Scheduling Algorithms

Instructor: William Zheng
Email:
william.zheng@digipen.edu
PHONE EXT: 1745

Lecture Outline

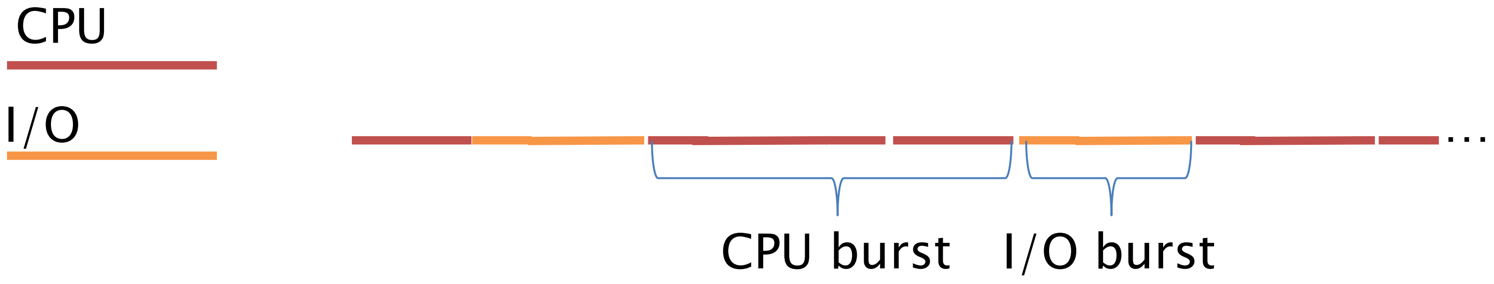
- Scheduling Intro
  - Preemptive and Non-preemptive algos
  - Types of Processes
  - Scheduling rubrics
- Non-preemptive Scheduling Algos
  - FCFS (First Come First Serve)
  - SJF
- Preemptive Scheduling Algos
  - Round-Robin
  - Multilevel Queues
  - Multilevel Queues with feedback

Preemptive and Non-preemptive Scheduling

- Non-preemptive schedules *only* when
  - A process terminates
  - A process goes into waiting state
- Preemptive schedules *also* when
  - A hardware interrupt arrives
    - Process goes from waiting to ready state.
    - Process goes from running to ready state.
  - A software-generated interrupt (system call)
    - I/O request occurs requesting to access a file on hard disk

Types of processes

- What processes/threads do?
  - Compute
  - I/O
- Types of processes/threads
  - I/O bound
  - Compute/CPU bound



Scheduling Effectiveness

- Waiting time — Enter Ready Queue → Process Start to run
- Turnaround time — Enter Ready Queue → Process Terminates
- Response time — I/O Request → I/O Processing

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)

First Come First Serve

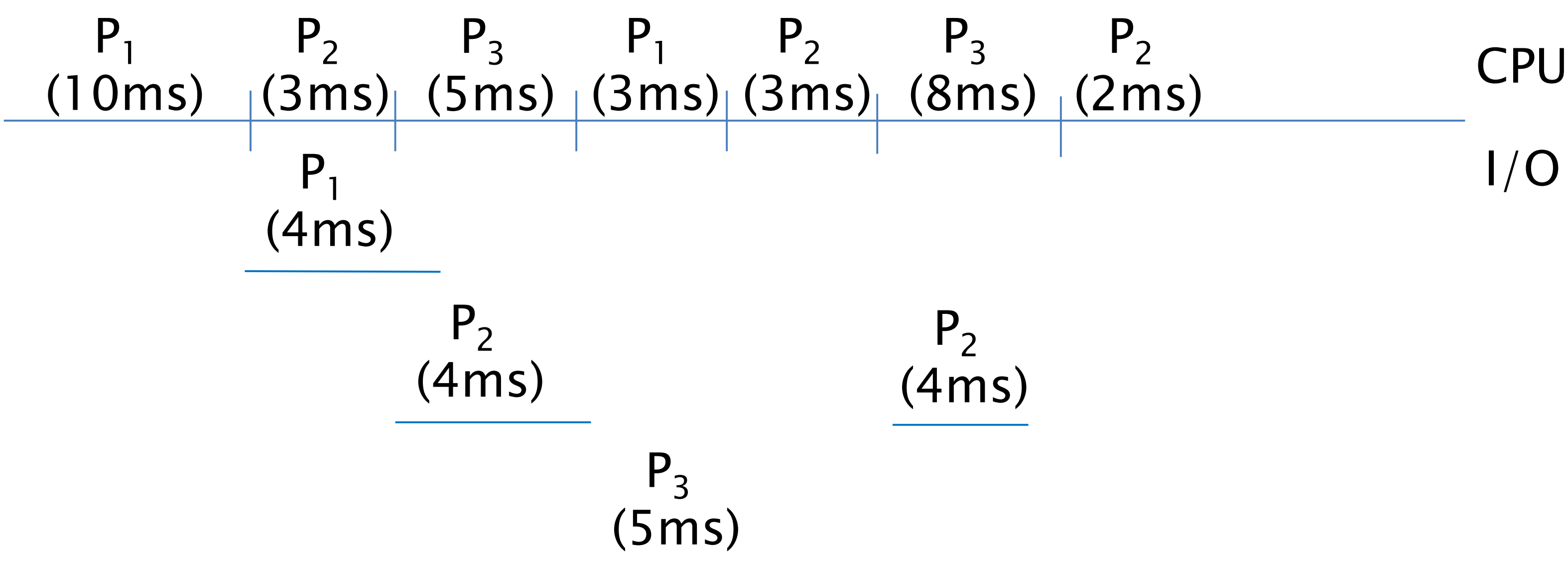
- Schedule according to arrival order
- Implemented using a queue data struct
  - What's a queue data struct?
  - Basically a linked-list
    - Always read from front of linked list
    - Always add to end of the linked list

FCFS waiting time example

- P_1 19 ms
- P_2 7 ms
- P_3 5 ms
- Schedule
- Wait time(P_1) = 0 ms
- Wait time(P_2) = 19 ms
- Wait time(P_3) = 26 ms
- Average wait time =
  - (0 ms + 19 ms + 26 ms)/3 = 15 ms

FCFS scheduling with I/O

- P_1 (10ms CPU, 4ms I/O, 3ms CPU)
- P_2 (3ms CPU, 4ms I/O, 3ms CPU, 4ms I/O, 2ms CPU)
- P_3 (5ms CPU, 5ms I/O, 8ms CPU)
- Schedule:



Priority Scheduling

- A **priority queue** is a type queue with two operations
  - Put – place an item on the queue
  - Get – get the item with *highest* value item from the queue
- Same as FCFS scheduling algorithm, except that the queue is replaced by a priority queue
- What priority?
  - Based on feature of the jobs OR
  - A user-assigned priority value OR
  - Dynamic run-time priorities (pre-emptive versions)

Shortest Job First

- Also known as *Shortest Remaining Time First*.
  - Shortest-next-CPU-burst algo
  - Associates with each process the length of the process's next CPU burst, rather than total length
  - Assigned to process with the smallest next CPU burst when CPU available
    - FCFS scheduling used to break the tie if the next CPU bursts of 2 processes are the same
- Non-preemptive priority scheduling
  - Always schedule the job with shortest job or shortest remaining time.
  - Minimizes average waiting time.

SJF waiting time example

- P_1 19 ms
- P_2 7 ms
- P_3 5 ms
- Schedule
- Wait time(P_1) = 12 ms
- Wait time(P_2) = 5 ms
- Wait time(P_3) = 0 ms
- Average wait time =
  (12 ms + 5 ms + 0 ms)/3 = 17/3 ms

SJF scheduling with I/O

- P_1 (10ms CPU, 4ms I/O, 3ms CPU)
- P_2 (3ms CPU, 4ms I/O, 3ms CPU, 4ms I/O, 2ms CPU)
- P_3 (5ms CPU, 5ms I/O, 8ms CPU)
- Schedule (according to next CPU burst):

Process in Ready Q before scheduling	P_1 10ms P_2 3ms P_3 5ms	P_1 10ms P_3 5ms	P_1 10ms P_2 3ms	P_1 10ms	P_2 2ms P_3 8ms	P_3 8ms	P_1 3ms
Scheduled Process	P_2 3ms	P_3 5ms	P_2 3ms	P_1 10ms	P_2 2ms	P_3 8ms	P_1 3ms
Process in Waiting State after scheduling		P_2 4ms	P_3 3ms(of 5ms)	P_2 4ms P_3 2ms	P_1 2ms	P_1 2ms	

Priority scheduling problem

- **Starvation**: a low priority job may sit on the queue indefinitely
  - Happens if there is always an incoming job request with a higher priority than the lowest priority job on the queue
- **Aging** as a solution:
  - if a job has waited on the queue for a sufficiently long time, its priority is increased by a set amount

Pre-emptive scheduling

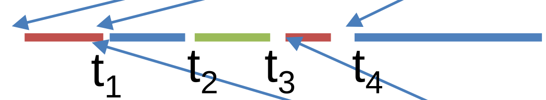
- Each job is allowed a time slice, or **time quantum**, in which to execute
- Once a time quantum has passed, or if the job has finished within that time, a different job gets a time slice
- A scheduling algorithm is used to determine the order in which jobs get a time slice.
  - **Round robin (RR) scheduling**
  - **Multilevel queue scheduling**
  - **Multilevel feedback queue scheduling**

Round robin scheduling

- Uses a FIFO queue to process jobs in a first-come first-serve order
  - Get a job from the front of the queue
  - Let it execute for a maximum of one time quantum
  - If it does not complete within that time
    - Pause the job, and put the job at the back of the queue
  - Repeat

RR waiting time example (5ms slices)

- P_1 7 ms
- P_2 19 ms
- P_3 5 ms
- Schedule



Waiting time: Sum of the periods spent waiting in the ready queue

1st part of waiting time for P_2 is 5ms
2nd part of waiting time for P_2 is t_4-t_2=7ms, total=5+7=12ms, i.e. total time spent in the ready queue for the remaining job.

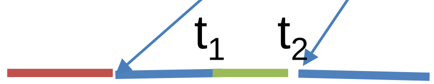
1st part of waiting time for P_1 is 0ms
2nd part of waiting time for P_1 is t_3-t_1=5(P_2)+5(P_3) =10ms

- Wait time(P_1) = 10ms
- Wait time(P_2) = 12ms
- Wait time(P_3) = 10ms
- Average wait time =

$$(10 \text{ ms} + 12 \text{ ms} + 10 \text{ ms})/3 = 32/3 \text{ ms}$$

RR waiting time example (10ms slices)

- P_1 7 ms
- P_2 19 ms
- P_3 5 ms
- Schedule

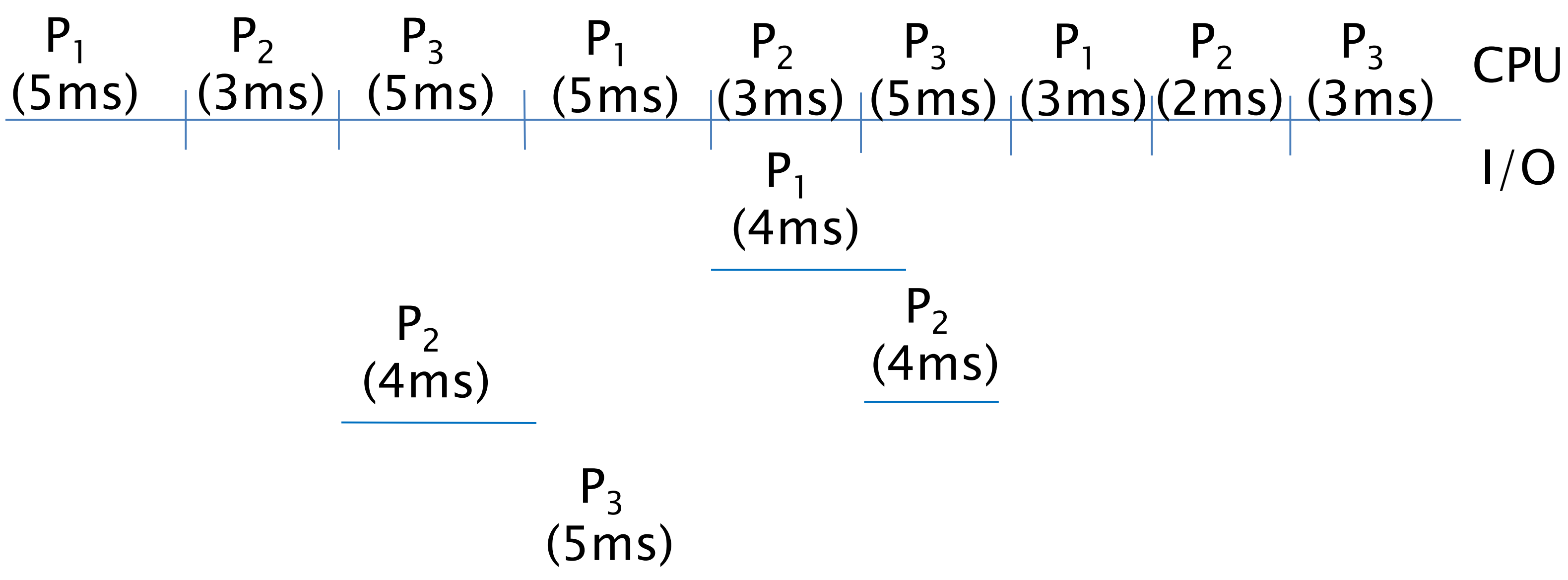


1st part of waiting time for P_2 is 7ms
2nd part of waiting time for P_2 is t_2-t_1=5(P_3)=5ms

- Wait time(P_1) = 0ms
- Wait time(P_2) = 12ms
- Wait time(P_3) = 17ms
- Average wait time =
  (0 ms + 12 ms + 17 ms)/3 = 29/3 ms

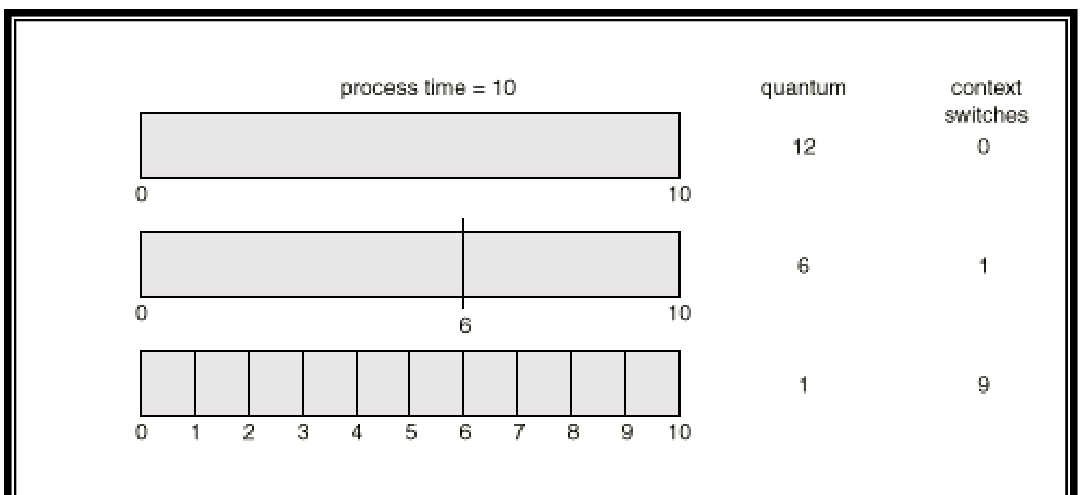
RR scheduling with I/O (5 ms slices)

- P_1 (10ms CPU, 4ms I/O, 3ms CPU)
- P_2 (3ms CPU, 4ms I/O, 3ms CPU, 4ms I/O, 2ms CPU)
- P_3 (5ms CPU, 5ms I/O, 8ms CPU)
- Schedule:



RR considerations

- The average waiting time decreases (in general) with decreasing time quantum size
- Context switching causes more processing time
- A balance must be made between time quantum size and context switching time
  - Want the time quantum to be "large" with respect to the context switching time



Multilevel queue scheduling

- Ready queue is partitioned into separate queues
  - Jobs are put on different queues according to prioritization criteria
    - System jobs might have top priority
    - User jobs might have less priority
    - Background jobs might have least priority
- Each queue has its own scheduling algorithm
- Scheduling must be done between the queues. E.g.,
  - Fixed priority scheduling; (i.e., serve all from foreground then from background).
    Problem: 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
    Problem: how to split?

Multilevel feedback queue scheduling

- Uses multiple queues, as in multilevel scheduling
- However, a job may be moved between each of the different level queues, depending on CPU usage
- If a job has long CPU burst cycles, it is put on a lower priority queue

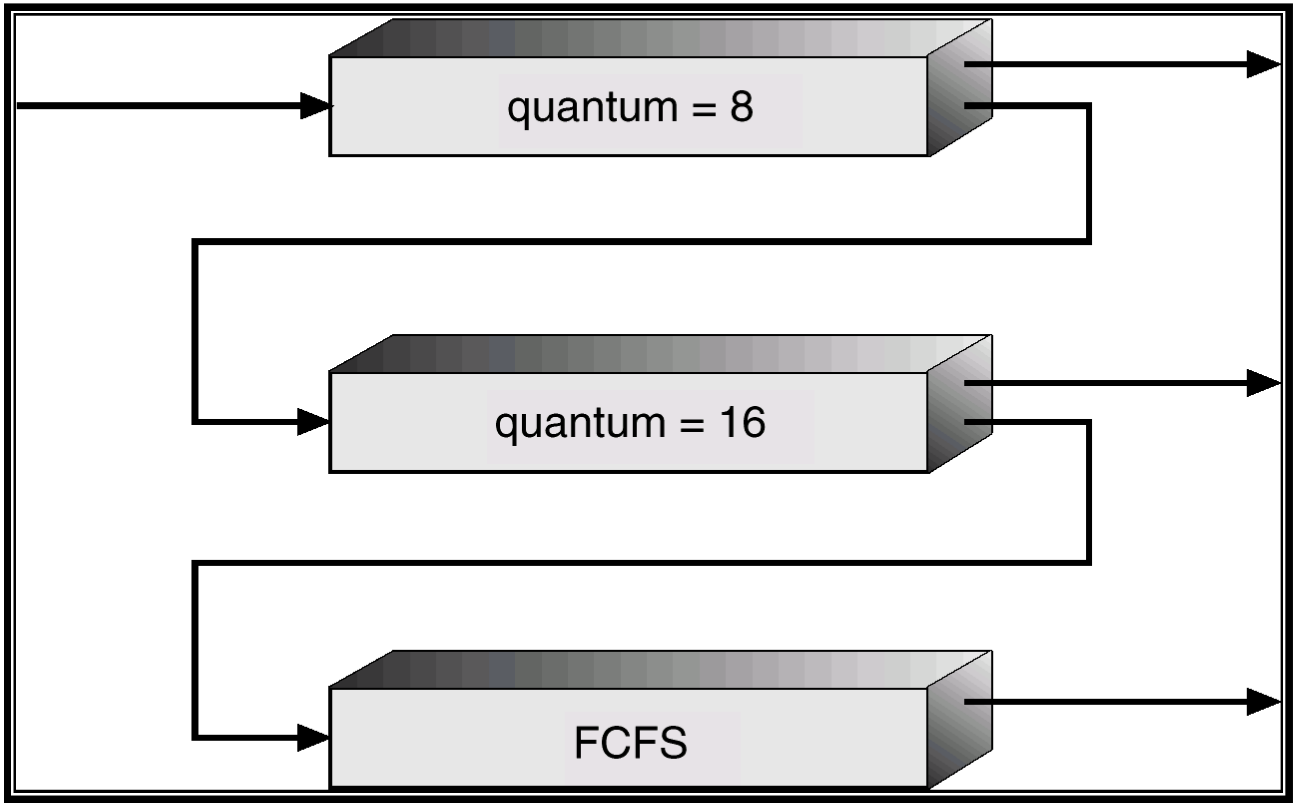
Multilevel feedback queue scheduling

- 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 – time quantum 8 milliseconds
  - Q_1 – 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.

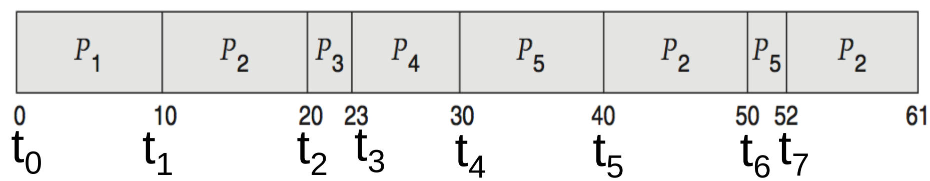
Multilevel Feedback Queues



Evaluation (1)

- Consider 5 processes arriving at time 0:

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



Preemptive Round Robin Scheduling (time quantum=10ms):

Waiting time for P_1 = 0 ms

Waiting time for P_2 =$(t_1-t_0)+(t_5-t_2)+(t_7-t_6)$ =10+20+2 = 32 ms

Waiting time for P_3 = t_2-t_0 = 20 ms

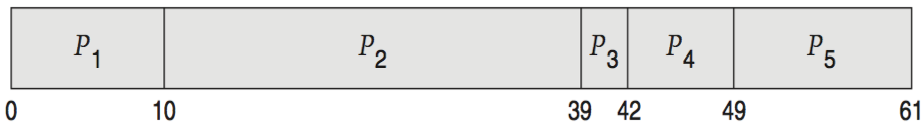
Waiting time for P_4 = t_3-t_0 = 23 ms

Waiting time for P_5 = $(t_4-t_0)+(t_6-t_5)$ = 40 ms

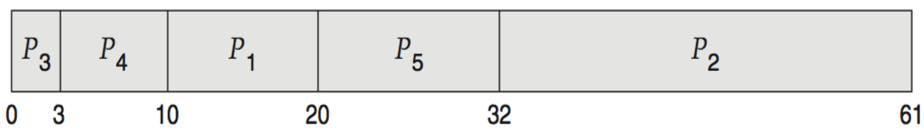
Average waiting time = (0+32+20+23+40)/5 = 23ms

Evaluation (2)

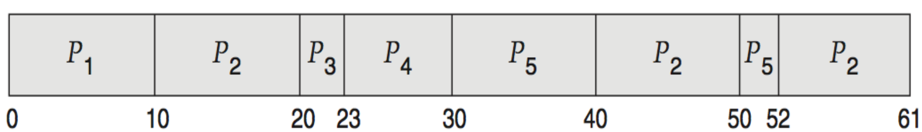
- FCFS is 28ms:



- Non-preemptive SFJ is 13ms:



- RR is 23ms:

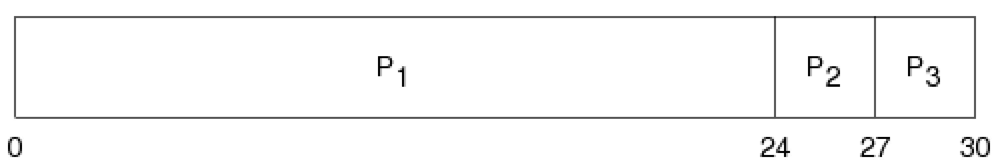


Process	Burst Time
P_1	10
P_2	29
P_3	3
P_4	7
P_5	12

Evaluation (3)

- Average turnaround time: (24+27+30)/3 = 27ms

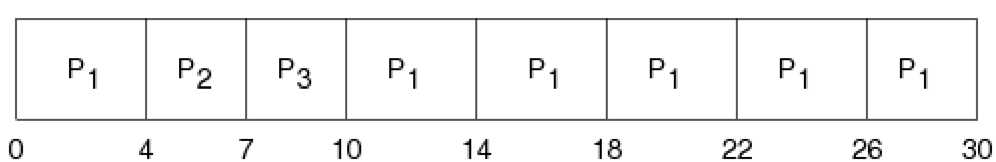
Process	Burst Time	Arrival	Start	Wait	Finish	TA
1	24	0	0	0	24	24
2	3	0	24	24	27	27
3	3	0	27	27	30	30



Evaluation (4)

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

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



Turnaround Time and Response Time

- Turnaround time
  - the interval from the time of submission of a process to the time of completion
  - Sum of the periods spent waiting to get into memory, waiting in the ready queue, executing on CPU, and doing I/O
- Response Time
  - The time from submission of a request until the first response is produced
  - The time taken to start responding
  - Not the time taken to output the response

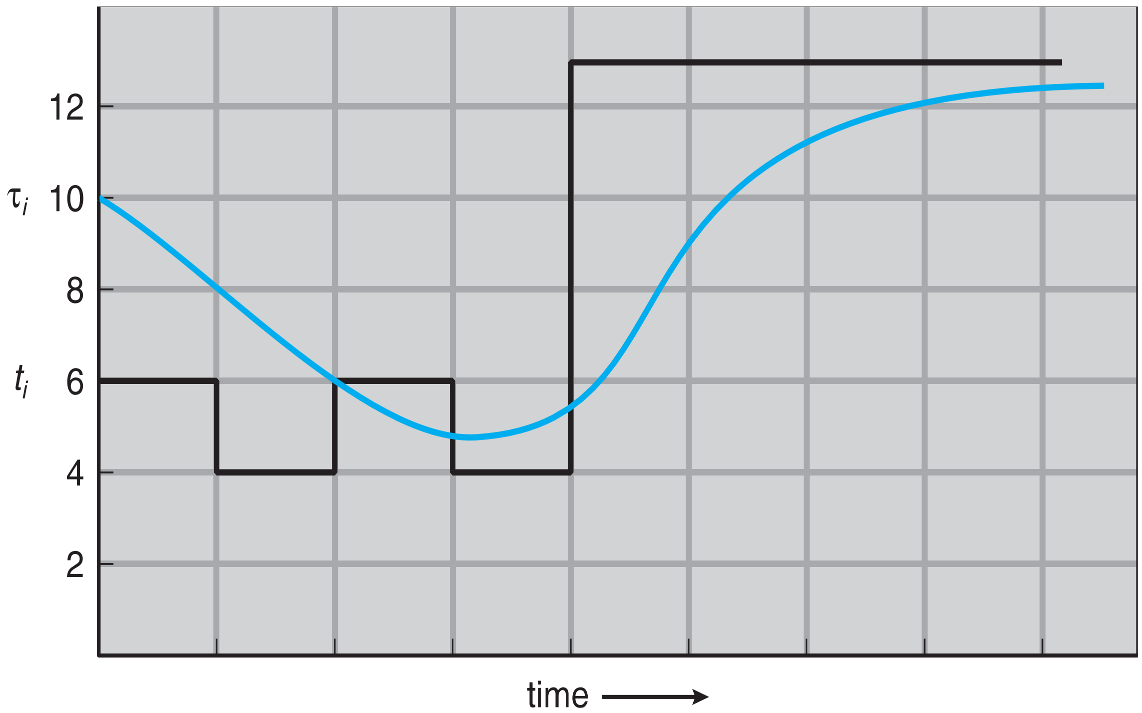
Determining Length of Next CPU Burst

- Can only estimate the length – should be similar to the previous one
  - Then pick process with shortest predicted next CPU burst
- Can be done by using the length of previous CPU bursts, using exponential averaging

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

- Commonly, α set to ½
- Preemptive version called **shortest-remaining-time-first**

Prediction of the Length of the Next CPU Burst



CPU burst (t_i)		6	4	6	4	13	13	13	…
"guess" (τ_i)	10	8	6	6	5	9	11	12	…