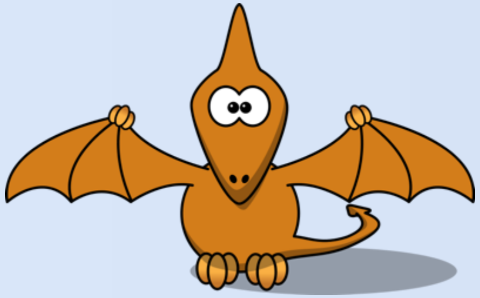
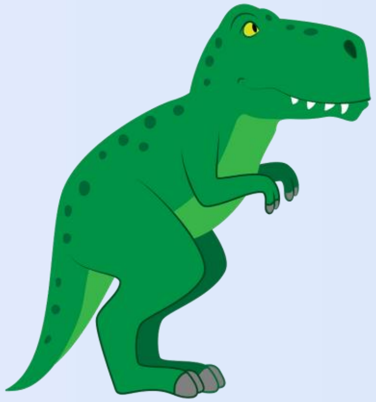


Chapter 5.

CPU Scheduling

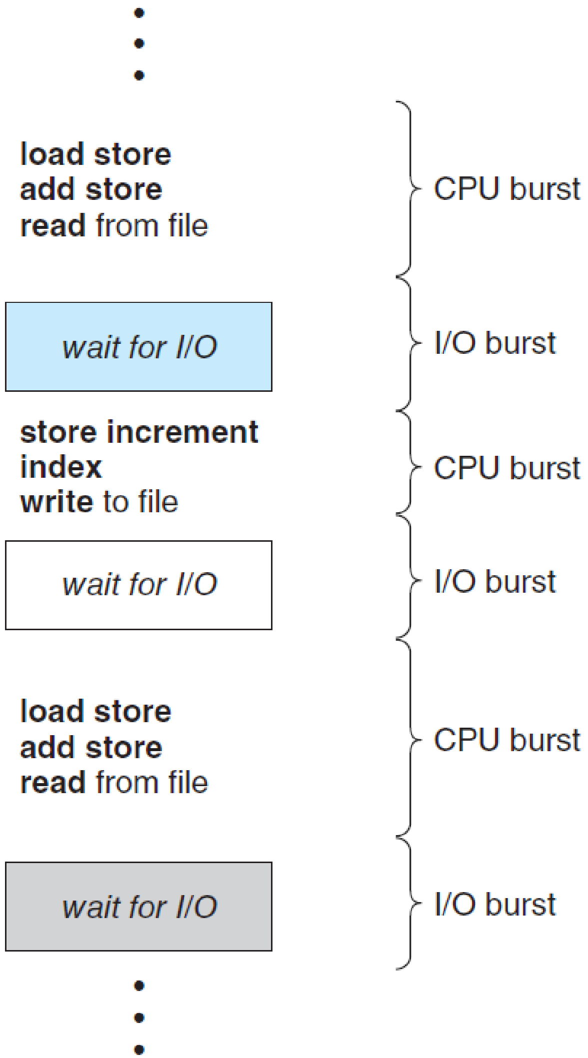


Operating System Concepts (10th Ed.)

5.1 Basic Concepts

- CPU *scheduling* is
  - the basis of multiprogrammed operating systems.
  - The objective of *multiprogramming* is
    - to have some processes running at all times
    - to maximize CPU utilization.

5.1 Basic Concepts



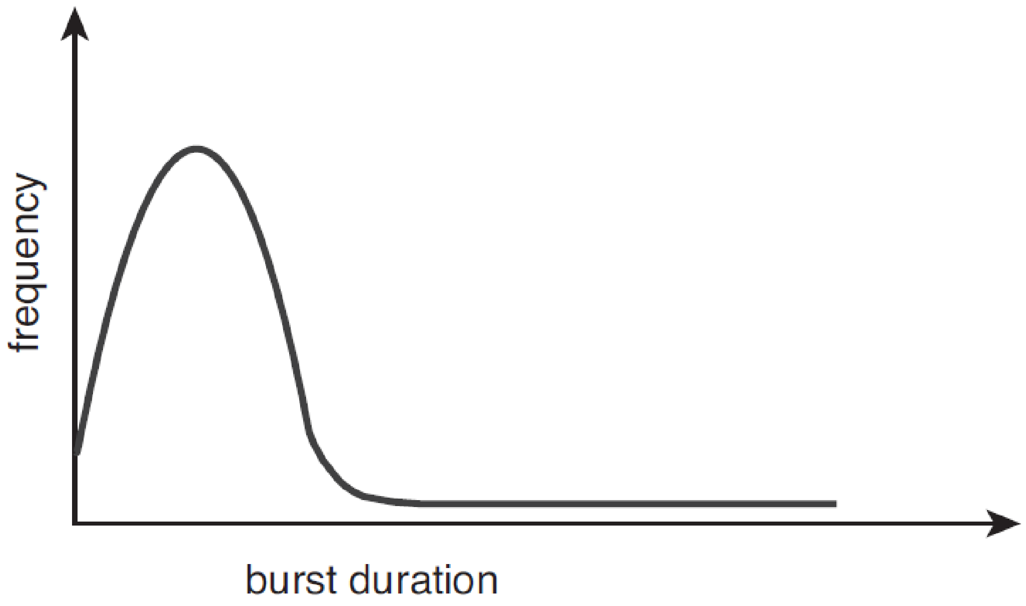


Figure 5.1 *Histogram of CPU-burst duration*

Figure 5.1 *Alternating sequence of CPU-bursts and I/O-bursts.*

5.1 Basic Concepts

- CPU scheduler
  - selects a process from the processes in memory
    - that are *ready* to execute and *allocates* the CPU to that process.
  - Then, how can we select a next process?
    - Linked List? or Binary Tree?
    - FIFO Queue: First-In, First-Out
    - Priority Queue: How can we determine the priority of a process?

5.1 Basic Concepts

- ***Preemptive*** .vs. ***Non-preemptive***:

  - ***Non-preemptive*** scheduling
    - a process keeps the CPU until it releases it,
    - either by terminating or by switching to the waiting state.

  - ***Preemptive*** scheduling
    - a process can be preempted by the scheduler.

5.1 Basic Concepts

- Decision Making for CPU-scheduling:
  1. When a process switches from the *running* to *waiting* state.
  2. When a process switches from the *running* to *ready* state.
  3. When a process switches from the *waiting* to *ready* state.
  4. When a process *terminates*.

  - No. 1 & 4: no choice – non-preemptive.
  - No. 2 & 3: choices – preemptive or non-preemptive.

5.1 Basic Concepts

- The ***dispatcher*** is
  - a module that gives control of the CPU's core
    - to the process selected by the CPU scheduler.
  - The functions of dispatcher:
    - switching context from one process to another
    - switching to user mode
    - jumping to the proper location to resume the user program

5.1 Basic Concepts

- The dispatcher should be as fast as possible
  - since it is invoked during every context switch.
  - The ***dispatcher latency*** is
    - the time to stop one process and start another running.

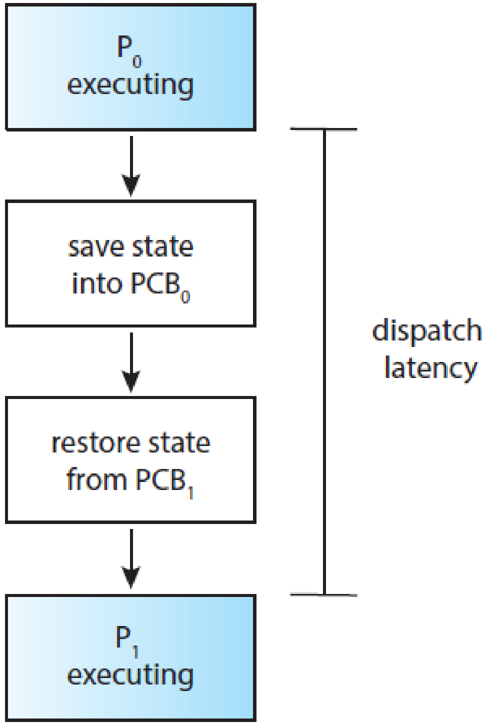


Figure 5.3 *The role of the dispatcher.*

5.1 Basic Concepts

- How *often* do *context switches* occur?

```
joonion@joonionpc:/proc/9$ vmstat 1 3
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd     free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 1  0      0 12879788   7896  53084    0    0     8   791    2    5  0  0 100  0  0
 0  0      0 12879796   7896  53084    0    0     0     0    4   30  0  0 100  0  0
 0  0      0 12879796   7896  53084    0    0     0     0    5   27  0  0 100  0  0
```

```
joonion@joonionpc:/proc$ cat /proc/1/status | grep ctxt
voluntary_ctxt_switches:        62
nonvoluntary_ctxt_switches:     0
```

5.2 Scheduling Criteria

- Scheduling Criteria
  - *CPU utilization*: to keep the CPU as busy as possible.
  - *Throughput*: the number of processes completed per time unit.
  - *Turnaround time*:
    - how long does it take to execute a process?
    - from the time of submission to the time of completion.
  - ***Waiting time***:
    - the amount of time that a process spends waiting in the ready queue.
    - the sum of periods spend waiting in the ready queue.
  - *Response time*:
    - the time it takes to start responding

5.3 Scheduling Algorithms

- CPU Scheduling Problem:
  - decide which of the processes in the ready queue
    - is to be allocated the CPU’s core.

5.3 Scheduling Algorithms

- The solutions for the scheduling problem:
  - **FCFS**: First-Come, First-Served
  - **SJF**: Shortest Job First (**SRTF**: Shortest Remaining Time First)
  - **RR**: Round-Robin
  - **Priority-based**
  - MLQ: Multi-Level Queue
  - MLFQ: Multi-Level Feedback Queue

5.3 Scheduling Algorithms

- **FCFS** Scheduling
  - First-Come, First-Served: the simplest CPU-scheduling algorithm.
  - The process that requests the CPU first
    - is allocated the CPU first.
    - can be easily implemented with a FIFO queue.

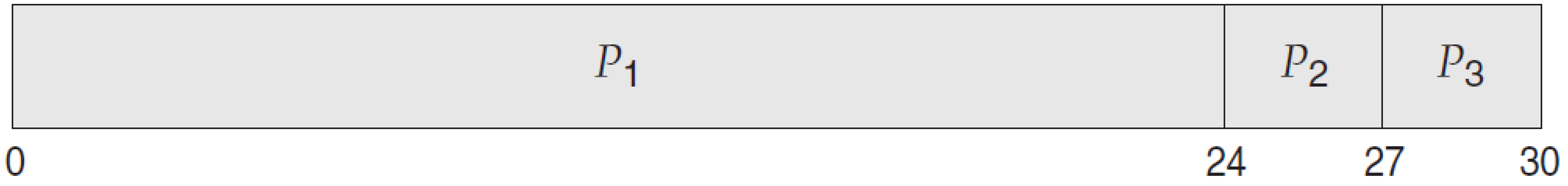
5.3 Scheduling Algorithms

- Consider the following set of processes
  - that arrive at time 0,
  - with the length of the CPU burst given in milliseconds:

Process	Burst Time
P_1	24
P_2	3
P_3	3

5.3 Scheduling Algorithms

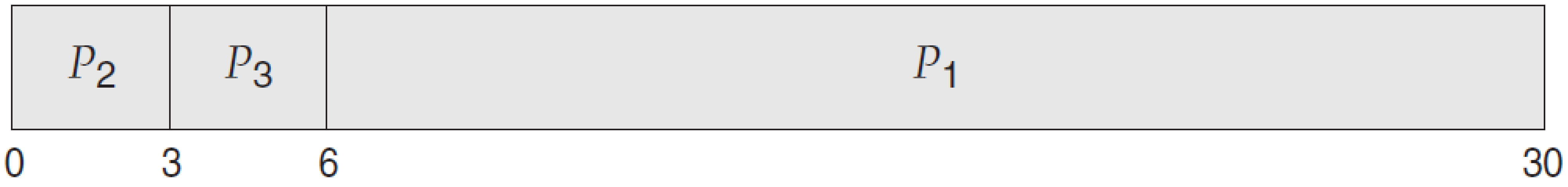
- If the processes arrive in the order P_1, P_2, P_3:
  - Gantt Chart served by the FCFS policy:



- Calculate the ***waiting time*** of this schedule.
  - Waiting Time for $P_1 = 0$, $P_2 = 24$, $P_3 = 27$
  - *Total* Waiting Time: $(0 + 24 + 27) = 51$
  - *Average* Waiting Time: $51/3 = 17$

5.3 Scheduling Algorithms

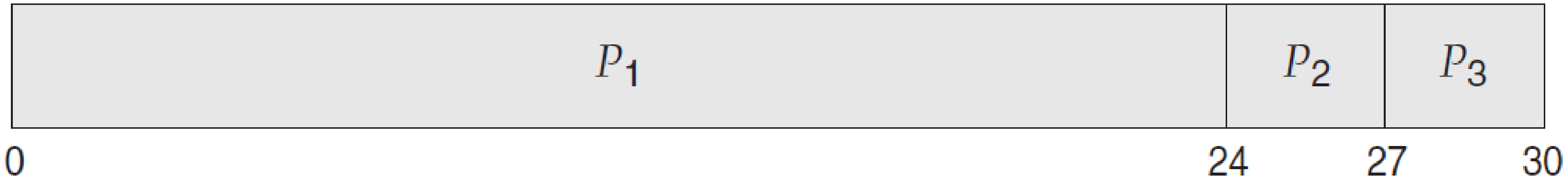
- If the processes arrive in the order P_2, P_3, P_1:
  - Gantt Chart served by the FCFS policy:



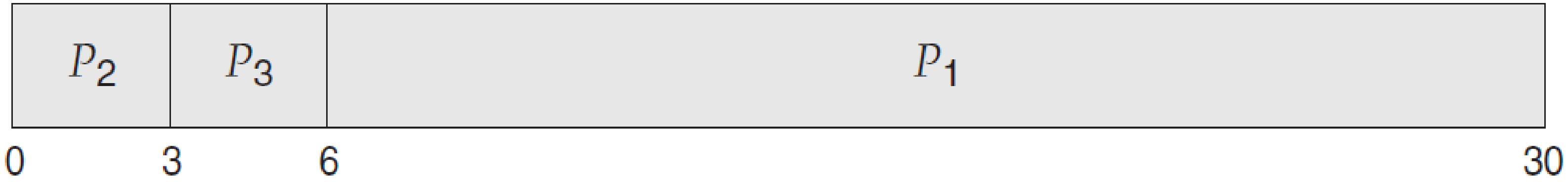
- Calculate the ***waiting time*** of this schedule.
  - Waiting Time for $P_1 = 6$, $P_2 = 0$, $P_3 = 3$
  - *Total* Waiting Time: $(6 + 0 + 3) = 9$
  - *Average* Waiting Time: $9/3 = 3$

5.3 Scheduling Algorithms

- Let us calculate the ***turnaround time***:



  - Turnaround Time for $P_1 = 24$, $P_2 = 27$, $P_3 = 30$
  - *Total* Turnaround Time: $(24 + 27 + 30) = 81$
  - *Average* Turnaround Time: $81/3 = 27$



  - Do it on your own:

5.3 Scheduling Algorithms

- Note that
  - The average waiting time under the ***FCFS*** policy
    - is generally *not minimal* and *may vary substantially*
    - if the processes' ***CPU-burst times*** vary greatly.
  - Preemptive or non-preemptive?
    - The FCFS scheduling algorithm is *non-preemptive*.

5.3 Scheduling Algorithms

- Note also that
  - The performance in a *dynamic* situation:
    - What if we have *one* CPU-bound and many I/O-bound processes?
  - *Convoy Effect*:
    - all the other processes wait for the one big process to get off the CPU.
    - results in lower CPU and device utilization than might be possible
    - if the shorter processes were allowed to go first.

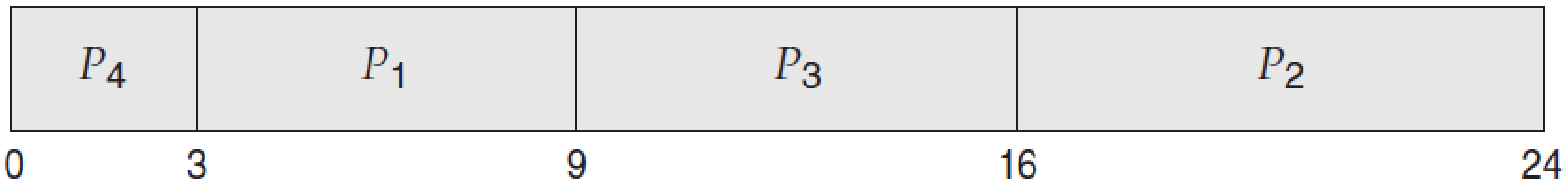
SJF Scheduling

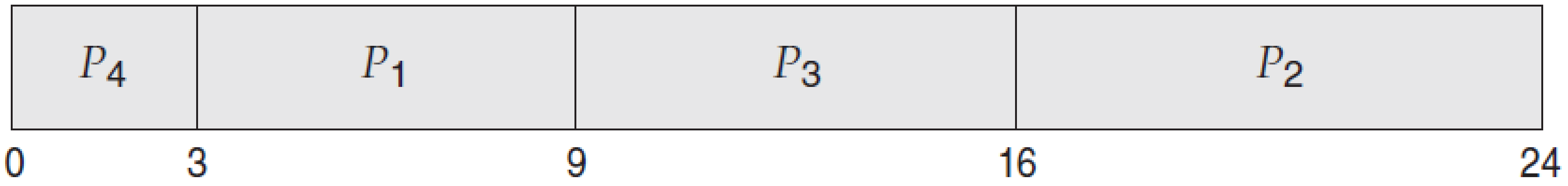
- Shortest-Job-First: *shortest-next-CPU-burst-first* scheduling.
- SJF associates with each process
  - the length of the process's next CPU burst.
- When the CPU is available,
  - assign it to the process that has the smallest next CPU burst.
- If two or more processes are even,
  - break the tie with the FCFS.

5.3 Scheduling Algorithms

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

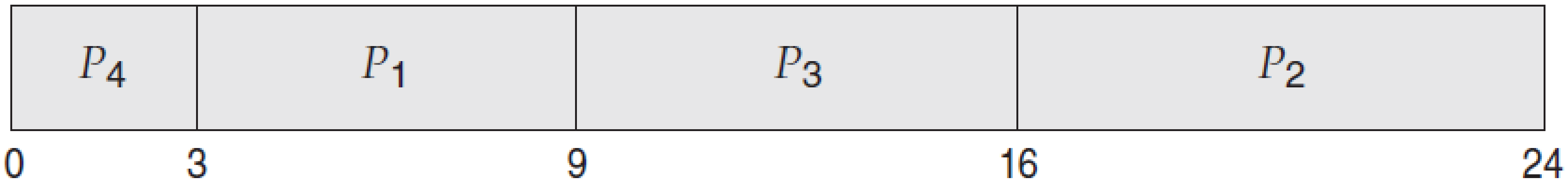
- Using the SJF scheduling,
  - Gantt Chart would be:





- Calculate the ***waiting time***:
  - Waiting Time for $P_1 = 3$, $P_2 = 16$, $P_3 = 9$, $P_4 = 0$
  - *Total* Waiting Time: $(3 + 16 + 9 + 0) = 28$
  - *Average* Waiting Time: $28/4 = 7$

5.3 Scheduling Algorithms



- Calculate the ***turnaround time***:
  - Turnaround Time for $P_1 = 9$, $P_2 = 24$, $P_3 = 16$, $P_4 = 3$
  - *Total* Turnaround Time: $(9 + 24 + 16 + 3) = 52$
  - *Average* Turnaround Time: $52/4 = 13$

5.3 Scheduling Algorithms

- Note that
  - The SJF scheduling algorithm is *provably optimal*,
    - it gives the *minimum average waiting time* for a given set of processes.
  - Moving a short process before a long one
    - decreases the waiting time of the short process
    - more than it increases the waiting time of the long process.
    - Consequently, the average waiting time decreases.

5.3 Scheduling Algorithms

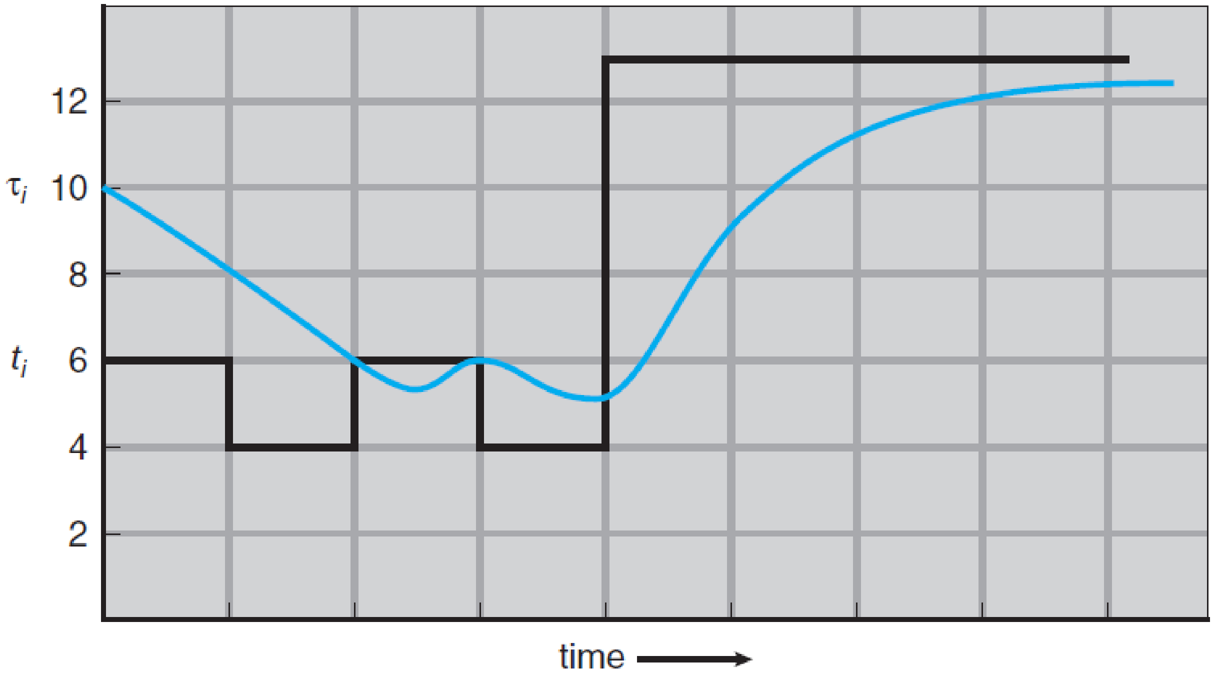
- Can you implement the SJF scheduling?
  - There is *no way* to know the length of the *next CPU burst*.

  - Try to *approximate* the SJF scheduling:
    - We may be able to ***predict*** the length of the next CPU.
    - Pick a process with the shortest *predicted* CPU burst.

5.3 Scheduling Algorithms

- How to predict the next CPU burst?
  - *exponential average* of the measured lengths of *previous* CPU burst.

  - $\boldsymbol{\tau_{n+1} = \alpha\tau_n + (1-\alpha)\tau_n}$, where
    - τ_n is the length of nth CPU burst,
    - τ_{n+1} is our predicted value for the next CPU burst,
    - for $0 \leq \alpha \leq 1$



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

$\alpha = 1/2$

Figure 5.4 *Prediction of the length of the next CPU burst.*

5.3 Scheduling Algorithms

- Note also that
  - The SJF algorithm can be either *preemptive* or *non-preemptive*.

  - The choice arises:
    - when a new process arrives at the *ready* queue
    - while a *previous* process is still executing.

  - What if a *newly* arrived process is *shorter* than
    - *what is left* of the currently executing process?

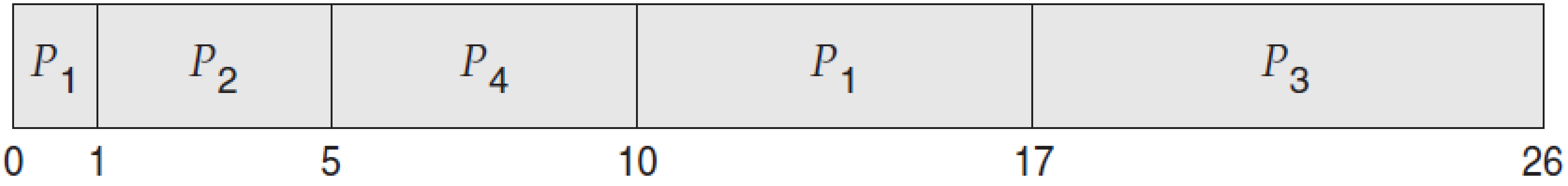
5.3 Scheduling Algorithms

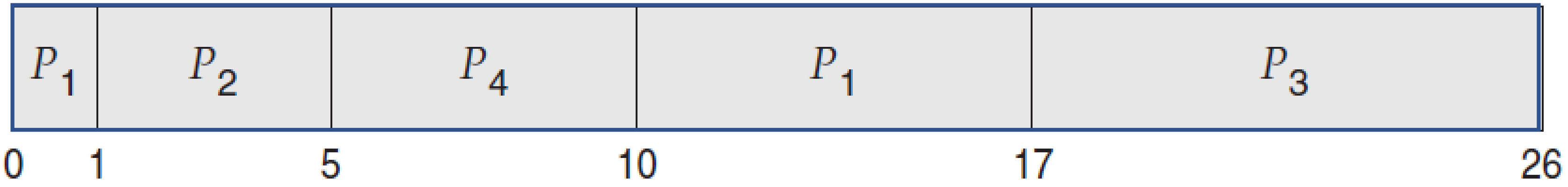
- SRTF Scheduling
  - Shortest-Remaining-Time-First: *Preemptive SJF* scheduling
  - SRTF will *preempt* the currently running process,
    - whereas a non-preemptive SJF will *allow* it to *finish* its CPU burst.

5.3 Scheduling Algorithms

- Consider the following:
  - if the processes arrive at the ready queue at the arrival time.

Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5





- The ***waiting time***:
  - *Total* Waiting Time: $[(10-1)+(1-1)+(17-2)+(5-3)] = 26$
  - *Average* Waiting Time: $26/4 = 6.5$

5.3 Scheduling Algorithms

- Draw a Gantt chart for non-preemptive SJF scheduling,
  - then calculate the average waiting time to be 7.75.

- Do the same thing with the *turnaround* time.
  - with a *non-preemptive SJF* scheduling algorithm
  - with a *preemptive SRTF* scheduling algorithm.

5.3 Scheduling Algorithms

- **RR** Scheduling
  - Round-Robin: *preemptive FCFS* with a *time quantum*.
  - A ***time quantum*** (or *time slice*) is a small unit of time.
    - generally from 10 to 100 milliseconds in length.
  - The ready queue is treated as a *circular queue*.
  - The scheduler goes around the ready queue,
    - allocating the CPU to each process
    - for a time interval of up to 1 time quantum.

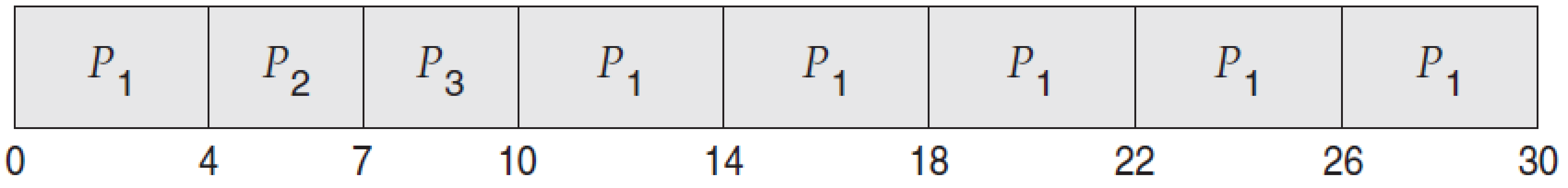
5.3 Scheduling Algorithms

- One of two things will happen:
  - The process may have a CPU burst of ***less than*** *one time quantum*.
    - the process itself will release the CPU voluntarily
    - the scheduler will proceed to the next process in the ready queue.
  - If the CPU burst is ***longer than*** *one time quantum*,
    - the timer will go off and will cause an interrupt to the OS.
    - a context switch will be executed,
    - the process will be put at the tail of the ready queue.

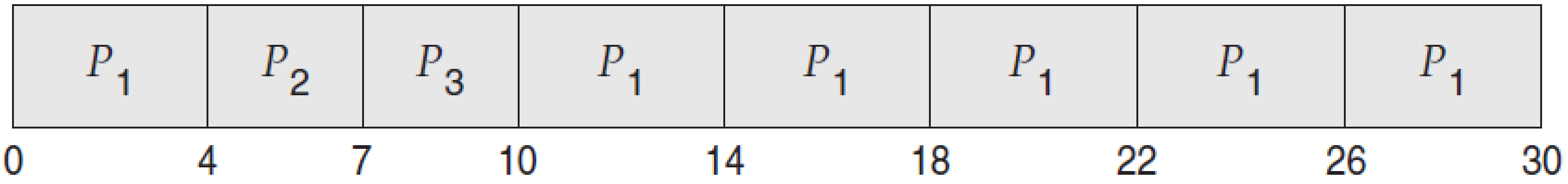
5.3 Scheduling Algorithms

Process	Burst Time
P_1	24
P_2	3
P_3	3

- When we use a time quantum of 4 milliseconds



5.3 Scheduling Algorithms



- The ***waiting time***:
  - Waiting Time for $P_1 = 10 - 4 = 6$, $P_2 = 4$, $P_3 = 7$
  - *Total* Waiting Time: $(6 + 4 + 7) = 17$
  - *Average* Waiting Time: $17/3 = 5.66$

5.3 Scheduling Algorithms

- Note that
  - The average waiting time under the RR policy is often long.
  - The RR scheduling algorithm is *preemptive*.
    - if a process's CPU burst exceeds one time quantum,
    - that process is *preempted* and is put back in the ready queue.

5.3 Scheduling Algorithms

- The performance of the RR scheduling algorithm
  - depends heavily on the ***size*** of the *time quantum*.

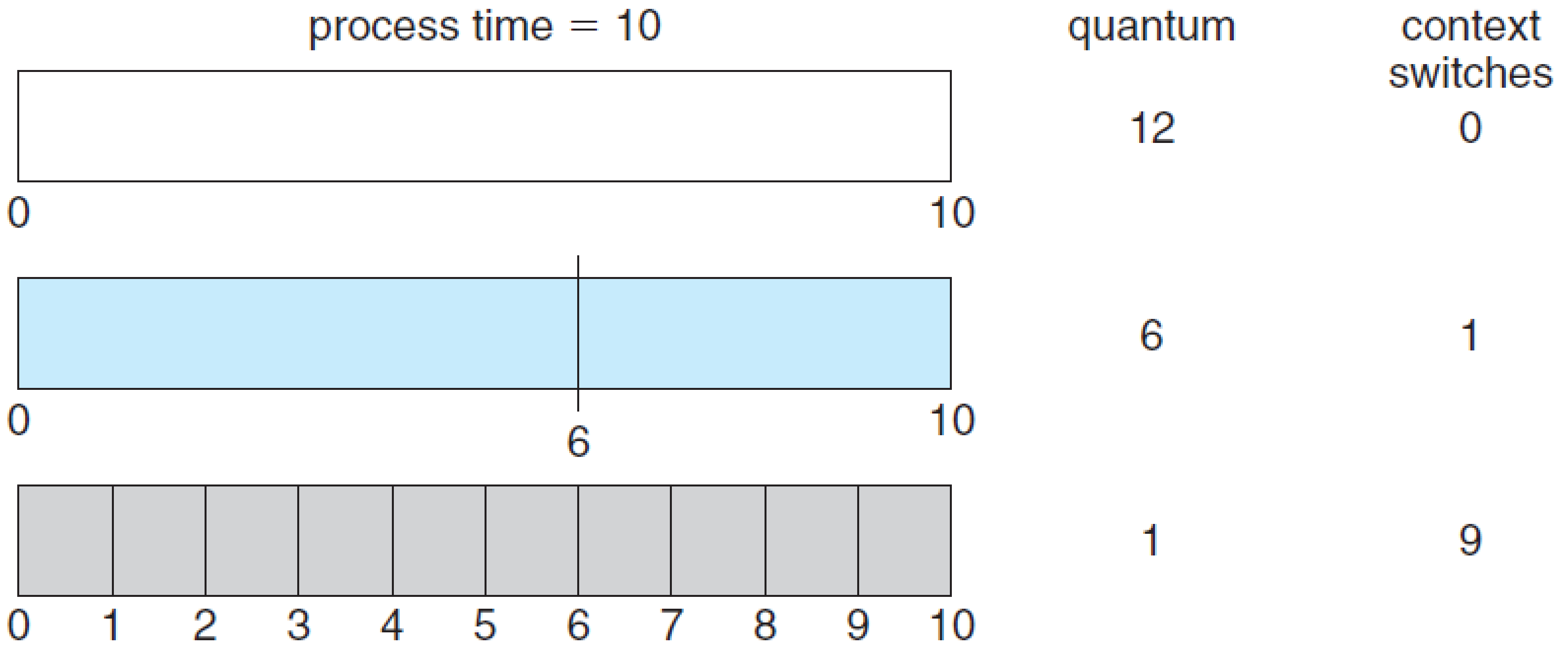
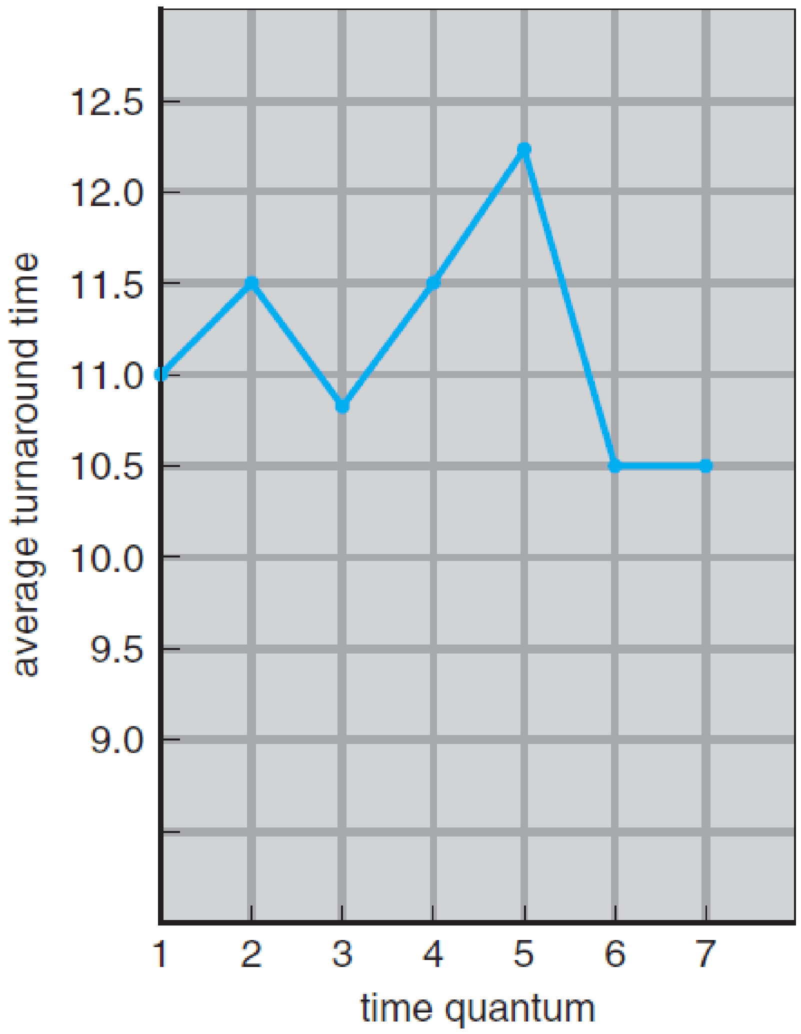


Figure 5.5 *How a smaller time quantum increases context switches.*



process	time
P_1	6
P_2	3
P_3	1
P_4	7

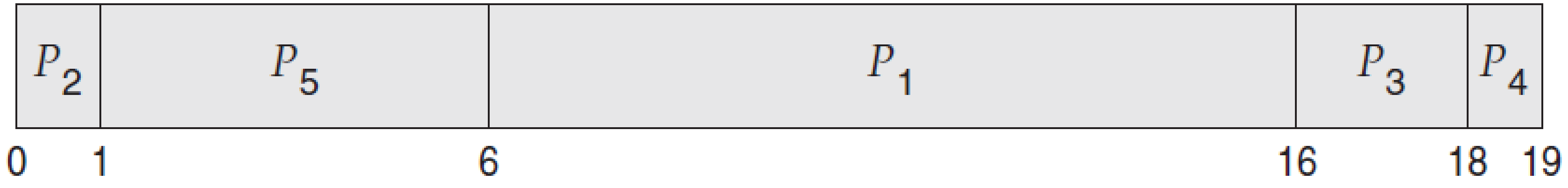
Figure 5.6 *How turnaround time varies with the time quantum.*

5.3 Scheduling Algorithms

- **Priority-base** Scheduling
  - A *priority* is associated with each process,
    - and the CPU is allocated to the process with the *highest priority*.
    - Processes with *equal* priority are scheduled in FCFS order.
  - Note that the *SJF* is a *special case* of the *priority-based* scheduling.
    - in this case, the priority is the *inverse* of the *next CPU burst*.
  - We assume that *low* numbers represent *high* priority.

5.3 Scheduling Algorithms

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



- the average waiting time: 8.2
- the average turnaround time: ?

5.3 Scheduling Algorithms

- Priority scheduling can be
  - either *preemptive* or *non-preemptive*.

- The problem of ***starvation*** (*indefinite blocking*)
  - a *blocked process*: ready to run, but waiting for the CPU.
  - some *low-priority* processes may *wait indefinitely*.

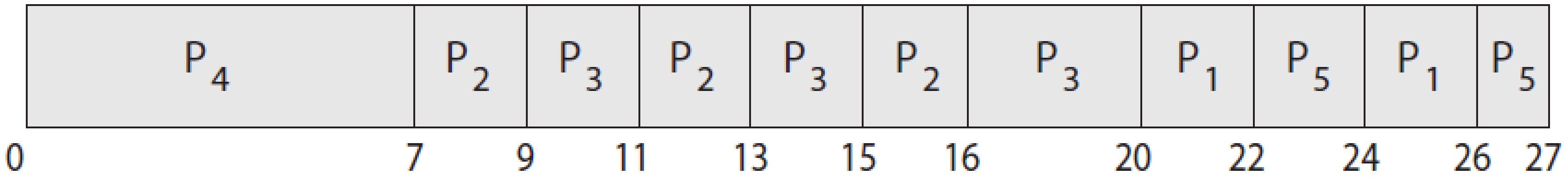
- A solution to the starvation problem is ***aging***
  - gradually increase the priority of processes
    - that wait in the system for a long time.

5.3 Scheduling Algorithms

- Combine RR and Priority scheduling:
  - execute the *highest-priority* process and
  - runs processes with the *same* priority using *round-robin* scheduling.

Process	Burst Time	Priority
P_1	4	3
P_2	5	2
P_3	8	2
P_4	7	1
P_5	3	3

- time quantum = 2



- Multi-Level Queue(MLQ) Scheduling

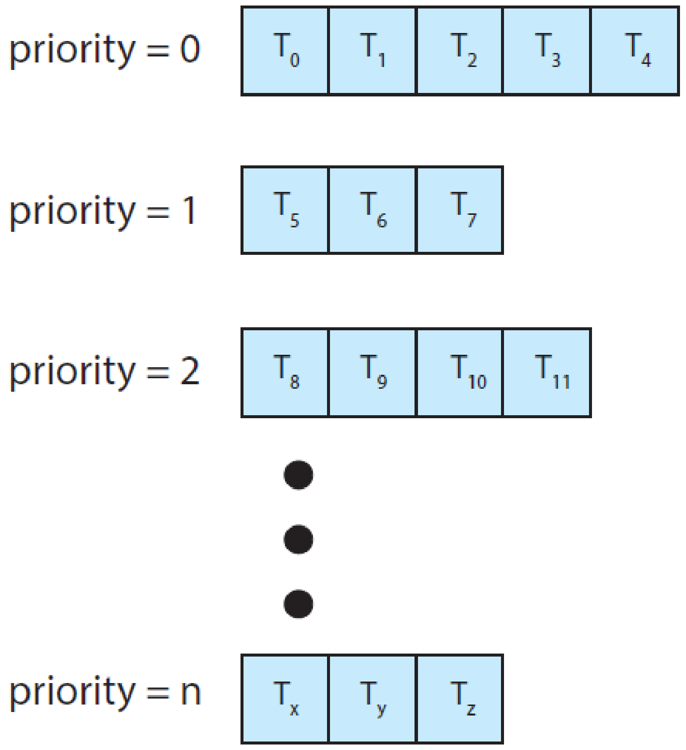


Figure 5.7 *Separate queues for each priority.*

5.3 Scheduling Algorithms

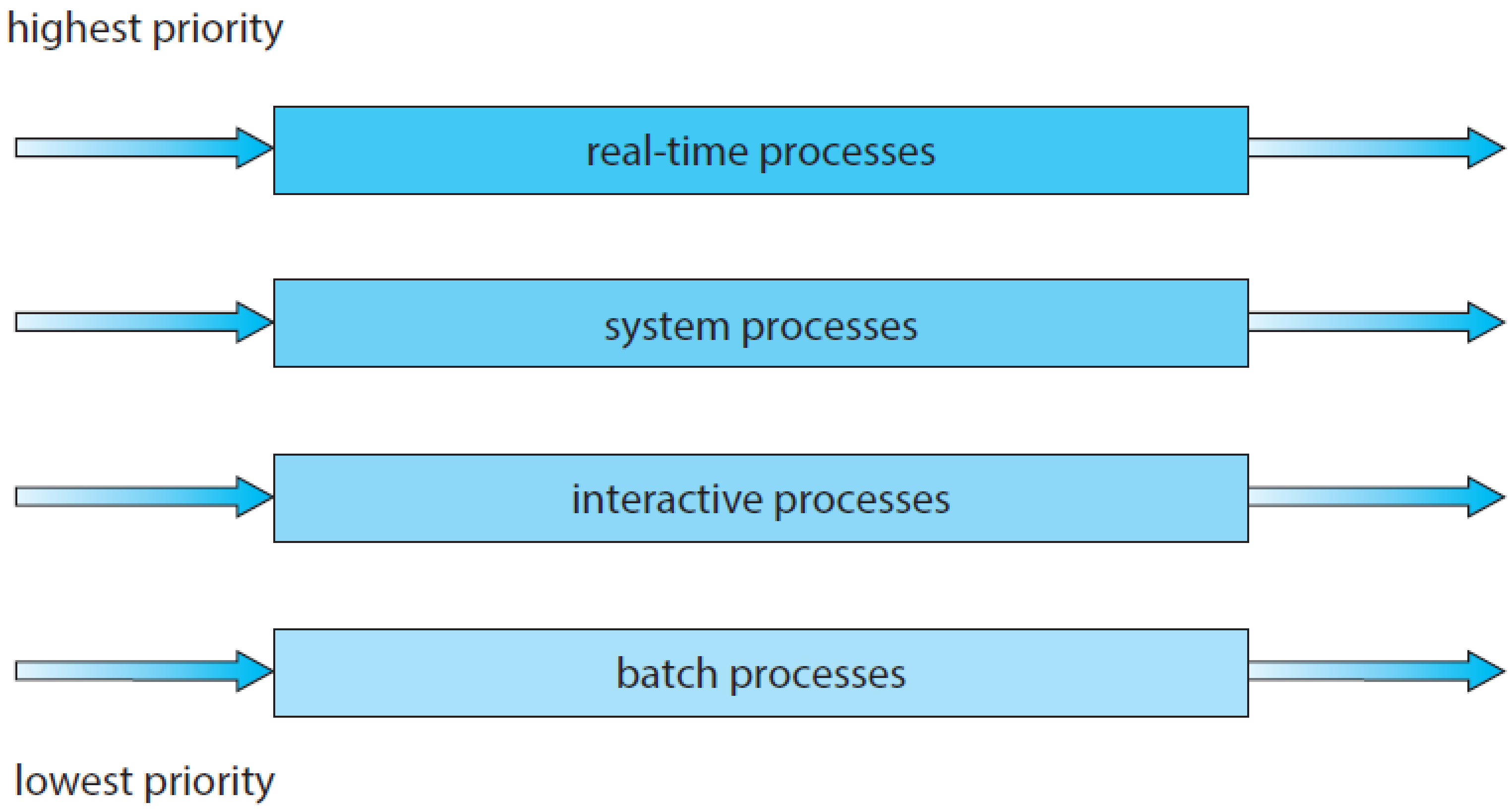


Figure 5.8 *Multi-Level Queue scheduling.*

5.3 Scheduling Algorithms

- Multi-Level Feedback Queue(MLFQ) Scheduling

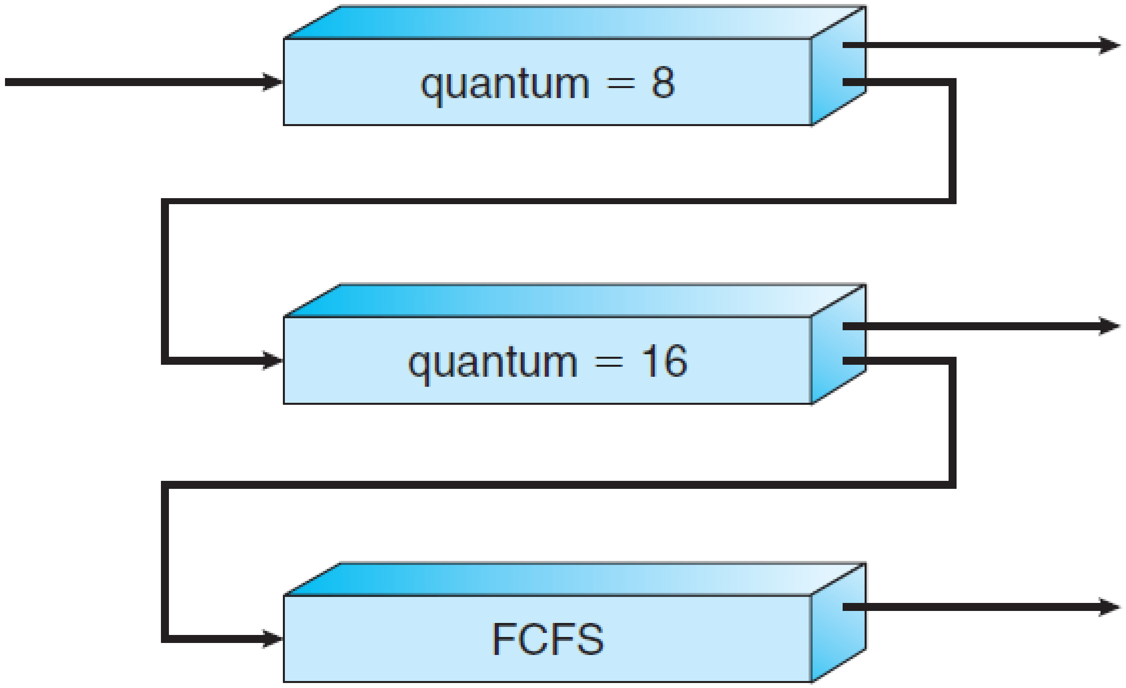


Figure 5.9 *Multi-Level Feedback Queue scheduling.*

5.4 Thread Scheduling

- On most modern operating systems
  - it is ***kernel threads*** – not ***processes*** – that are being scheduled,
  - and *user threads* are managed by a thread library.
    - So, the kernel is unaware of them,
    - ultimately mapped to associated kernel threads.

5.6 Real-Time CPU Scheduling

- Scheduling in the Real-Time Operating System.
  - ***Soft Realtime*** .vs. ***Hard Realtime***
  - Soft real-time systems provide no guarantee
    - as to when a critical real-time process will be scheduled.
    - guarantee only that a critical process is preferred to noncritical one.
  - Hard real-time systems have stricter requirements.
    - A task must be services by its deadline.

Practice Exercises

- Exercise 5.3 (p.251)

Process	Arrival Time	Burst Time
P_1	0.0	8
P_2	0.4	4
P_3	1.0	1

a. What is the average turnaround time for these processes with the FCFS scheduling algorithm?

b. What is the average turnaround time for these processes with the SJF scheduling algorithm?

Practice Exercises

Process	Arrival Time	Burst Time
P_1	0.0	8
P_2	0.4	4
P_3	1.0	1

c. The SJF algorithm is supposed to improve performance, but notice that we chose to run process P_1 at time 0 because we did not know that two shorter processes would arrive soon. Compute what the average turnaround time will be if the CPU is left idle for the first 1 unit and then SJF scheduling is used. Remember that processes P_1 and P_2 are waiting during this idle time, so their waiting time may increase. This algorithm could be known as ***future-knowledge scheduling***.

Any Questions?