Process/CPU scheduling (contd.)

Indranil Sen Gupta (odd section)
and Mainack Mondal (even section)
CS39002

Spring 2019-20

The key concepts so far

- CPU burst, I/O burst
- CPU scheduler (which process should execute next)

The key concepts so far

- CPU burst, I/O burst
- CPU scheduler (which process should execute next)

- Non preemptive scheduling (a process runs uninterrupted)
- Pre-emptive scheduling (CPU forcibly taken from running process)

The key concepts so far

- CPU burst, I/O burst
- CPU scheduler (which process should execute next)

- Non preemptive scheduling (a process runs uninterrupted)
- Pre-emptive scheduling (CPU forcibly taken from running process)

- Dispatcher (gives control of CPU to scheduled process)

Scheduling criteria

- CPU utilization – keep the CPU as busy as possible
- Throughput – # of processes that complete their execution per time unit
- 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)
- Burst time – amount of time a process is executed

Scheduling algorithm optimization criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

CPU scheduling algorithms

Today’s class

- Algo 1: First come first serve (FCFS)
- Algo 2: Shortest job first (SJF)
- Algo 3: Priority scheduling
- Algo 4: Round robin scheduling
- Algo 5: Multi level queue scheduling
- Algo 6: Multi level feedback queue scheduling

Today's class

- Algo 1: First come first serve (FCFS)
- Algo 2: Shortest job first (SJF)
- Algo 3: Priority scheduling
- Algo 4: Round robin scheduling
- Algo 5: Multi level queue scheduling
- Algo 6: Multi level feedback queue scheduling

Algo 1. First Come First Serve scheduling (FCFS)

- Non preemptive scheduling
  - Process that requests CPU first is allocated the CPU first
  - Ready list is maintained as a FIFO queue

Algo 1. First Come First Serve scheduling (FCFS)

- Non preemptive scheduling
  - Process that requests CPU first is allocated the CPU first
  - Ready list is maintained as a FIFO queue
  - Issue: Average waiting time is long

Example 1

Process	P1	P2	P3
Arrival time	0	0	0
CPU burst	24ms	3ms	3ms

Algo 1. First Come First Serve scheduling (FCFS)

- Non preemptive scheduling
  - Process that requests CPU first is allocated the CPU first
  - Ready list is maintained as a FIFO queue
  - Issue: Average waiting time is long

Example 1

Process	P1	P2	P3
Arrival time	0	0	0
CPU burst	24ms	3ms	3ms

Draw Gantt chart and calculate average waiting time for two schedules: P1, P2, P3 and P2, P3, P1

Algo 1. First Come First Serve scheduling (FCFS)

- Non preemptive scheduling
  - Process that requests CPU first is allocated the CPU first
  - Ready list is maintained as a FIFO queue
  - Issue: Average waiting time is long

Example 1

Process	P1	P2	P3
Arrival time	0	0	0
CPU burst	24ms	3ms	3ms

Draw Gantt chart and calculate average waiting time for two schedules: P1, P2, P3 and P2, P3, P1 (Ans: 17 ms and 3 ms)

Yet another example

Example 2

Process	P1	P2	P3	P4	P5
Arrival time	0	2ms	3ms	5ms	9ms
CPU burst	3ms	3ms	2ms	5ms	3ms

Draw Gantt chart and calculate average waiting time

Yet another example

Example 2

Process	P1	P2	P3	P4	P5
Arrival time	0	2ms	3ms	5ms	9ms
CPU burst	3ms	3ms	2ms	5ms	3ms

Draw Gantt chart and calculate average waiting time

(Ans: 11/3 ms)

Problems with FCFS

- Convoy effect
  - A process with large CPU burst delays several process with shorter CPU bursts

Problems with FCFS

- Convoy effect
  - A process with large CPU burst delays several process with shorter CPU bursts

- Prefers CPU bound processes
  - Since burst times of I/O bound processes are small
  - Lower device (e.g., I/O) utilization

Today's class

- Algo 1: First come first serve (FCFS)
- Algo 2: Shortest job first (SJF)
- Algo 3: Priority scheduling
- Algo 4: Round robin scheduling
- Algo 5: Multi level queue scheduling
- Algo 6: Multi level feedback queue scheduling

Algo 2: Shortest Job First (SJF)

- Still non pre-emptive
- Idea: Execute the shortest processes first
  - Challenge: How to know which one is “shortest”?

Algo 2: Shortest Job First (SJF)

- Still non pre-emptive
- Idea: Execute the shortest processes first
    - Challenge: How to know which one is “shortest”?

- Associate with each process an estimate of the length of the next CPU burst for the process
    - When CPU is available, assign CPU to the process with smallest estimate

SJF: example

Process	P1	P2	P3	P4
Arrival time	0	0	0	0
CPU burst	6ms	8ms	7ms	3ms

What is the SJF schedule and corresponding wait time?

Compare with the following FCFS schedule: P1, P2, P3, P4

SJF: example

Process	P1	P2	P3	P4
Arrival time	0	0	0	0
CPU burst	6ms	8ms	7ms	3ms

What is the SJF schedule and corresponding wait time?

Compare with the following FCFS schedule: P1, P2, P3, P4

(Ans: SJF – 7 ms and FCFS – 10.25 ms)

SJF: guarantee

- Optimality: The SJF algorithm minimizes the average waiting time

- Prove it for a set of n processes which arrive at the same time with CPU burst times $t_1 \leq t_2 \leq t_3 \leq t_4 \ldots \leq t_n$, ignoring further arrivals.

SJF: guarantee

- Optimality: The SJF algorithm minimizes the average waiting time

- Prove it for a set of n processes which arrive at the same time with CPU burst times $t1 \leq t2 \leq t3 \leq t4 \ldots \leq tn$, ignoring further arrivals.
- Hint: Contradiction

SJF: Key issue

- How to estimate the next CPU burst time?
  - A common approach is to use exponential average of the measured length of previous CPU bursts

SJF: Key issue

- How to estimate the next CPU burst time?
  - A common approach is to use exponential average of the measured length of previous CPU bursts

$$Let\ t_n = Length\ of\ n^{th}\ CPU\ burst$$

$$\tau_{n+1} = predicted\ value\ of\ the\ next\ CPU\ burst$$

$$Then, \tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n, 0 \leq \alpha \leq 1$$

SJF: Key issue

- How to estimate the next CPU burst time?
  - A common approach is to use exponential average of the measured length of previous CPU bursts

$$Let\ t_n = Length\ of\ n^{th}\ CPU\ burst$$

$$\tau_{n+1} = predicted\ value\ of\ the\ next\ CPU\ burst$$

$$Then, \tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n, 0 \le \alpha \le 1$$

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

SJF: Key issue

- How to estimate the next CPU burst time?
  - A common approach is to use exponential average of the measured length of previous CPU bursts

$$Let\ t_n = Length\ of\ n^{th}\ CPU\ burst$$

$$\tau_{n+1} = predicted\ value\ of\ the\ next\ CPU\ burst$$

$$Then, \tau_{n+1} = \alpha t_n + (1-\alpha)\tau_n, 0 \leq \alpha \leq 1$$

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

$$\alpha = 0 \rightarrow \tau_{n+1} = \tau_n \rightarrow recent\ history\ has\ no\ effect$$

$$\alpha = 1 \rightarrow \tau_{n+1} = t_n \rightarrow Only\ the\ most\ recent\ CPU\ burst\ has\ effect$$

Shortest remaining time first scheduling

- Pre-emptive version of SJF
    - A smaller CPU burst time process can evict a running process

Shortest remaining time first scheduling

- Pre-emptive version of SJF
  - A smaller CPU burst time process can evict a running process

Process	P1	P2	P3	P4
Arrival time	0	1ms	2ms	3ms
CPU burst	8ms	4ms	9ms	5ms

- Draw preemptive gantt chart and computing waiting time.

Shortest remaining time first scheduling

- Pre-emptive version of SJF
  - A smaller CPU burst time process can evict a running process

Process	P1	P2	P3	P4
Arrival time	0	1ms	2ms	3ms
CPU burst	8ms	4ms	9ms	5ms

- Draw preemptive gantt chart and computing waiting time.

(Ans: 6.5 ms)

Today's class

- Algo 1: First come first serve (FCFS)
- Algo 2: Shortest job first (SJF)
- Algo 3: Priority scheduling
- Algo 4: Round robin scheduling
- Algo 5: Multi level queue scheduling
- Algo 6: Multi level feedback queue scheduling

Algo 3. Priority scheduling

- A priority is assigned to each process
  - CPU is allotted to the process with highest priority
  - SJF is a type of priority scheduling

Algo 3. Priority scheduling

- A priority is assigned to each process
  - CPU is allotted to the process with highest priority
  - SJF is a type of priority scheduling

Process	P1	P2	P3	P4	P5
Arrival time	0	0	0	0	0
CPU burst	10ms	1ms	2ms	1ms	5ms
Priority	3	1	4	5	2

What is the average waiting time?

Algo 3. Priority scheduling

- A priority is assigned to each process
  - CPU is allotted to the process with highest priority
  - SJF is a type of priority scheduling

Process	P1	P2	P3	P4	P5
Arrival time	0	0	0	0	0
CPU burst	10ms	1ms	2ms	1ms	5ms
Priority	3	1	4	5	2

What is the average waiting time?

(Ans: 8.2 ms)

Assigning priority: static approach

- Each process has a static priority
  - Large change of indefinite blocking
  - Can lead to starvation

Assigning priority: dynamic approach

- Compute highest response time (RN)

$$RN = \frac{Time\ since\ arrival + CPU\ burst\ time}{CPU\ burst\ time}$$

Assigning priority: dynamic approach

- Compute highest response time (RN)

$$RN = \frac{Time\ since\ arrival + CPU\ burst\ time}{CPU\ burst\ time}$$

- For a waiting process
    - “Time since arrival increase” -> RN increase

Assigning priority: dynamic approach

- Compute highest response time (RN)

$$RN = \frac{Time\ since\ arrival + CPU\ burst\ time}{CPU\ burst\ time}$$

- For a waiting process
  - “Time since arrival increase” -> RN increase
- For a short process
  - “CPU burst time decrease” -> RN increase

Assigning priority in Linux

- Priority of a process is determined by nice value
    - Nice value range from -20 to 19
    - -20 is highest priority and 19 is lowest priority
    - Default nice value is 0

Assigning priority in Linux

- Priority of a process is determined by nice value
  - Nice value range from -20 to 19
  - -20 is highest priority and 19 is lowest priority
  - Default nice value is 0

- “nice” and “renice” used for set/change nice value
  - A user can only decrease priority
  - superuser can increase peiority

Today's class

- Algo 1: First come first serve (FCFS)
- Algo 2: Shortest job first (SJF)
- Algo 3: Priority scheduling
- Algo 4: Round robin scheduling
- Algo 5: Multi level queue scheduling
- Algo 6: Multi level feedback queue scheduling

Algo 4. Round robin (RR) scheduling

- Designed for time-sharing systems
    - A small unit of time, time quantum or time slice is defined
    - Typically 10-100 ms
    - READY queue is a circular queue in this case
    - The CPU goes around each process in READY queue and execute for 1 time slice
    - A timer is set to interrupt the CPU at the end of each time slice

RR scheduling: more details

- Once a process gets the CPU two things might happen
  - The process has CPU burst ≤ 1 time slice, so the process release CPU voluntarily

RR scheduling: more details

- Once a process gets the CPU two things might happen
  - The process has CPU burst $\leq$ 1 time slice, so the process release CPU voluntarily
  - If CPU burst is > 1 time slice then timer interrupt, context switch, next process is loaded from READY queue

RR scheduling: more details

- Once a process gets the CPU two things might happen
  - The process has CPU burst ≤ 1 time slice, so the process release CPU voluntarily
  - If CPU burst is > 1 time slice then timer interrupt, context switch, next process is loaded from READY queue

Example:

Process	P1	P2	P3
Arrival time	0	0	0
CPU burst	24ms	3ms	3ms

If time quantum $\delta = 4$ ms, then what is the avg. wait time? (schedule P1, P2, P3,…)

RR scheduling: more details

- Once a process gets the CPU two things might happen
  - The process has CPU burst ≤ 1 time slice, so the process release CPU voluntarily
  - If CPU burst is > 1 time slice then timer interrupt, context switch, next process is loaded from READY queue

Example:

Process	P1	P2	P3
Arrival time	0	0	0
CPU burst	24ms	3ms	3ms

If time quantum δ = 4 ms, then what is the avg. wait time? (schedule P1, P2, P3,…)

(Ans: 5.66ms)

RR scheduling: Analysis

- n process in READY queue, time slice δ
    - Each process gets 1/n CPU time, each lasts for δ time or less
    - Max. wait time for each process = $(n - 1)(\delta + \sigma)$
    - σ = scheduling overhead

RR scheduling: Analysis

- n process in READY queue, time slice δ
  - Each process gets 1/n CPU time, each lasts for δ time or less
  - Max. wait time for each process = (n - 1) ($\delta + \sigma$)
  - σ = scheduling overhead

- Very large δ = FCFS (why?)
- Very small δ = Large number of context switch (why?)

RR scheduling: Analysis

- n process in READY queue, time slice δ
  - Each process gets 1/n CPU time, each lasts for δ time or less
  - Max. wait time for each process = (n - 1) ($\delta + \sigma$)
  - σ = scheduling overhead
- Very large δ = FCFS (why?)
- Very small δ = Large number of context switch (why?)
- Typically $\delta \ggg \sigma$ (e.g., δ = 10 ms, σ = 10 μs)

Exercise

Process	P1	P2	P3	P4
Arrival time	0	0	0	0
CPU burst	6ms	3ms	1ms	7ms

Compute average turnaround time for δ = 1,2,3,4,5,6,7ms

Compute average wait time for δ = 1,2,3,4,5,6,7ms

Assume the schedule is P1, P2, P3, P4

Today's class

- Algo 1: First come first serve (FCFS)
- Algo 2: Shortest job first (SJF)
- Algo 3: Priority scheduling
- Algo 4: Round robin scheduling
- Algo 5: Multi level queue scheduling
- Algo 6: Multi level feedback queue scheduling

Algo 5. Multi level queue scheduling

- Ready queue is partitioned into separate queues, eg:
  - foreground (interactive)
  - background (batch)
- Process permanently in a given queue
- Each queue has its own scheduling algorithm

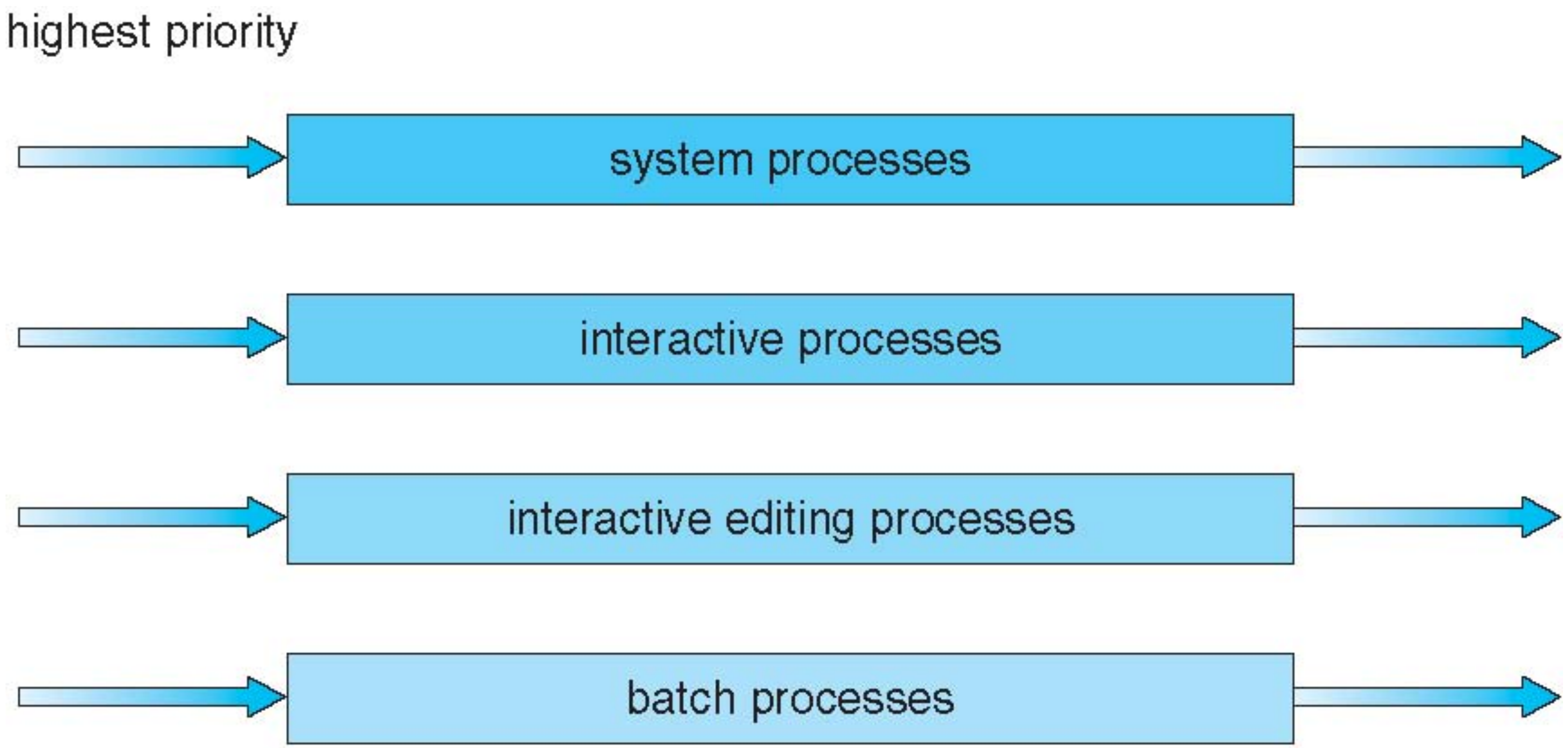
Algo 5. Multi level queue scheduling

- Ready queue is partitioned into separate queues, eg:
  - foreground (interactive)
  - background (batch)
- Process permanently in a given queue
- Each queue has its own scheduling algorithm
- Scheduling must be done between the queues:
  - Fixed priority scheduling: serve all from foreground then from background. Possibility of starvation.

Algo 5. Multi level queue scheduling

- Ready queue is partitioned into separate queues, eg:
  - foreground (interactive)
  - background (batch)
- Process permanently in a given queue
- Each queue has its own scheduling algorithm
- Scheduling must be done between the queues:
  - Fixed priority scheduling: serve all from foreground then from background. Possibility of 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

Multi level queues



Today's class

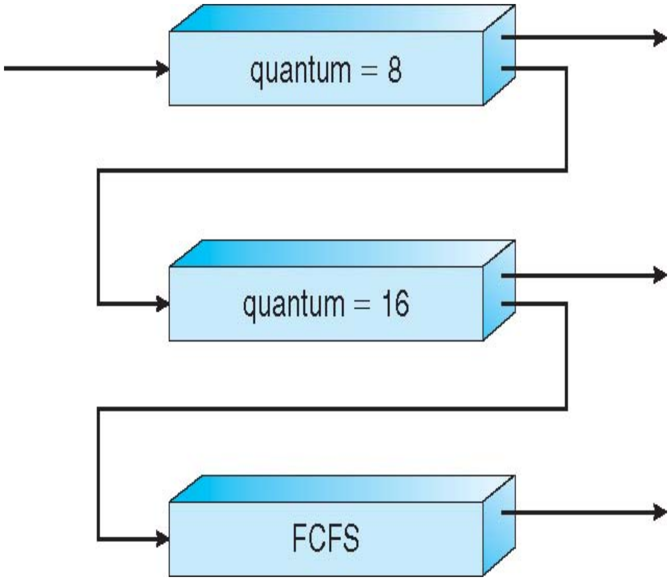
- Algo 1: First come first serve (FCFS)
- Algo 2: Shortest job first (SJF)
- Algo 3: Priority scheduling
- Algo 4: Round robin scheduling
- Algo 5: Multi level queue scheduling
- Algo 6: Multi level feedback queue scheduling

Algo 6. Multi level feedback queue scheduling

- We allow processes to move between queues
- I/O bound and interactive processes in high priority queue
  - A process waiting too long in lower priority queue will move to a higher priority queue
  - Avoids starvation

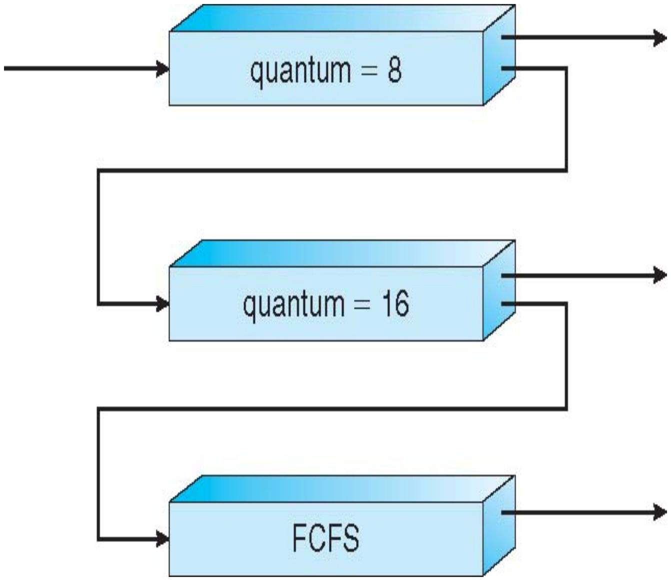
Multi level feedback queue: Example

- Three queues:
  - Q_0 – RR with time quantum (δ) 8 ms
  - Q_1 – RR with δ = 16ms
  - Q_2 – FCFS



Multi level feedback queue: Example

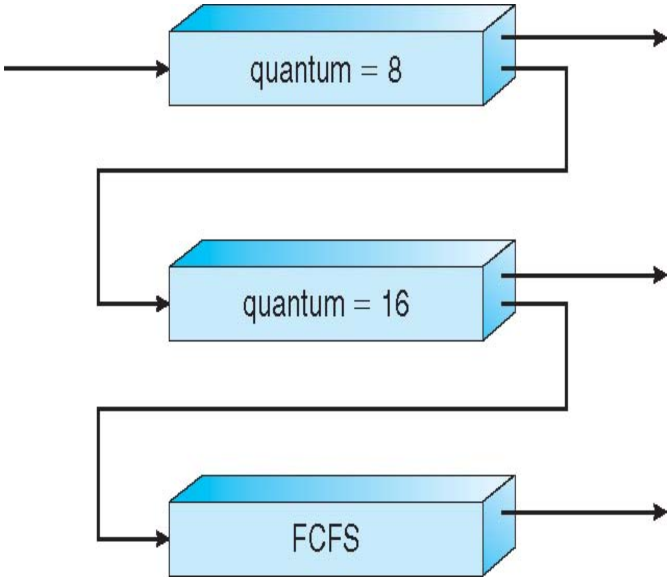
- Three queues:
  - Q_0 – RR with time quantum (δ) 8 ms
  - Q_1 – RR with $\delta = 16$ms
  - Q_2 – FCFS



- A process in Q1 can execute only when Q0 is empty

Multi level feedback queue: Example

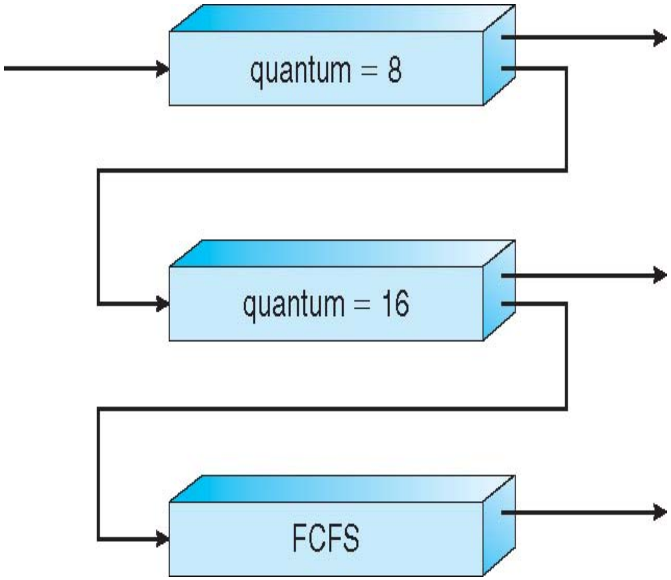
- Three queues:
  - Q_0 – RR with time quantum (δ) 8 ms
  - Q_1 – RR with $\delta = 16$ms
  - Q_2 – FCFS



- A process in Q1 can execute only when Q0 is empty
- A process in Q0 can pre-empt a process in Q1 or Q2

Multi level feedback queue: Example

- Three queues:
    - Q_0 – RR with time quantum (δ) 8 ms
    - Q_1 – RR with $\delta = 16$ms
    - Q_2 – FCFS



- A process in Q1 can execute only when Q0 is empty
- A process in Q0 can pre-empt a process in Q1 or Q2
- If the CPU burst of a process exceeds δ its moved to lower priority queue

Issue with Multi level feedback queue scheduling

- Long running processes may starve
  - Permanent demotion of priority hurts processes that change their behavior (e.g., lots of computation only at beginning)
  - Eventually all long-running processes move to FCFS

Issue with Multi level feedback queue scheduling

- Long running processes may starve
  - Permanent demotion of priority hurts processes that change their behavior (e.g., lots of computation only at beginning)
  - Eventually all long-running processes move to FCFS

- Solution
  - periodic priority boost: all processes moved to high priority queue
  - Priority boost with aging: recompute priority based on scheduling history of a process

Summary

- Algo 1: First come first serve (FCFS)
- Algo 2: Shortest job first (SJF)
- Algo 3: Priority scheduling
- Algo 4: Round robin scheduling
- Algo 5: Multi level queue scheduling
- Algo 6: Multi level feedback queue scheduling

Next class

- Multithreading