Process/CPU scheduling

Saptarshi Ghosh and Mainack Mondal

CS31202 / CS30002

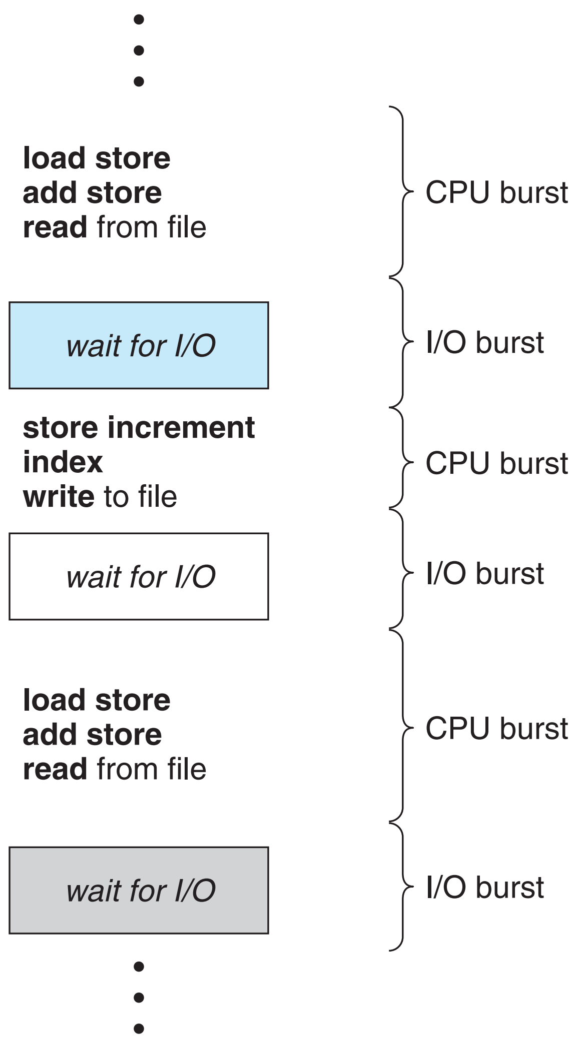
Why process scheduling: Recap

- CPU scheduling forms the central idea behind multiprogramming OS
  - By switching the CPU among processes, the OS better utilizes the computer system
  - Modern OS also consider lightweight process like entities called threads which need to be scheduled

Still: Why do we need scheduling today

- Recall
  - CPU burst: the process is being executed in the CPU
  - I/O burst: the process is waiting for I/O to be done
  - A Process alternates between CPU and I/O burst

Why is CPU and I/O burst important?



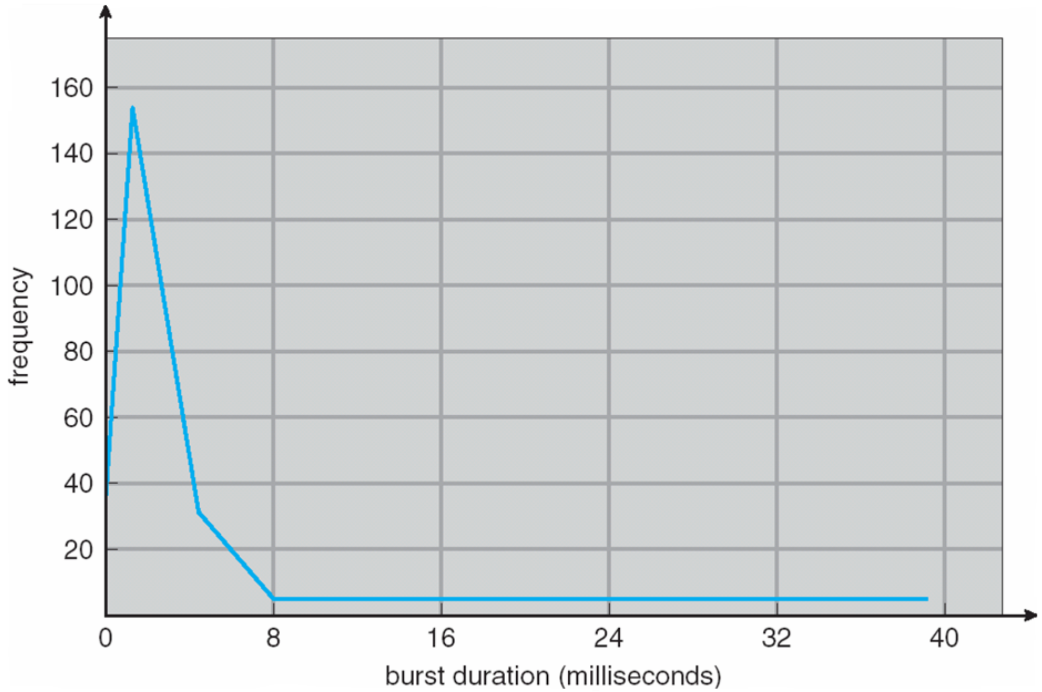
Maximum CPU utilization obtained with multiprogramming

A Process alternates between CPU and I/O burst

CPU burst distribution is of main concern

Characteristics of CPU bursts

- Typically, CPU bursts follow an exponential or hyper-exponential distribution
  - Large number of short CPU bursts
  - Small number of large CPU bursts



Schedulers

- Short-term scheduler (or CPU scheduler) –
  - selects which process from the ready queue should be executed next in CPU
  - Sometimes the only scheduler in a system
  - Short-term scheduler is invoked frequently (milliseconds), hence must be fast

- Long-term scheduler (or job scheduler) – selects which processes should be brought into the ready queue
  - Long-term scheduler is invoked infrequently (seconds, minutes) --> okay if it is relatively slow
  - The long-term scheduler controls the degree of multiprogramming

When is scheduler called?

- A processes switches from RUNNING to WAITING
  - E.g., I/O, wait() call
- A processes switches from RUNNING to READY
  - E.g., timer interrupt, I/O interrupt
- A process terminates
  - E.g., exit() call

Non-preemptive scheduling

- Scheduling happens only when
    - A processes switches from RUNNING to WAITING
    - A process terminates

- Once CPU is allocated to a process, it keeps the CPU for as long as the process requires

Pre-emptive scheduling

- CPU can be forcibly taken away from a running process
  - E.g., due to timer interrupt upon completion of time slice
  - Process moves from RUNNING state to READY state

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler.
- Functions
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program

- Dispatch latency should be low --> dispatcher should be as fast as possible

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)

Next ...

- How to schedule?
- In other words, how to select the process to be run next, from among the processes in the ready queue?

Scheduling criteria

- CPU utilization – keep the CPU as busy as possible
- Throughput – number of processes that complete their execution per time unit (on average)
- Turnaround time – amount of time to completely execute a process (interval from the time of submission of a process to the time of its completion)
- Response time – amount of time from when a process is submitted to the time when the first response is produced (practically more important for interactive system)
- Waiting time – amount of time a process spends waiting in the ready queue
- Burst time – amount of time a process is executed on CPU

Scheduling algorithm optimization criteria

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

CPU scheduling algorithms

Scheduling algorithms

- 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

Scheduling algorithms

- 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)

Example 1

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

Gantt chart for the schedule P1, P2, P3:

Waiting time for P1 = 0; P2 = 24; P3 = 27

Average waiting time: (0 + 24 + 27)/3 = 17

Algo 1. First Come First Serve scheduling (FCFS)

Example 1

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

Gantt chart for the schedule P2, P3, P1:

Waiting time for P1 = 6; P2 = 0; P3 = 3

Average waiting time: (6 + 0 + 3)/3 = 3

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)

Problems with FCFS

- Average waiting time can be long
    - Convoy effect - a process with large CPU burst can delay several processes with shorter CPU bursts (see previous example)

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

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

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: 11/5 ms)

Scheduling algorithms

- 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

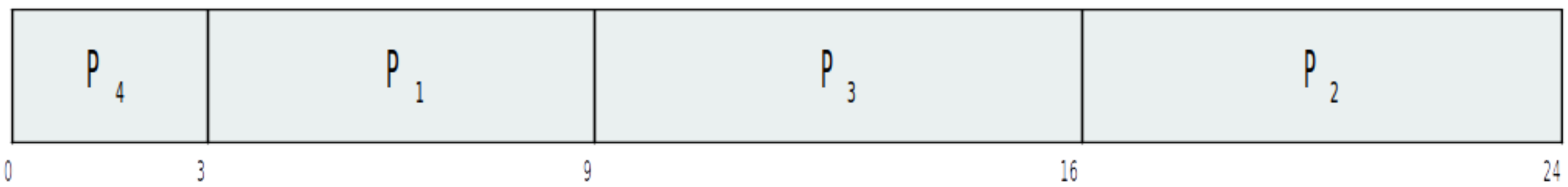
Process	P1	P2	P3	P4
Arrival time	0	0	0	0
CPU burst	6ms	8ms	7ms	3ms

What is the SJF schedule and corresponding wait time?

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?



- Average waiting time = (3 + 16 + 9 + 0) / 4 = 7

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 (assuming estimates are accurate)

- 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.
  - Hint: Prove by 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

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

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

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

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

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

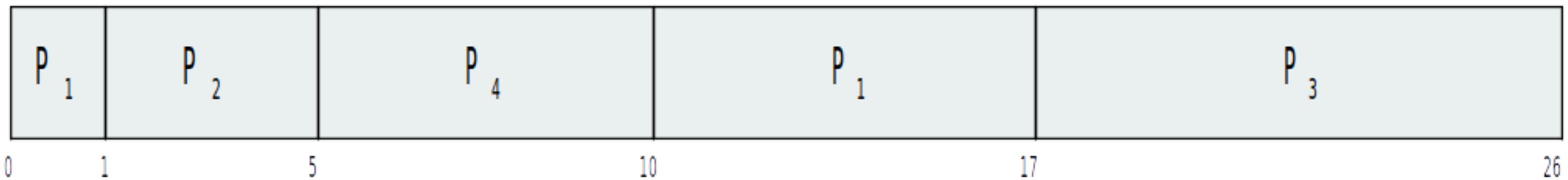
  - Draw preemptive gantt chart and compute waiting time

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 compute waiting time



  - Avg waiting time = [(10-1)+(1-1)+(17-2)+(5-3)]/4 = 26/4 = 6.5 ms

Scheduling algorithms

- 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
  - Can be preemptive or non-preemptive
  - SJF is a type of priority scheduling (priority = inverse of predicted next CPU burst time)

Algo 3. Priority scheduling

- A priority is assigned to each process
  - CPU is allotted to the process with highest priority
  - Can be preemptive or non-preemptive
  - 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? Assume non-preemptive

Algo 3. Priority scheduling

- A priority is assigned to each process
  - CPU is allotted to the process with highest priority
  - Can be preemptive or non-preemptive
  - 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? Assume non-preemptive

Ans: 8.2 ms

Assigning priority: static approach

- Each process has a static priority
  - Large chance of indefinite blocking
  - Can lead to starvation (low priority processes may never execute)

Assigning priority: dynamic approach

- Compute highest response ratio (RN)

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

Assigning priority: dynamic approach

- Compute highest response ratio (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 ratio (RN)

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

- For a waiting process
    - "Time since arrival increase" -> RN increase

Aging

Assigning priority: dynamic approach

- Compute highest response ratio (RN)

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

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

Aging

- 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 ranges 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 ranges 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 of her processes
  - superuser can increase priority of a process

Scheduling algorithms

- 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, this process put at tail of READY queue, 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, this process put at tail of READY queue, next process is loaded from READY queue

Example:

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

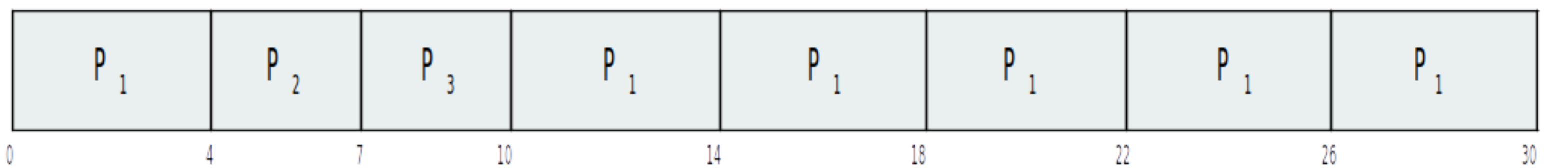
If time quantum $\delta = 4$ ms, then what is the avg. wait time? (schedule P1, P2, P3,…)

RR scheduling: more details

Example:

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

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



Avg. Wait time : (6 + 4 + 7)/3 = 5.66 ms

RR scheduling: Analysis

- n process in READY queue, time slice δ
  - Each process gets 1/n CPU time, in chunks each of which 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, in chunks each of which 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, in chunks each of which 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 $\delta >>> \sigma$ (e.g., $\delta = 10$ ms, $\sigma = 10$ 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

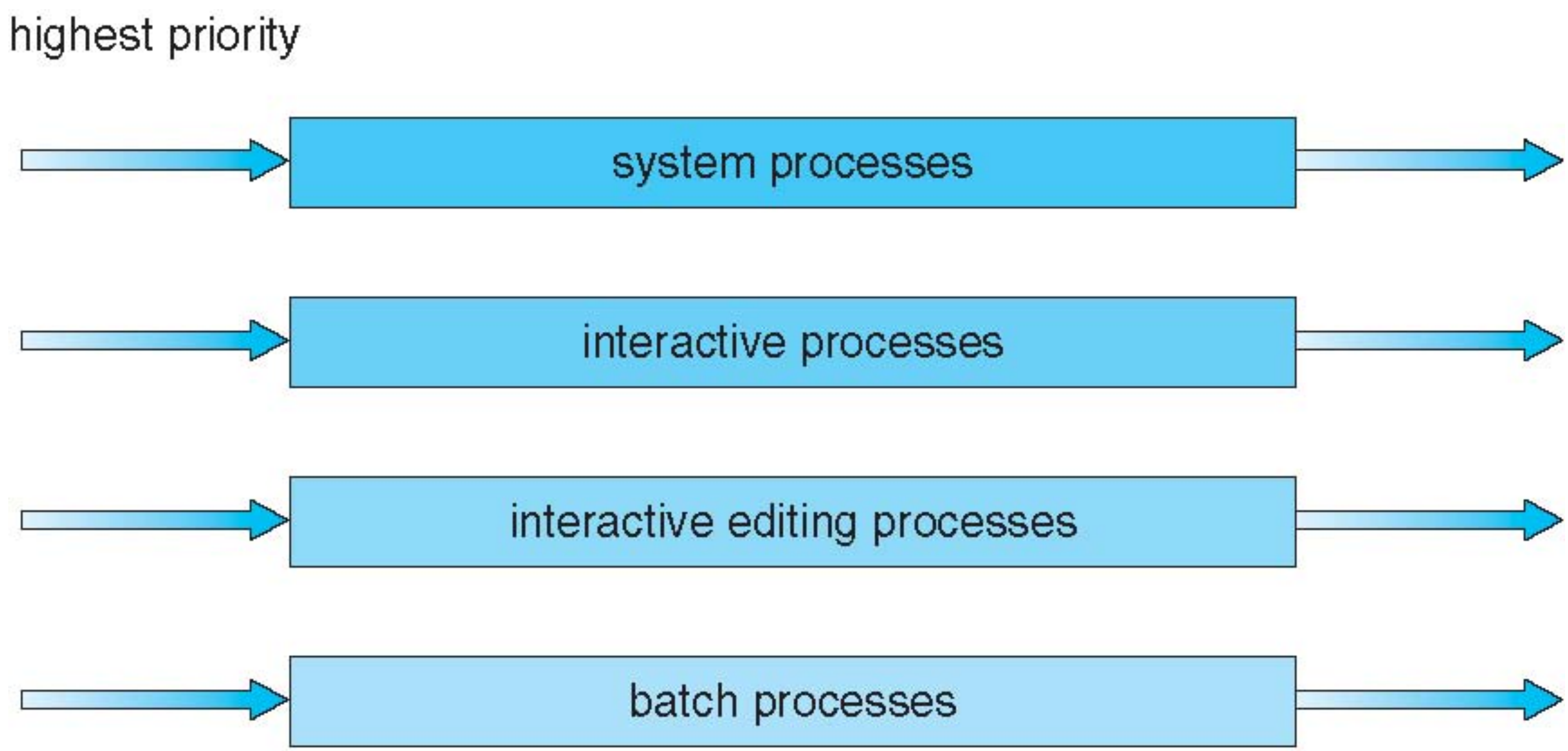
Scheduling algorithms

- 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, e.g.:
  - foreground (interactive)
  - background (batch)
- A process is permanently assigned to a particular queue
- Each queue has its own scheduling algorithm

Multi level queues



Algo 5. Multi level queue scheduling

- Ready queue is partitioned into separate queues, e.g.:
  - foreground (interactive)
  - background (batch)
- A process is permanently assigned to a particular 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, e.g.:
  - foreground (interactive)
  - background (batch)
- A process is permanently assigned to a particular 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

Scheduling algorithms

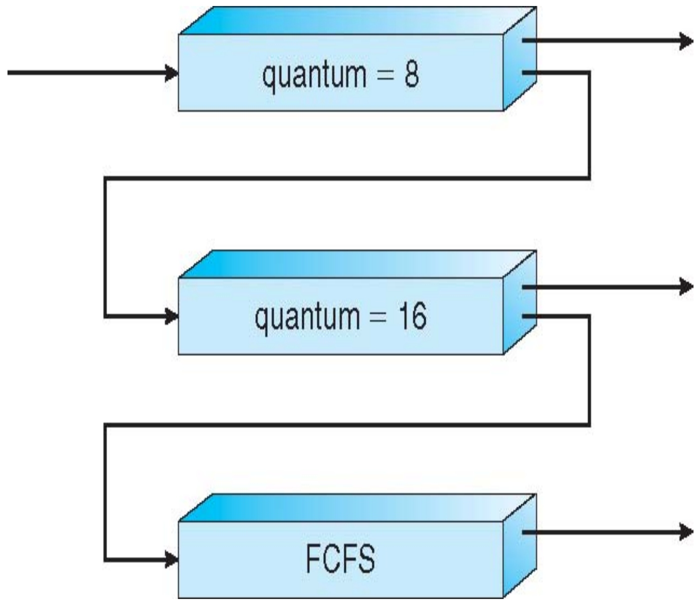
- Algo 1: First come first serve (FCFS)
- Algo 2: Shortest job first (SJF)
- Algo 3: Priority scheduling
- Algo 4: Round robin scheduling
- Algo 5: Multi level queue scheduling
- Algo 6: Multi level feedback queue scheduling

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

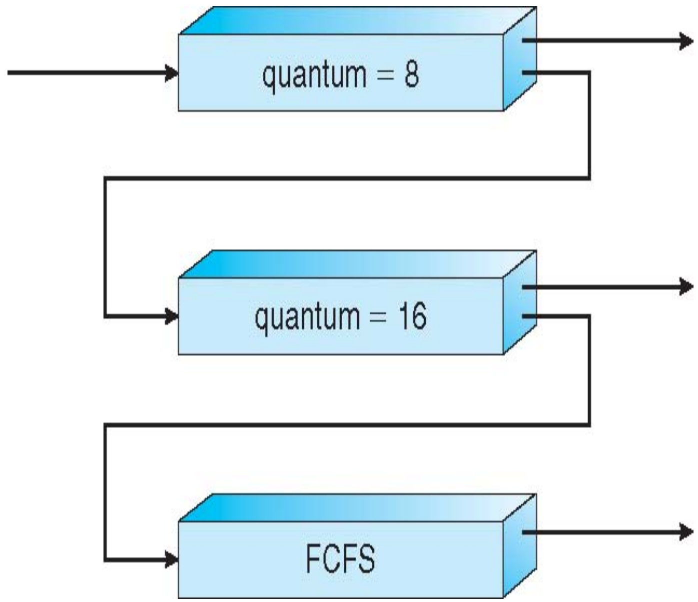
- Three queues:
  - Q_0 – RR with time quantum (δ) 8 ms
  - Q_1 – RR with $\delta = 16$ms
  - Q_2 – FCFS



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

Multi level feedback queue: Example

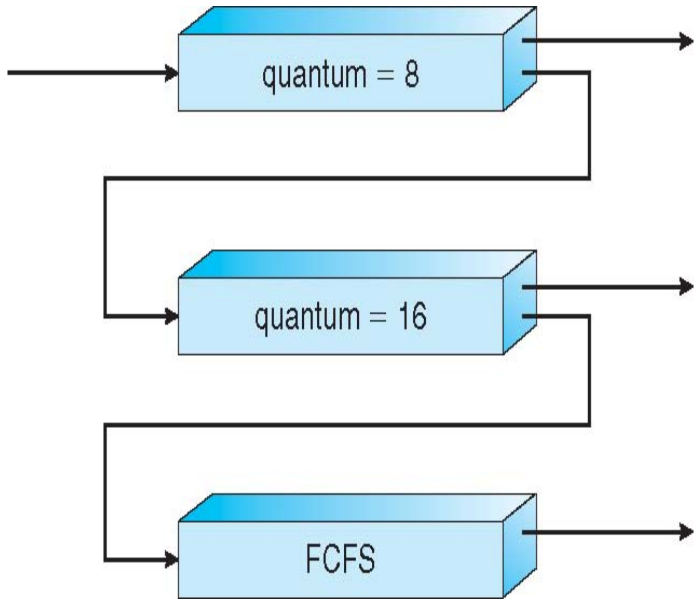
- Three queues:
  - Q_0 – RR with time quantum (δ) 8 ms
  - Q_1 – RR with δ = 16ms
  - Q_2 – FCFS



- 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

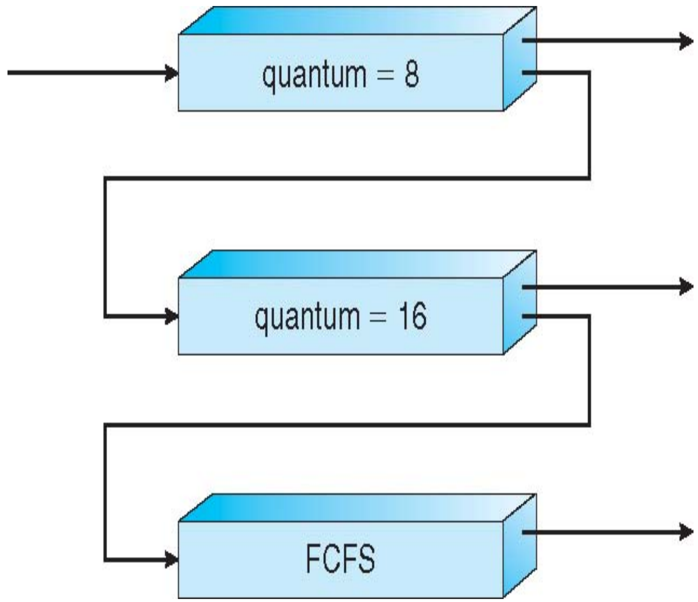
- Three queues:
  - Q_0 – RR with time quantum (δ) 8 ms
  - Q_1 – RR with δ = 16ms
  - Q_2 – FCFS



- 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 it is moved to lower priority queue

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 new process enters Q0; when it gains CPU, receives 8 ms CPU time
- If does not finish in 8 ms, process is moved to Q1
- When process in Q1 gains CPU, receives additional 16 ms CPU time
- If it still does not finish, process moved to Q2

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