Process Scheduling

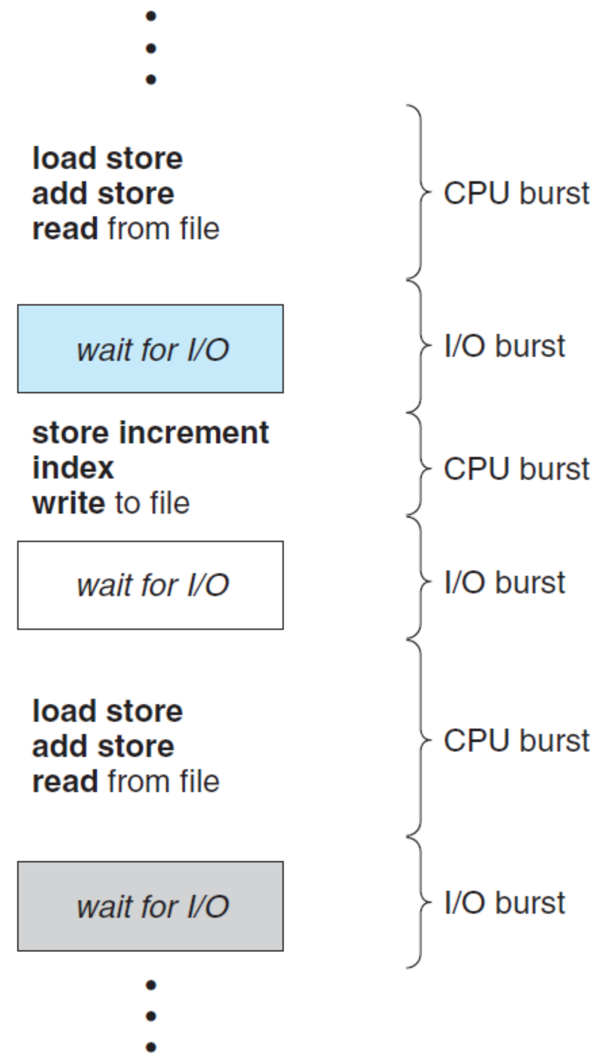
Process Scheduling

- CPU Scheduling: switching the CPU among processes
- Actual scheduling entity is the *thread*.
- The terms "CPU scheduling", "process scheduling", and "thread scheduling" are often used interchangeably.

- Why is process scheduling needed?
  - To maximize "___ __________"
    - process run → issue I/O → sit idle → CPU wasted
  - Any other reasons?

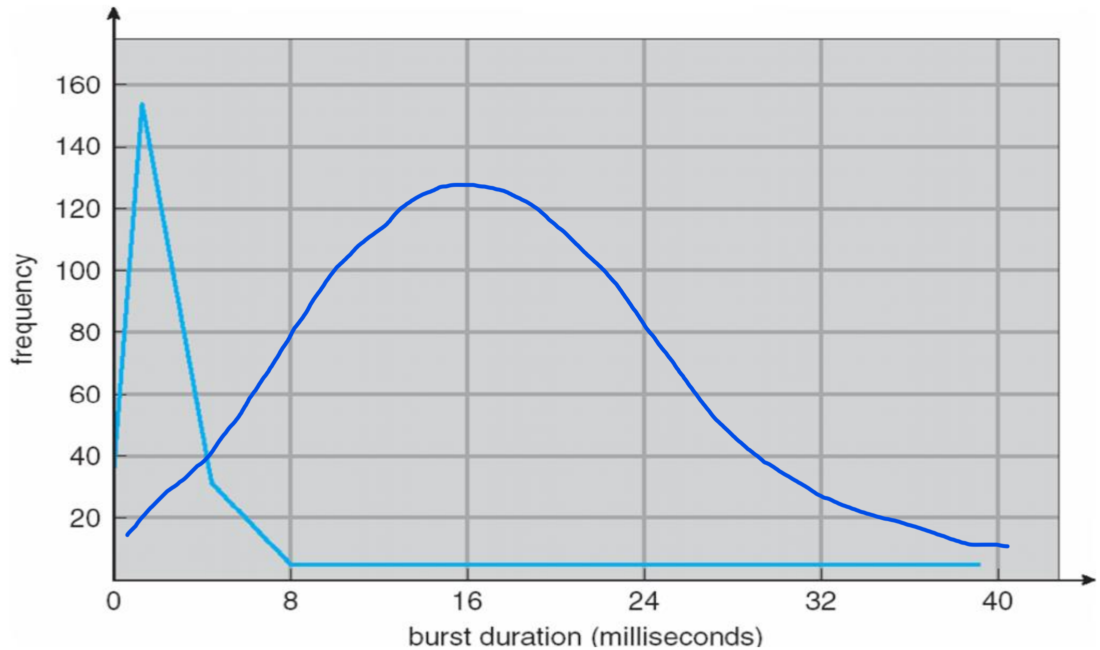
CPU-I/O Burst Cycle

- Process execution
  - Alternating cycle of CPU execution and I/O wait
    - CPU burst → I/O burst → CPU burst → I/O burst …
- I/O: Input/Output
  - Data transfer operation between CPU and devices
    - e.g. disk I/O, network I/O, mouse, keyboard …



CPU Burst duration

- Histogram of CPU burst durations



- Large number of short CPU bursts
- Small number of long CPU bursts
- I/O bound program has many short CPU bursts, why?

CPU Scheduler

- When CPU becomes idle, OS selects one process from the ready queue
  - done by the CPU scheduler, also called ‘short-term scheduler’

- When to make a scheduling decision
  1) Process changes from running state to waiting state
     - result of an I/O request or an invocation of wait()
  2) Process changes from running state to ready state
     - (timer) interrupt occurs
  3) Process changes from waiting state to ready state
     - at completion of I/O
  4) Process terminates

- 1,4: must make a scheduling decision

Preemptive vs. Nonpreemptive

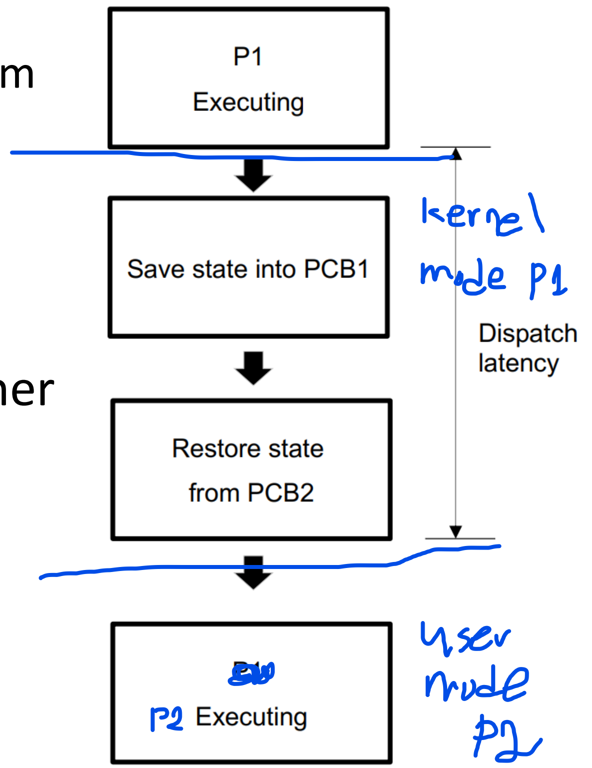
- Nonpreemptive schedulers
  - do scheduling only for 1 and 4
- Otherwise, preemptive
- In other words,
  - Can OS force the rescheduling of an actively running process?
    - yes: preemptive
    - no: nonpreemptive
- Under nonpreemptive scheduling
  - Process continues to run until it voluntarily releases CPU or terminates.

Preemptive scheduling

- Issues
  - Potential cause of race condition
    - Preemption happens while one process is updating a shared data
    - Another process reads inconsistent data
  - Complication to the kernel design
    - Preemption during system call handling
      - Kernel data structure may fall into inconsistent state
    - One option
      - Disallow preemption while handling system calls
      - Bad for real-time computing

Dispatcher

- A component that gives control of CPU to the selected process
  - Responsibilities
    - Switching context
    - Switching to user mode
    - Jumping to the proper location in the user program to restart(continue) that program



- Dispatch latency
  - Time it takes to stop one process and start another

Scheduling Criteria

- CPU utilization
  - We want to keep the CPU busy 100% of the time with useful work
  - In reality, it is 30-40%
- Throughput
  - Maximize the number of jobs processed per ~~hour~~
- Turnaround time
  - from the time of submission to completion
- Waiting time
  - Sum of times spent waiting in the ready queue
- Response time
  - from the time of submission till the first response is produced (mainly for interactive jobs)
- Fairness
  - Make sure each process gets fair share of the CPU

Using the Scheduling Criteria

- Maximize
  - CPU utilization, throughput, fairness
- Minimize
  - Turnaround time, waiting time, response time
- Optimize for:
  - average
  - minimum or maximum
- For interactive systems
  - It is better to minimize the variance in the response time than the average

FCFS Scheduling

- First-come, first-served scheduling algorithm
  - Serves the jobs in the order they arrive
  - Nonpreemptive
  - Simple and easy to implement
    - Process enters the ready queue → PCB linked to the tail of ready queue
    - Process at the head is allocated the CPU
  - Downside: average waiting time could be high

Process	Burst Time
P_1	24 ms
P_2	3 ms
P_3	3 ms

- Suppose processes arrive in the order of P_1, P_2, P_3. The Gantt chart for the schedule is:

- waiting time for P1=0, P2=24, P3=27
- Average waiting time

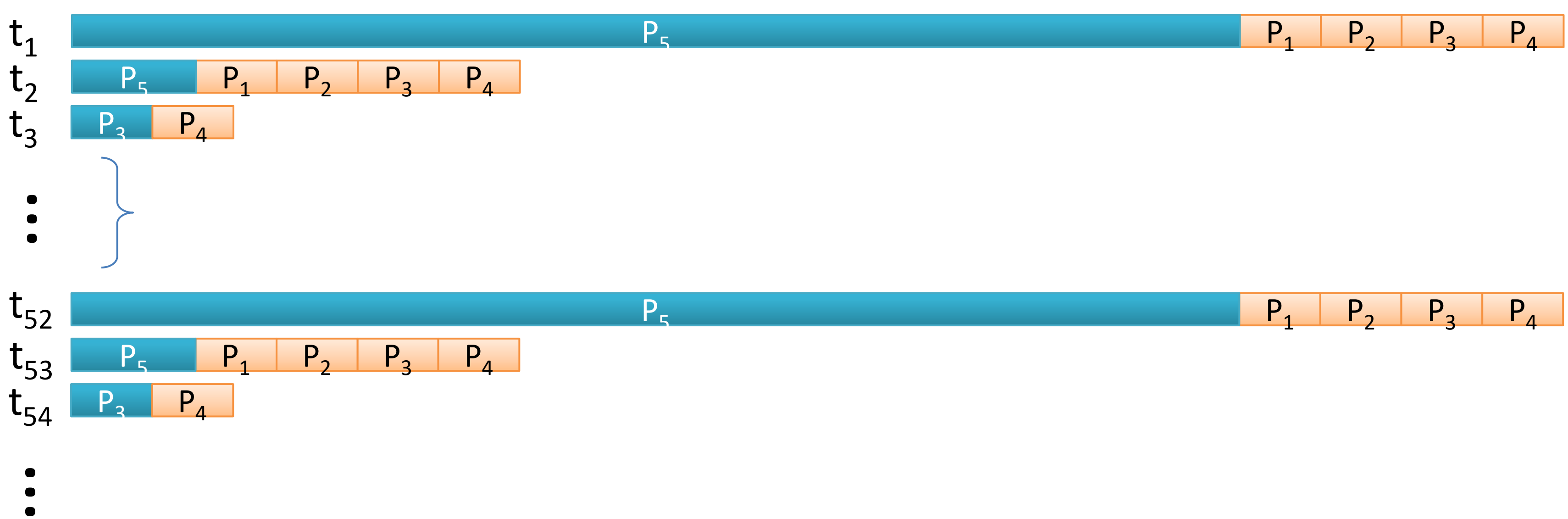
  (0+24+27)/3 = 17 ms

FCFS Scheduling: Reducing The Waiting Time

- Suppose processes arrive in the order of P_2, P_3, P_1. The Gantt chart for the schedule is:

- Average waiting time: (0+3+6)/3 = **3 ms !!**

- Under FCFS, average waiting time has room for improvement.
- Convoy effect can happened under FCFS

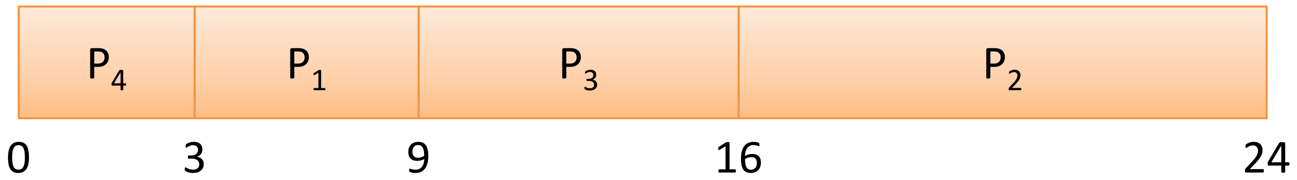


SJF: Shortest-Job-First Scheduling

- Key idea
  - Associate with each process the length of next CPU burst
  - Schedule the process with the smallest CPU burst

Process	Burst Time
P_1	6 ms
P_2	8 ms
P_3	7 ms
P_4	3 ms

Scheduling by SJF

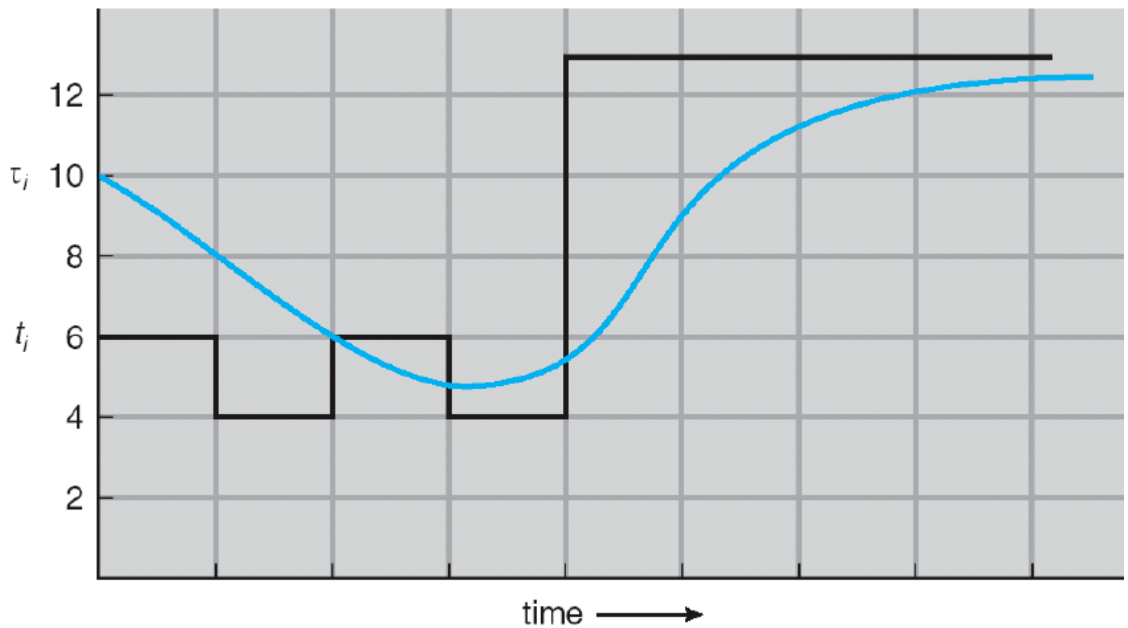


- Average waiting time = (3+16+9+0)/4=7
  - AWT for FCFS is?
- SJF gives optimal minimum average waiting time
- Difficult to know the CPU burst length
- In long-term scheduling:
  - Process time limit is specified by user
  - SJF is used mostly in the long-term scheduling scenario

SJF: Approximating CPU burst length

- In short-term scheduling:
  - too difficult to know the length of CPU burst → approximation is used
  - Idea: use the history to estimate the future
- Prediction with exponential average

Why exponential?



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

SJF: Exponential Average

- t_n: length of the *n*th CPU bust
- τ_{n+1}: predicted value for the next CPU burst
- Define τ_{n+1} as:

$$\tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n \quad \text{where } 0 \le \alpha \le 1$$

  - α controls the weight of recent and past history
    - if $\alpha=1$, what does it mean?
  - Previous figure: $\alpha=1/2$, $\tau_0=10$
- If we expand the formula:

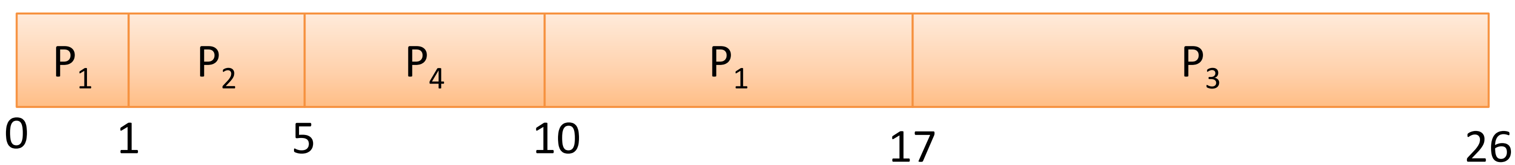
$$\tau_{n+1} = \alpha t_n + (1-\alpha)\alpha t_{n-1} + \dots + (1-\alpha)^j \alpha t_{n-j} + \dots + (1-\alpha)^{n+1}\tau_0$$

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

SJF: Preemptive or Nonpreemptive?

- SJF can be either preemptive or nonpreemptive
  - When? A new process arrives at the ready queue, and it could be shorter than the current process
  - If it is preemptive, current one will be descheduled
  - Shortest-remaining-time-first scheduling

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



- Average waiting time = [(10-1)+(1-1)+(17-2)+(5-3)]/4 = 6.5 ms
- What about the nonpreemptive case?

Priority Scheduling

- Priority scheduling
  - A priority is associated with each process and the scheduler selects the process with the highest priority
- Priority
  - some fixed range of numbers (e.g. 0 – 100)
  - let's use low number for high priority
- SJF is a *special case* of the priority scheduling
  - Priority is the inverse of the next CPU burst
    - The larger the CPU burst, the lower the priority
- FCFS: a priority scheduling with equal priority
  - Is FCFS also a special case of priority scheduling?

Priority Scheduling

- Assume these 5 processes arrived at time 0

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

P_2	P_5	P_1	P_3	P_4

0 1 6 16 18 19

- average waiting time: 8.2 ms

Priority Scheduling

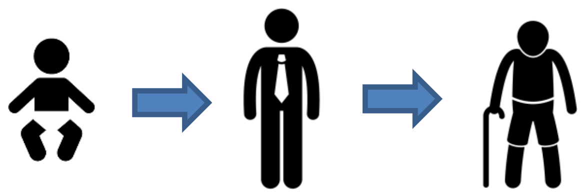
- Two types of priority
  - Internally defined
    - use measurable quantities to compute the priority
    - E.g., time limits, memory requirements, # of open files, ratio of average I/O burst to average CPU burst … etc
  - Externally defined
    - set by criteria outside the operating system
    - E.g., importance of process, type and amount of funds paid for computers, the department sponsoring the work
- Preemptive or nonpreemptive priority scheduling
  - When process arrives at the ready queue, priority is checked
    - if preemptive: switch the current process if priority is higher
    - if nonpreemptive: put the high priority process to the head of the ready queue

Starvation Problem

- Problem with Priority Scheduling

  - Starvation (indefinite blocking)
    - Low priority processes may never get a chance to be scheduled
      - There is no guarantee how long it should wait

- Solution



  - Aging
    - Gradually increase the priority of processes
      - For example, increase the priority of processes that stayed 15 minutes in the ready queue
      - Eventually it will become a high-priority process and be scheduled
        - » There is a bound to the wait time

Round-Robin Scheduling

- Characteristics of Round-robin
  - Designed for time-sharing system
  - Similar to FCFS, but preemption is added
  - The notion of time quantum (slice)
  - Treat ready queue as FIFO, new process added to the tail

- Algorithm
  - Always pick the first process in the ready queue
  - Run for a time quantum
  - If not finished, put it to the tail and pick the next one (the first one in the ready queue)

Round-Robin Scheduling

- Example
  - Assume the time quantum of 4 ms

Process	Burst Time
P_1	24
P_2	3
P_3	3

P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1

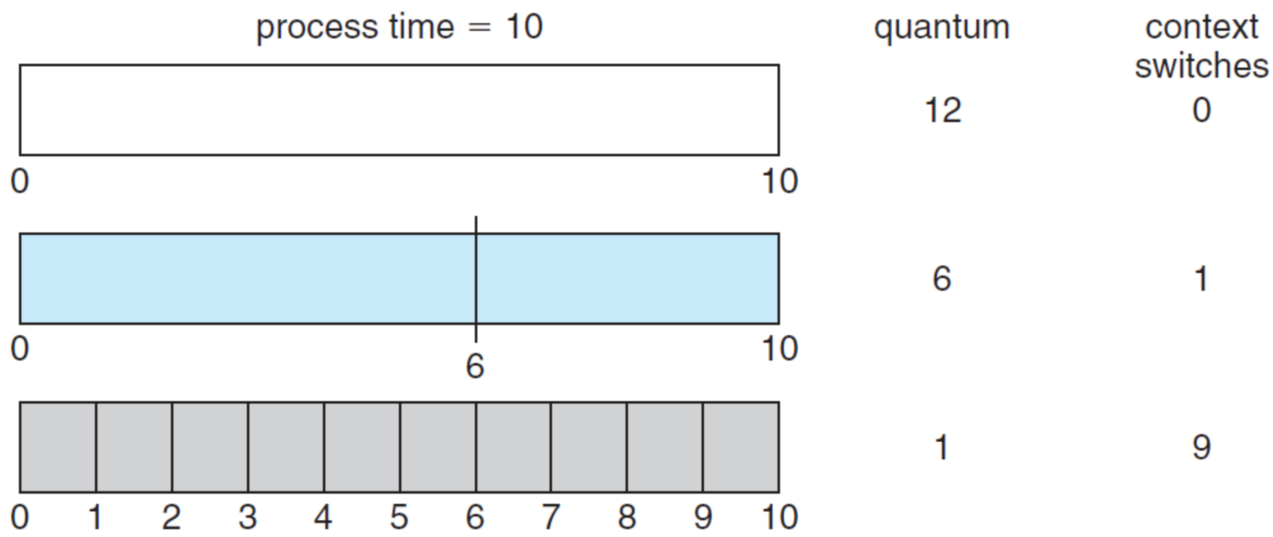
0 4 7 10 14 18 22 26 30

  - Average waiting time

$$(6 + 4 + 7)/3 = 5.66$$

Round-Robin Scheduling

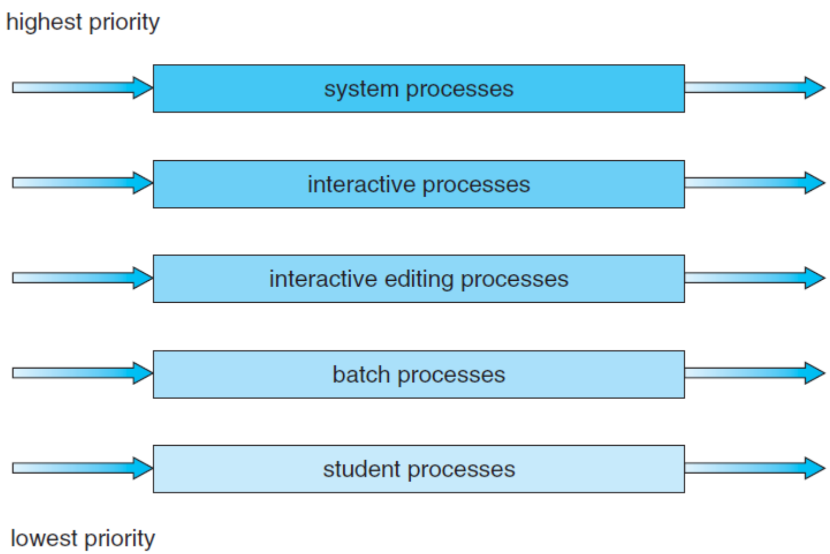
- Effect of time quantum to the performance



- If time quantum is very large (infinite), it reduces to FCFS
- As time quantum gets smaller, the context switch overhead increases
- Ideally the time quantum should be significantly larger than the context switch time
- In practice, time quantum = 10ms, context switch=10us

Multilevel Queue Scheduling

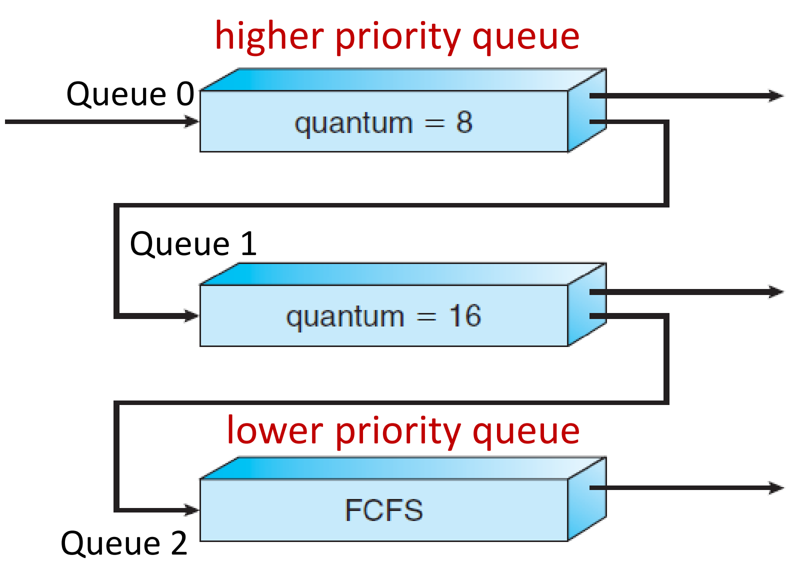
- Designed to handle scheduling of processes classified into different classes
  - Each class has different requirements
    - Foreground vs. background processes
    - System vs. interactive vs. batch processes
- Partition the ready queue into several separate queues



- Processes permanently assigned to one queue
- Each queue has its own scheduling algorithm
  - RR, FCFS … etc.
- Priority between queues
  - usually the absolute priority
  - No process in 'batch queue' can run unless all above queues are empty
- Time slicing among queues with different weights

Multilevel Feedback Queue Scheduling

- What is the problem with Multilevel Queue?
- MFQS (Multilevel Feedback Queue Scheduling)
  - Processes can move between queues
  - Intention: separate processes according to CPU burst
    - Long processes move down to the lower level
    - better for Interactive and I/O processes
    - Aging can be applied to prevent starvation



Multilevel Feedback Queues

- Process enters Queue 0
- Process in Q0 preempts Q1
- If process in Q0 does not finish in 8 ms, it is put into the tail of Q1
- If process in Q1 does not finish in 16 ms, it is put into the tail of Q2
- Process in Q1 will execute only when Q0 is empty

Most general form of CPU scheduler