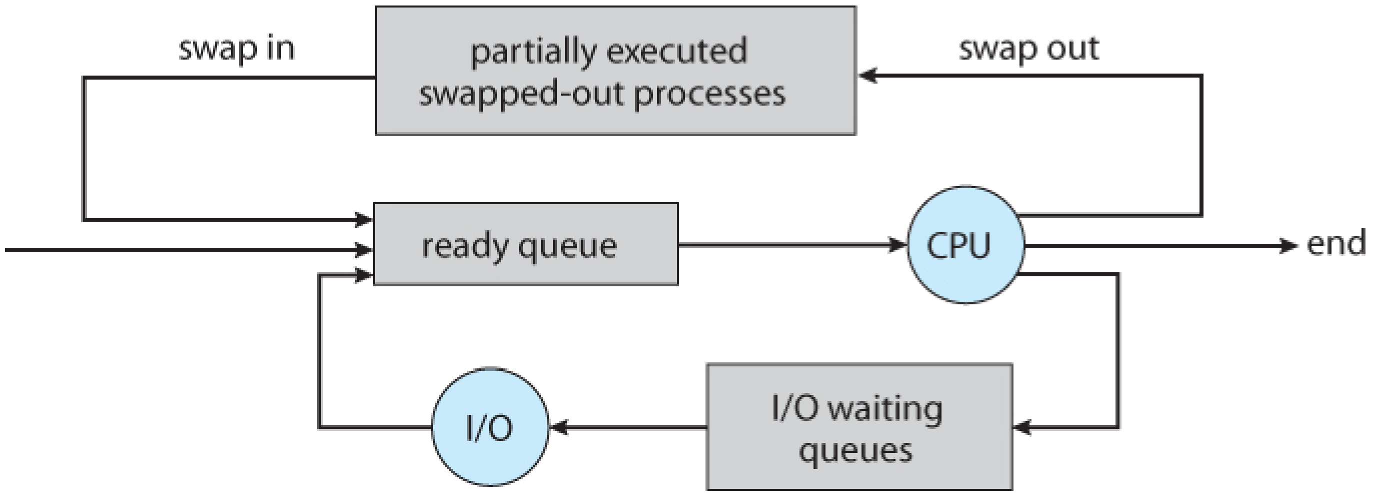
Levels of Scheduling

❑ Long-term scheduling/Job scheduler (High-level)

- Determine which process should be brought into the ready queue
- Invoked infrequently
- Coarse-grained control of the degree of multiprogramming
- Should balance different types of processes:
  - I/O-intensive process: spends more time doing I/O than computations, many short CPU bursts.
  - CPU-intensive process: spends more time doing computations; few very long CPU bursts.



❑ Medium-term scheduling (swapping in/out)



- Adjust the degree of multiprogramming by swapping.
- Swapping: removes a process from memory, stores on disk, and bring back in from disk to continue execution later on.
  - ✓ Swap out: memory-to-disk
  - ✓ Swap in: disk-to-memory
- Normally, the medium-term scheduler may decide to swap out a process which has not been active for some time, or a process which has a low priority, or a process which is taking up a large amount of memory in order to free up main memory for other processes, etc.

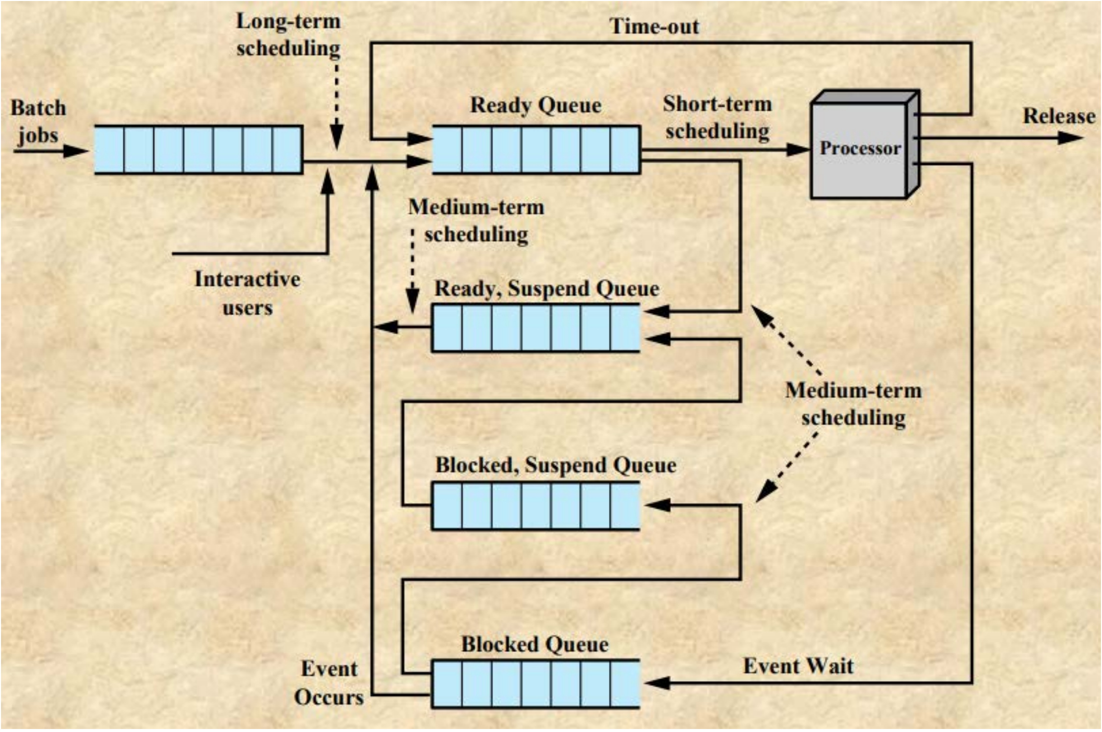
Levels of Scheduling

❑ Short-term scheduling/CPU scheduling

- Determine which process in the ready queue will be executed by the CPU

- Invoked when an event occurs that may lead to the blocking of the current process or that may provide an opportunity to preempt a currently running process in favor of another
  - ✓ Clock interrupts
  - ✓ I/O interrupts
  - ✓ Operating system calls
  - ✓ Signals, etc.

- Scheduler is consuming CPU too! It executes most frequently, must be very careful about its computation overhead.



Scheduling Objectives

❑ Metrics—system-wide

- Maximize processor/CPU utilization
  - ✓ percentage of time that CPU is running users' processes, to keep system as busy as possible
- Maximize throughput
  - ✓ number of processes completed per time unit
  - ✓ number of instruction executed per time unit
- Fairness
  - ✓ don't starve any processes—treat the all the same

Scheduling Objectives

❑ Metrics—per process/user-oriented

- Minimize waiting time
  - ✓ time spent in ready queue
- Minimize turnaround time
  - ✓ for a batch job, time between the submission of a process and the completion of the process, equal to sum of waiting time in ready queue and blocking queue, plus running time.
- Minimize response time
  - ✓ for interactive job, time from the submission of a request until the first response happens.

Scheduling Objectives

❑ Metrics

- Achieve a balance between response and utilization
- Minimize overhead (system level)
  - ✓ Context switching
  - ✓ Scheduling complexity
- It is difficult to find the optimal solution of the scheduling since the scheduling problem is mostly an NP-hard/NP-complete problem (e.g., job shop scheduling). Instead, looking for heuristic approaches is the common way.

Process Scheduling

❑ Non-preemption V.S. Preemption

- Non-preemption
  - ✓ once a process is scheduled to CPU, it continues execution until either it terminates, or relinquish (e.g. yield), or switches to a blocked state.
- Preemption
  - ✓ OS allowed to reallocate CPU (e.g. timeout, for higher priority process, etc.) to other process.

❑ Concurrency V.S. Parallelism

- Concurrency
  - ✓ schedule multiple processes onto a single CPU—time multiplex manner
- Parallelism
  - schedule multiple processes onto multiple CPUs—spatial multiplex

Process Scheduling

Scheduling algorithm		Dispatcher
Policy: to decide who gets to run	+	Mechanism: how to do the context switch

- Decision mode--when to do the scheduling
  - non-preemption
  - preemption—high priority or periodically (the process's time slice expires)

- Priority function
  - static information, e.g., job length, CPU-intensive or I/O intensive, memory requirement
  - dynamic information, e.g., deadline, arrival time, recent CPU consumption

Process Scheduling

Scheduling algorithm	+	Dispatcher
Policy: to decide who gets to run		Mechanism: how to do the context switch

- Events affect the scheduling algorithm:

1) Current process goes from running to waiting state (e.g., wait for I/O)
2) Current process terminates, running to exit state
3) Current process goes from running to read state (e.g., time slice is up)
4) I/O is complete --- Current process goes from waiting to ready

Process Scheduling

Scheduling algorithm	+	Dispatcher
Policy: to decide who gets to run		Mechanism: how to do the context switch

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

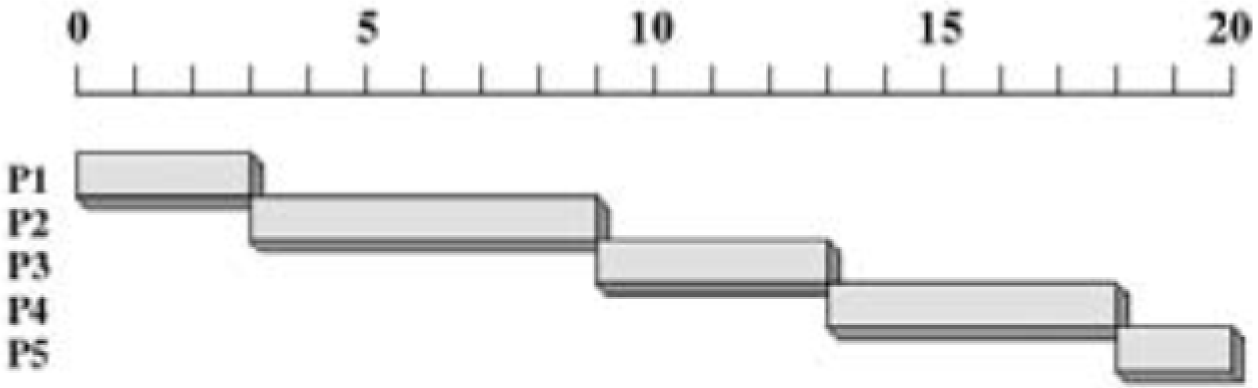
  1) switching context
  2) switching between user mode and kernel mode
  3) jumping to the proper location in the user program to restart that program

- Dispatch latency: time consumption of the dispatcher to stop one process and start another.

Scheduling Algorithms

- First-Come, First-Served (FCFS)
  - General specification
    - ✓ Decision mode: non-preemption Priority: equal
  - As a process become ready, it join the ready queue, scheduler always selects process from the front of the ready queue.

Process	Arrival time	Service time (Burst time)
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2



Waiting time for P1 = 0; P2 = 1; P3 = 5, P4= 7, P5 = 10,
Average waiting time: (0 + 1 + 5 + 7 + 10)/5 = 4.6

Scheduling Algorithms

First-Come, First-Served (FCFS)

- Simple to implement; low overhead, since no priority calculation, no extra context switch.
- Average waiting time may be long and suffer from convoy effect.
  - ✓ Convoy effect: long waiting time due to the slow processes.

Case 1:

- ✓ suppose that the processes arrive in the order :P1, P2, P3.
- ✓ The Gantt Chart for the schedule is

- ✓ Waiting time P1=0; P2=24; P3=27
- ✓ Average waiting time: (0+24+27)/3=17

Process	Service time
P1	24
P2	3
P3	3

Case 2:

- ✓ suppose that the processes arrive in the order :P2, P3, P1.
- ✓ The Gantt Chart for the schedule is

- ✓ Waiting time P1=6; P2=0; P3=3
- ✓ Average waiting time: (6+0+3)/3=3

Scheduling Algorithms

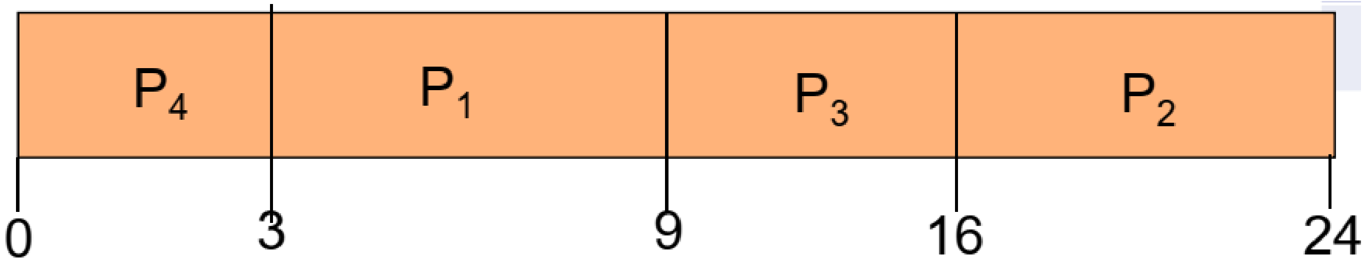
- Shortest Job Next (SJN)/Shortest Job First(SJF)
  - General specification
    - ✓ Decision mode: non-preemption by default (could be implemented as preemption)
    - ✓ Priority: 1/service time—a process with a lower service time has a higher priority
  - Pros:
    - ✓ Decrease average waiting time as compared to FCFS
  - Cons:
    - ✓ Higher complexity as compared to FCFS
    - ✓ Difficult to predict service time
    - ✓ Risk to starving longer process, as long as there are shorter processes around

- Shortest Remaining-time First (SRF)
  - General specification
    - ✓ Decision mode: preemption
    - ✓ Priority: 1/(service time-running time)
  - Still need estimate of service time, and can starve longer processes.
  - May give better turnaround time than SJN/SJF.

Scheduling Algorithms

❑ Shortest Job Next (SJN)/Shortest Job First(SJF)

- Suppose that the processes arrive in the order: P1 , P2 , P3 , P4
- The Gantt Chart for the schedule is:

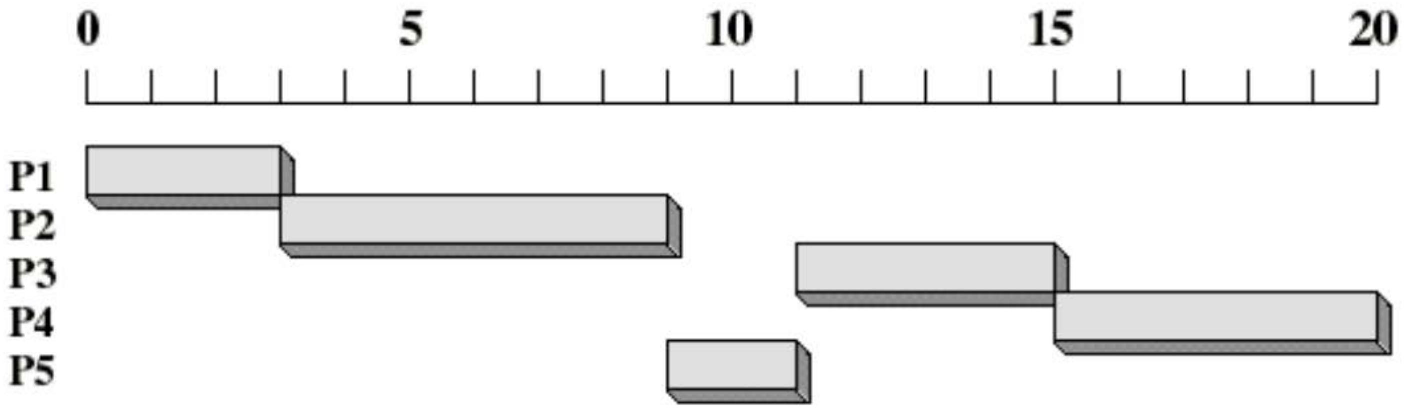


Process	Service time
P1	6
P2	8
P3	7
P4	3

- Waiting time for P1=3 , P2=16 , P3=9 , P4=0
- Average waiting time: (3+16+9+0)/4=7
- Suppose FCFS is applied, average waiting time: (0+6+14+21)/4=10.25

Scheduling Algorithms

❑ Shortest Job Next (SJN)/Shortest Job First(SJF)



Process	Arrival time	Service time
P1	0	3
P2	2	6
P3	4	4
P4	6	5
P5	8	2

- Waiting time for P1=0 , P2=1, P3=7 , P4=9, P5=1.
- Average waiting time: (0+1+7+9+1)/5=3.6

❑ Shortest Job Next (SJN)/Shortest Job First(SJF)

- How to predict the service/burst time of a process?
- In reality, the prediction is done by using the length of previous service/burst time of the process.

1. t_n = actual length of n^{th} CPU burst
2. τ_{n+1} = predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Define : $\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\tau_n$.

Shortest Job Next (SJN)/Shortest Job First(SJF)

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

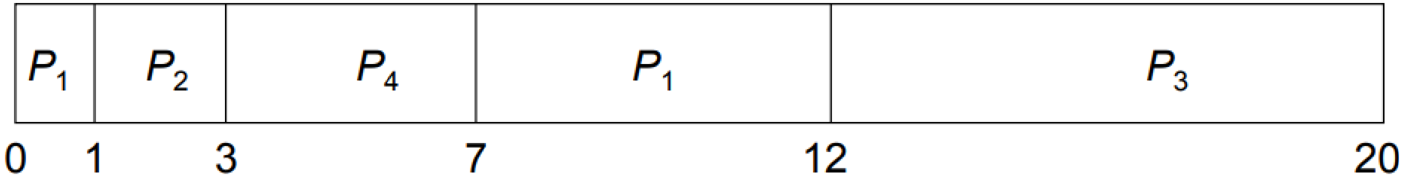
- If $\alpha=0$
  - ✓ $\tau_{n+1} = \tau_n$
  - ✓ No weightage to recent history
- If $\alpha=1$
  - ✓ $\tau_{n+1} = \alpha\, t_n$
  - ✓ Only the actual last CPU burst counts
  - ✓ If $0 < \alpha < 1$, we expand the formula:

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

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

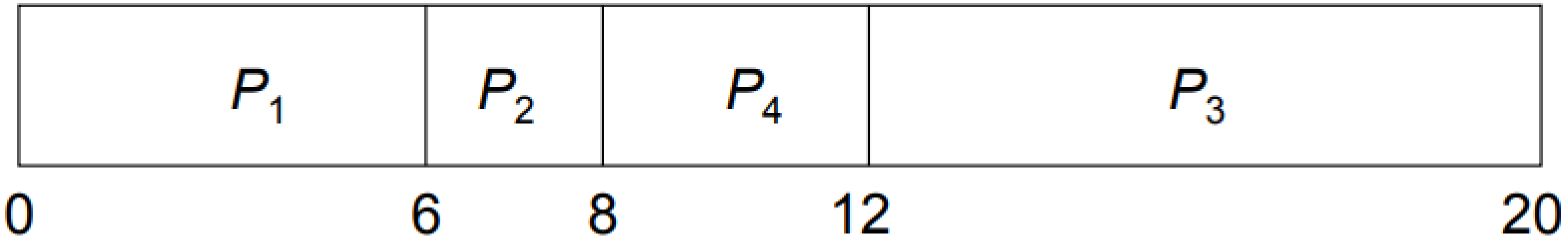
Shortest Remaining-time First (SRF)

- Suppose we have four processes arriving in the order showing in the table.
- The Gantt Chart for the schedule is:



Process	Arrival time	Service time
P1	0	6
P2	1	2
P3	2	8
P4	3	4

- Waiting time for P1=6 , P2=0 , P3=10 , P4=0
- Average waiting time: (6+0+10+0)/4=4
- Suppose SJN is applied, average waiting time: (0+5+10+5)/4=5



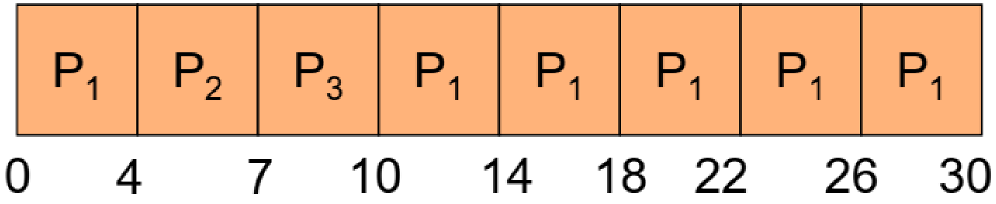
Scheduling Algorithms

Round Robin (RR)

- General specification
  - decision: preemption, periodically with time slice
  - priority: equal
- Use preemption based on clock - time slicing (time quantum), generate interrupt at periodic intervals, usually 10-100ms
- When interrupt occurs, place running process back to Ready queue, select next process to run by using FCFS.
- Designed especially for interactive jobs.
- Average waiting time may be long.
- What's the right length of a time slice?
  - short means short processes move through quickly, but high overhead to deal with process scheduling and context switching.
  - very long time slice making it degenerate into FCFS.
  - should be slightly greater than time of typical job burst time.

Round Robin (RR)

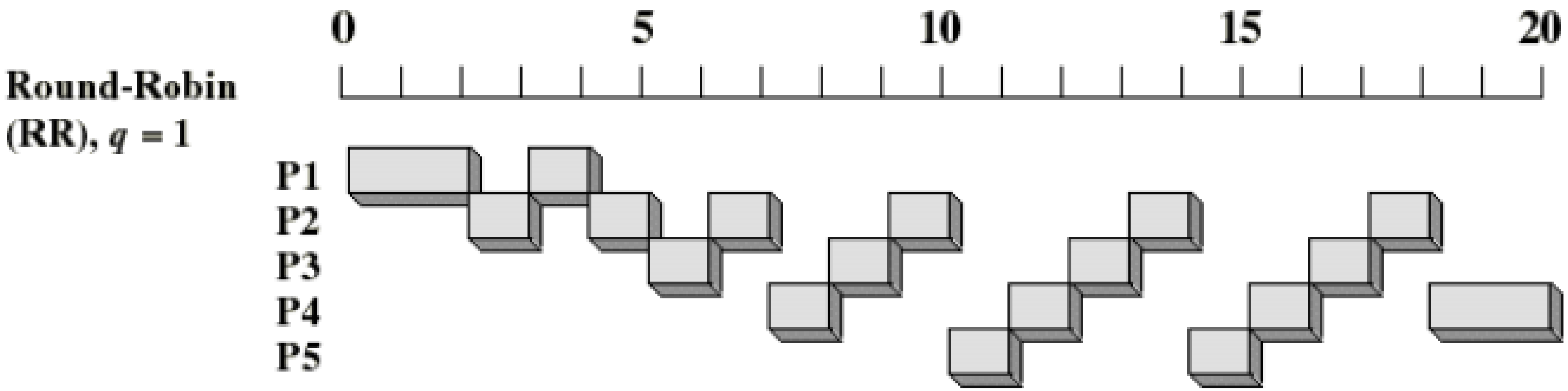
- Suppose that the processes arrive in the order: P1 , P2 , P3. The time quantum length is 4.
- The Gantt Chart for the schedule is:



Process	Service time
P1	24
P2	3
P3	3

- Wait time for P1=6; P2=4; P3=7
- Average waiting time: (6+4+7)/3=5.67.
  - ✓ Recall that the average waiting time for FCFS is 17. However, Ave waiting time (RR) < Ave waiting time (FCFS) is not always true.
- Typically, RR incurs higher average turnaround time, but lower response time than FCFS.

Round Robin (RR)



Process	Arrival time	Service time (Burst time)
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

Waiting time for P1 = 1; P2 = 10; P3 = 10, P4= 9, P5 = 5,
Average waiting time: (1+ 10 + 10 + 9 + 5)/5 = 7
Recall that the average waiting time for FCFS is 4.6

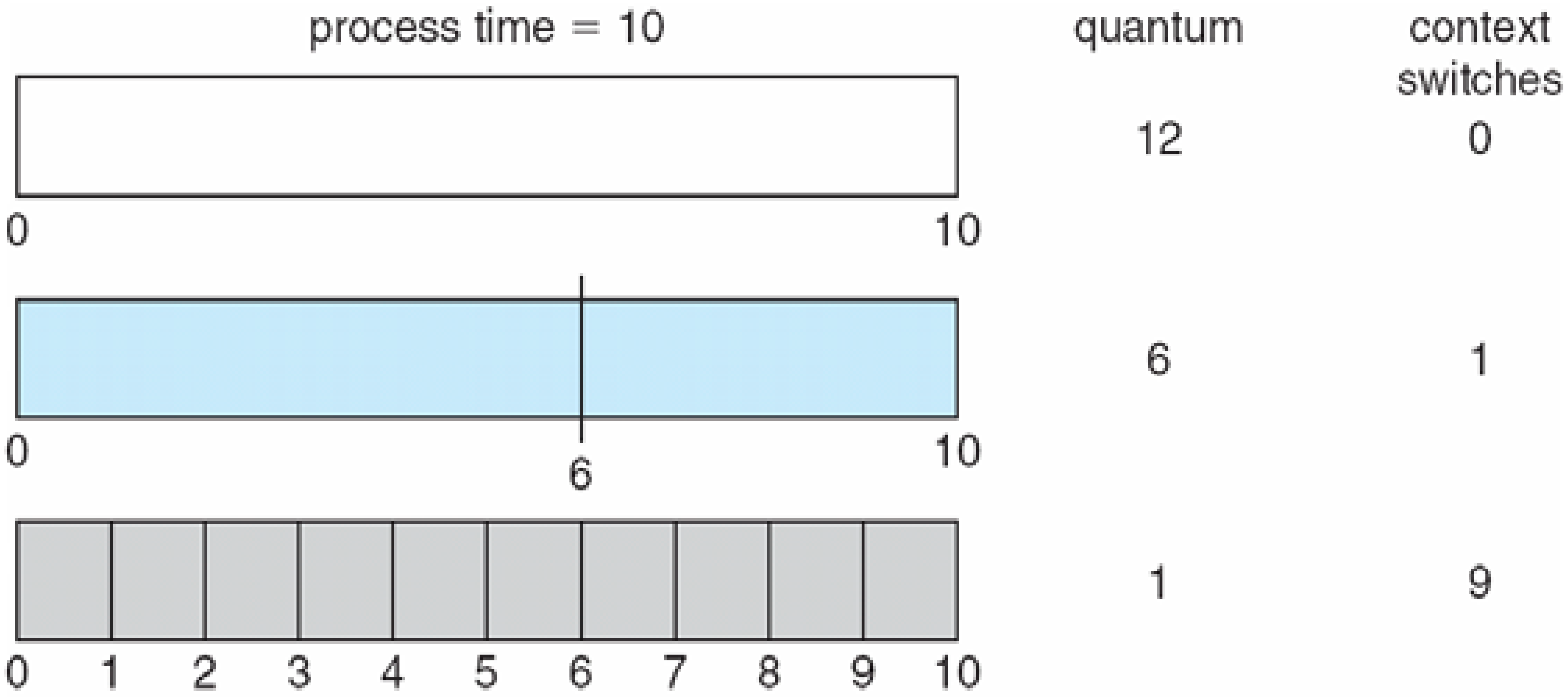
❑ Round Robin (RR)

- Assumption of RR
  - ✓ there are n processes in the ready queue, and
  - ✓ the time quantum length 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 large ($q \rightarrow +\infty$), RR→FIFS
  - ✓ If q is small, RR incurs frequent process context switch, thus generating high overhead.

Scheduling Algorithms

❑ Round Robin (RR)

➢ Time Quantum and Context Switch



Round Robin (RR)

- Time quantum length selection.
  - Must be substantially larger than the time required to handle the clock interrupt and dispatching (i.e., dispatch latency).
  - Should be larger than the typical length of interaction (but not much more) to avoid penalizing I/O intensive processes.