CSC139 Operating System Principles

Spring 2019, Part 2-3

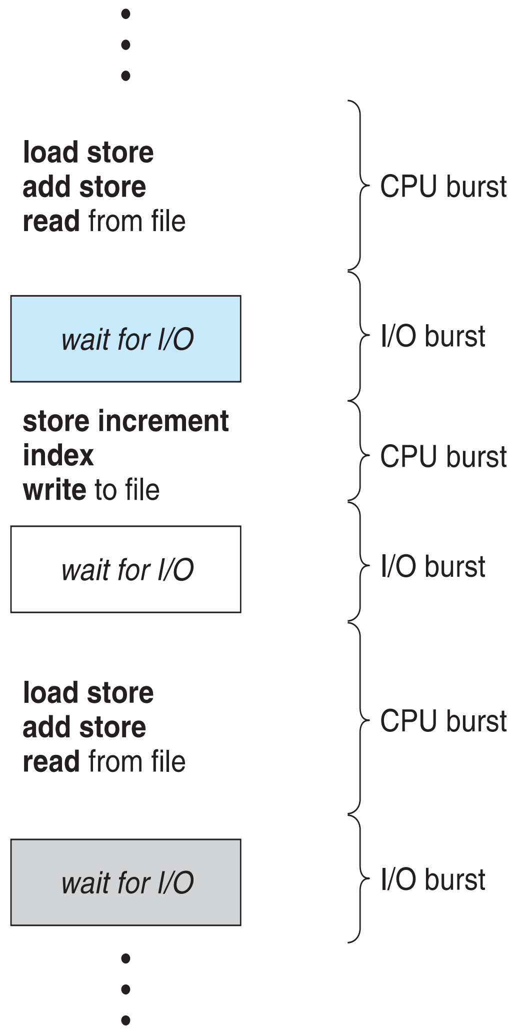
Instructor: Dr. Yuan Cheng

Session Plan

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
    - First-In-First-Out
    - Shortest-Job-First, Shortest-Remaining-Time-First
    - Priority Scheduling
    - Round Robin
    - Multi-level Queue
    - Multi-level Feedback Queue

Basic Concepts

- During its lifetime, a process goes through a sequence of CPU and I/O bursts
- The CPU scheduler (a.k.a. short-term scheduler) will select one of the processes in the ready queue for execution.
- The CPU scheduler algorithm may have tremendous effects on the system performance
  - Interactive systems
  - Real-time systems



When to Schedule?

- Under the simple process state transition model, CPU scheduler can be *potentially* invoked at four different points:
    1. When a process switches from the running state to the waiting state.
    2. When a process switches from the running state to the ready state
    3. When a process switches from the waiting state to the ready state
    4. When a process terminates

Non-preemptive vs. Preemptive Scheduling

- Under *non-preemptive scheduling*, each running process keeps the CPU until it completes or it switches to the waiting (blocked) state
- Under *preemptive scheduling*, a running process may be forced to release the CPU even though it is neither completed nor blocked
  - In time-sharing systems, when the running process reaches the end of its time *quantum (slice)*
  - In general, whenever there is a change in the ready queue

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- Dispatch latency – time it takes for the dispatcher to stop one process and start another running

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)
- Meeting the deadline (real-time systems)

Scheduling Algorithm Optimization Criteria

- Maximize the CPU utilization
- Maximize the throughput
- Minimize the (average) turnaround time
- Minimize the (average) waiting time
- Minimize the (average) response time

Waiting Time

- Waiting time definition

$$T_{waiting} = T_{start} - T_{arrival}$$

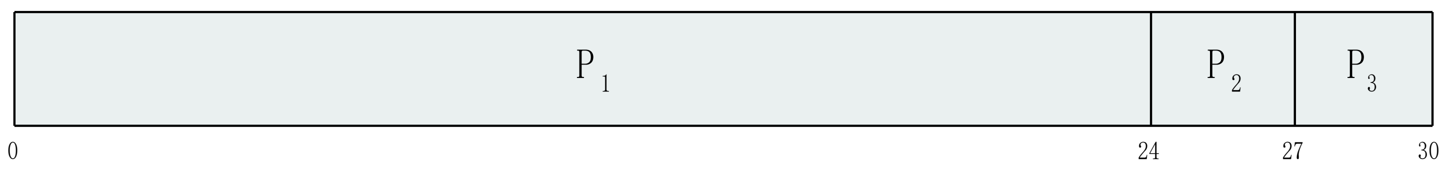
- Average waiting time = Sum($T_{waiting}$)/ #processes

- For now, we assume
  - Average waiting time is the performance measure
  - Only one CPU burst (in milliseconds) per process
  - Only CPU, No I/O
  - Once started, each process runs to completion

First-Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
  The Gantt Chart for the schedule is:



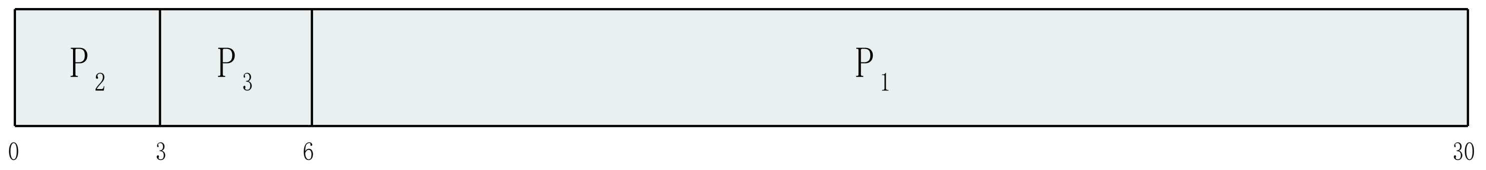
- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order:

P_2 , P_3 , P_1

- The Gantt chart for the schedule is:



- Waiting time for $P_1 = 6;$ $P_2 = 0$; $P_3 = 3$
- Average waiting time: (6 + 0 + 3)/3 = 3
- Much better than previous case

FCFS Advantages and Disadvantages

- Advantage: simple
- Disadvantages:
  - convoy effect - short process behind long process
  - may lead to poor overlap of I/O and CPU since CPU-bound processes will force I/O bound processes to wait for the CPU, leaving the I/O devices idle

Convoy Effect

Workload Assumptions

1. ~~Each job runs for the same amount of time~~
2. ~~All jobs arrive at the same time~~
3. All jobs only use the CPU (no I/O)
4. The run time of each job is known

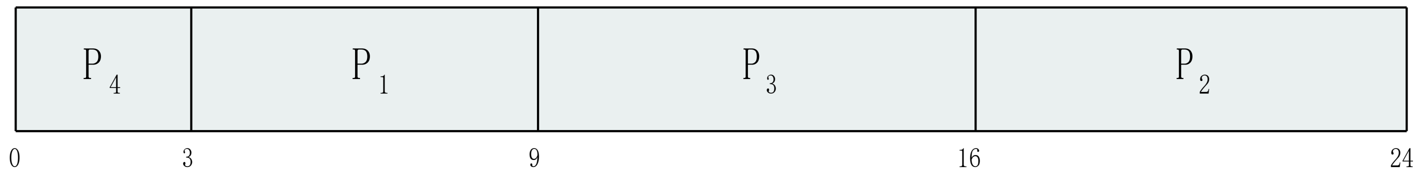
Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time
- SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request
  - Could ask the user
- Two schemes:
  - Non-preemptive
  - Preemptive: a.k.a. Shortest-Remaining-Time-First (SRTF)

Example of SJF

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- SJF scheduling chart



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

A Preemptive Scheduler

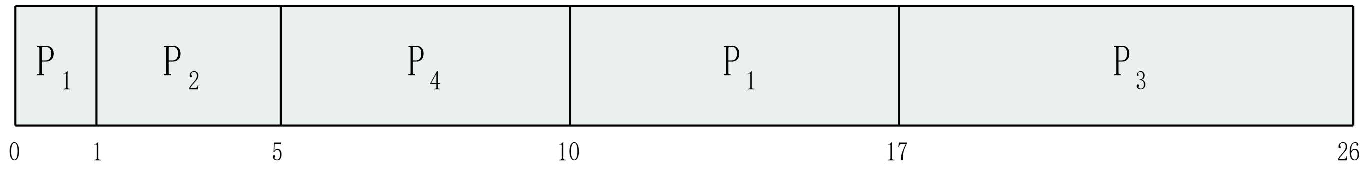
- Previous schedulers: FIFO and SJF are non-preemptive
- New scheduler: SRTF (Shortest Remaining Time First)
- Policy: Switch jobs so we always run the one that will complete the quickest

Example of Shortest-remaining-time-first

- Now we add the concepts of varying arrival times and preemption to the analysis

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

- *Preemptive* SJF Gantt Chart



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

Optimality of SJF and SRTF

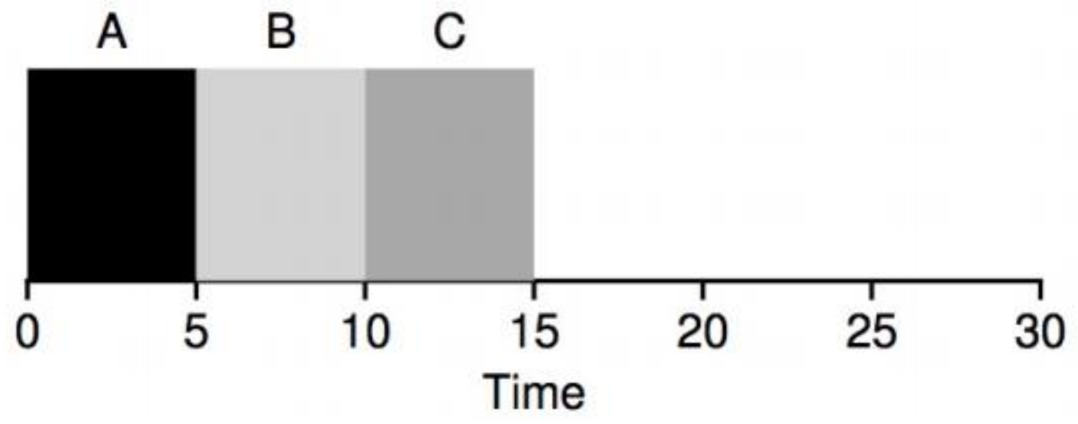
- Non-preemptive SJF is optimal if all the processes are ready simultaneously
  - Gives minimum average waiting time for a given set of processes
- SRTF is optimal if what?
  - If the processes may arrive at different times

Response Time

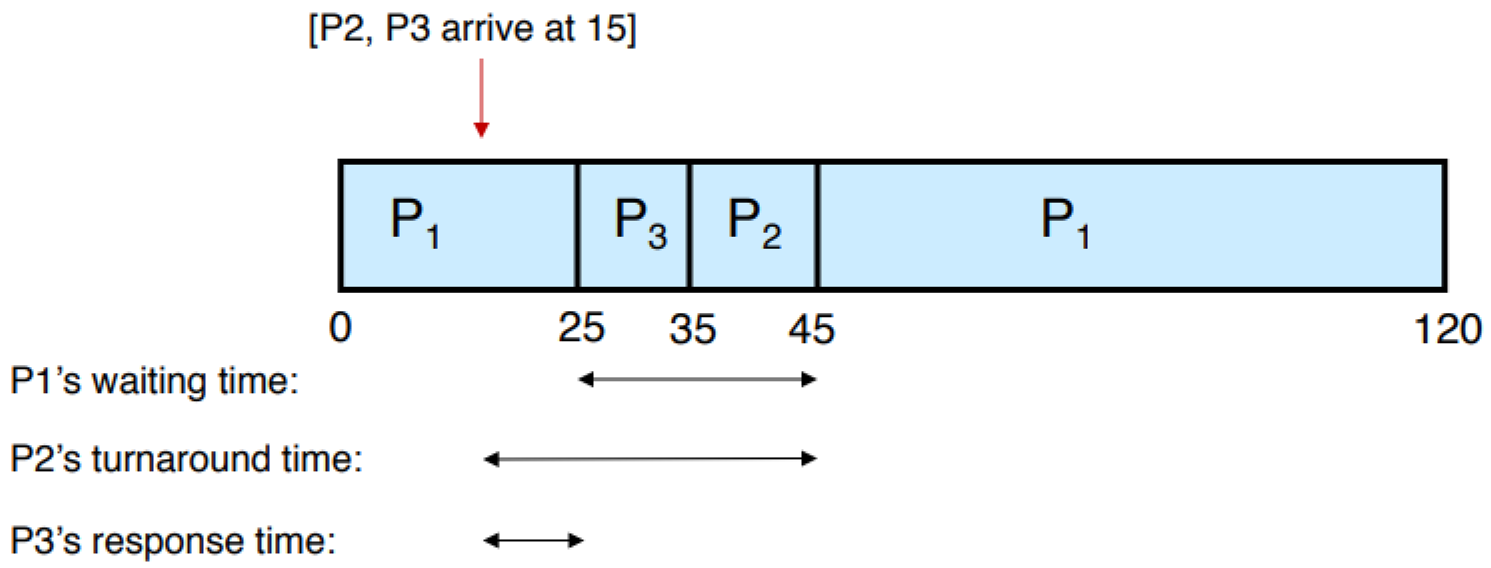
- Response time definition

$$T_{response} = T_{first\ run} - T_{arrival}$$

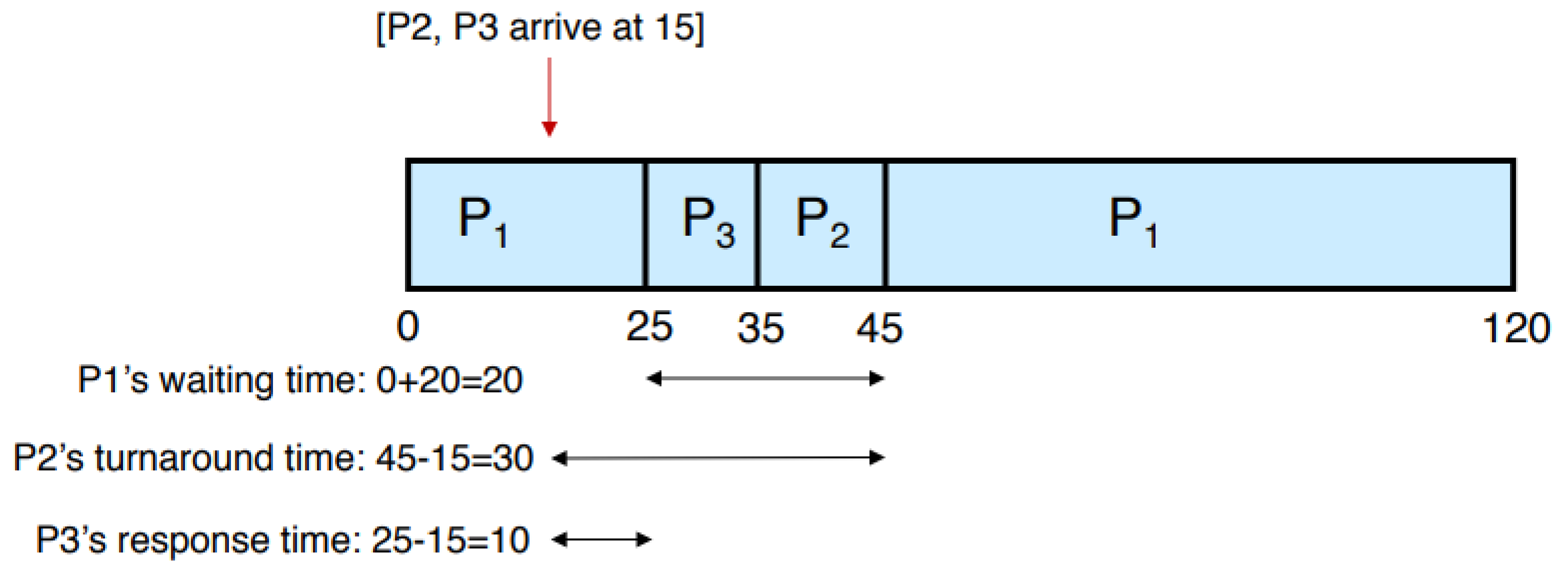
- SJF's average response time (all 3 jobs arrive at same time)
  - (0+5+10)/3 = 5



Waiting, Turnaround, Response



Waiting, Turnaround, Response



[P2, P3 arrive at 15]
P_1
P_3
P_2
P_1
0
25
35
45
120
P1's waiting time: 0+20=20
P2's turnaround time: 45-15=30
P3's response time: 25-15=10

Q: What is P1's response time?

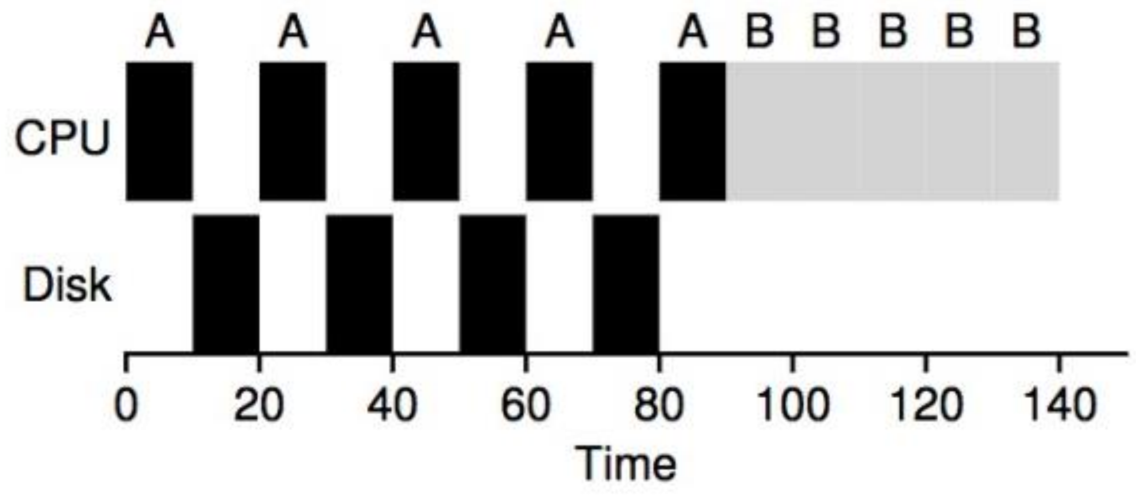
Workload Assumptions

1. ~~Each job runs for the same amount of time~~
2. ~~All jobs arrive at the same time~~
3. ~~All jobs only use the CPU (no I/O)~~
4. The run time of each job is known

Extension to Multiple CPU & I/O Bursts

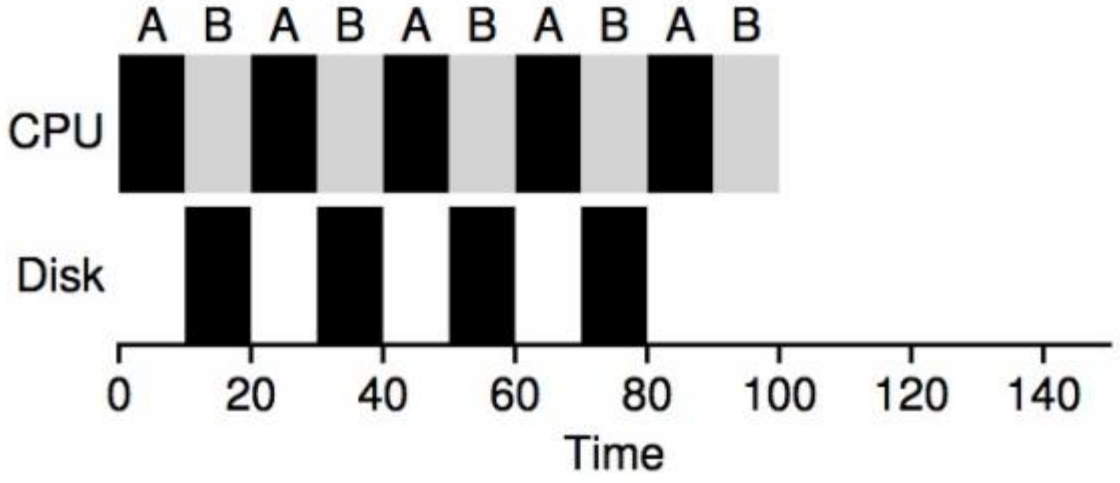
- When the process arrives, it will try to execute its first CPU burst
  - It will join the ready queue
  - The priority will be determined according to the underlying scheduling algorithm and considering only that specific (i.e., first) burst
- When it completes its first CPU burst, it will try to perform its first I/O operation (burst)
  - It will join the device queue
  - When the device is available, it will use the device for a time period indicated by the length of the first I/O burst
- Then, it will re-join the ready queue and try to execute its second CPU burst
  - Its new priority may now change (as defined by its second CPU burst)

Not I/O Aware



Poor use of resources

I/O Aware (Overlap)



Overlap allows better use of resources

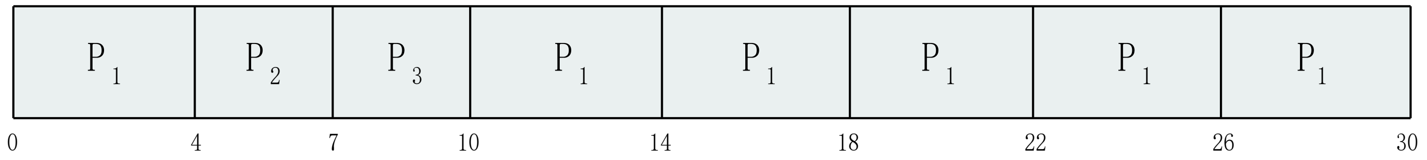
Round Robin (RR)

- Each process gets a small unit of CPU time (time quantum *q*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are *n* processes in the ready queue and the time quantum is *q*, then each process gets 1/*n* of the CPU time in chunks of at most *q* time units at once. No process waits more than $(n-1)q$ time units.
- Timer interrupts every quantum to schedule next process
- Performance
  - *q* large $\Rightarrow$ FIFO
  - *q* small $\Rightarrow$ *q* must be large with respect to context switch, otherwise overhead is too high

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

- The Gantt chart is:



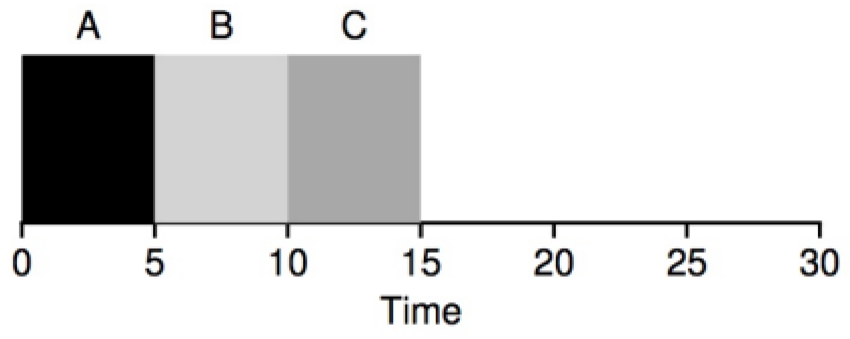
- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 usec

RR vs. SJF

Process	Burst Time
A	5
B	5
C	5

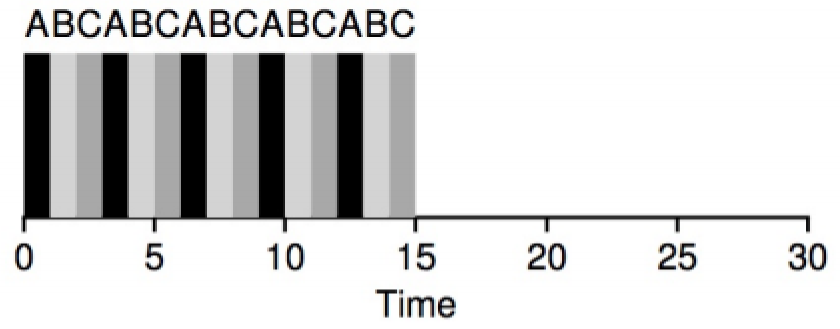
SJF's average response time

– (0 + 5 + 10) / 3 = 5



RR's average response time (time quantum = 1)

– (0 + 1 + 2) / 3 = 1



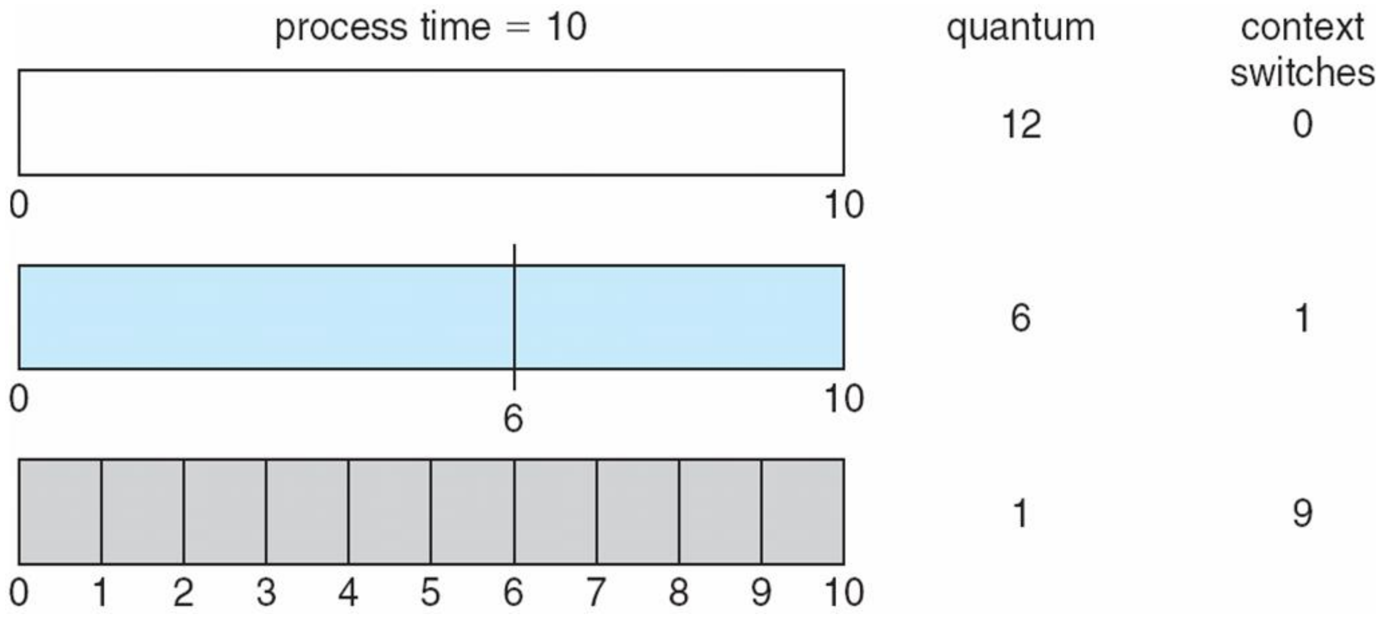
Tradeoff Consideration

- Typically, RR achieves higher average turnaround time than SJF, but better response time
  - Turnaround time only cares about when processes finish
- RR is one of the worst policies
  - If turnaround time is the metric

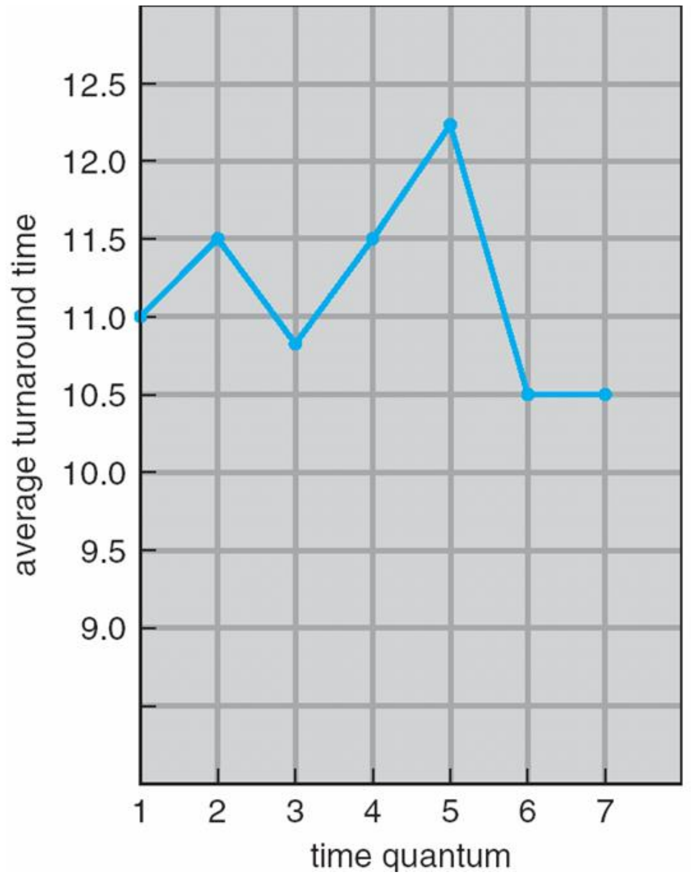
Choosing a Time Quantum

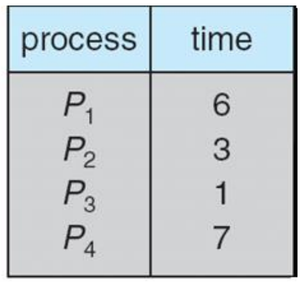
- The effect of quantum size on context-switching time must be carefully considered
- The time quantum must be large with respect to the context-switch time
- Turnaround time also depends on the size of the time quantum

Time Quantum and Context Switch Time



Turnaround Time Varies With The Time Quantum





process	time
P_1	6
P_2	3
P_3	1
P_4	7

80% of CPU bursts
should be shorter than q

Workload Assumptions

1. ~~Each job runs for the same amount of time~~
2. ~~All jobs arrive at the same time~~
3. ~~All jobs only use the CPU (no I/O)~~
4. ~~The run time of each job is known~~

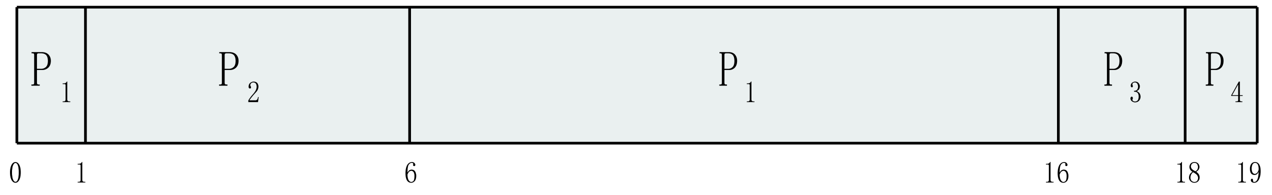
Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - Non-preemptive
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ≡ Starvation – low priority processes may never execute
- Solution ≡ Aging – as time progresses increase the priority of the process

Example of Priority Scheduling

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

- Priority scheduling Gantt Chart

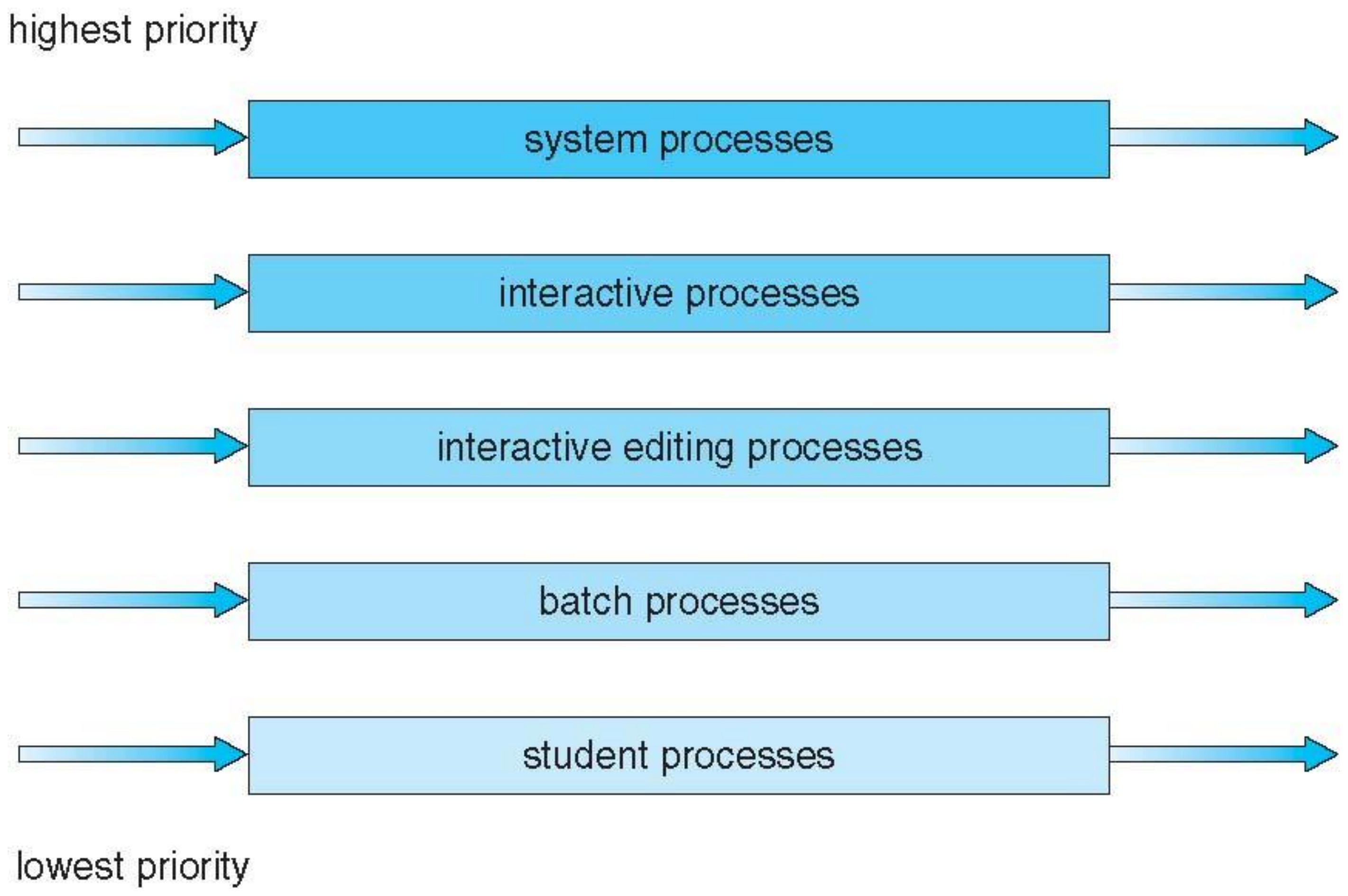


- Average waiting time = 8.2 msec

Multilevel Queue

- Ready queue is partitioned into separate queues, e.g.,
  - foreground (interactive)
  - background (batch)
- Process permanently in a given queue
- Each queue has its own scheduling algorithm:
  - foreground – RR
  - background – FCFS
- Scheduling must be done between the queues:
  - Fixed priority scheduling; (i.e., 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

Multilevel Queue Scheduling



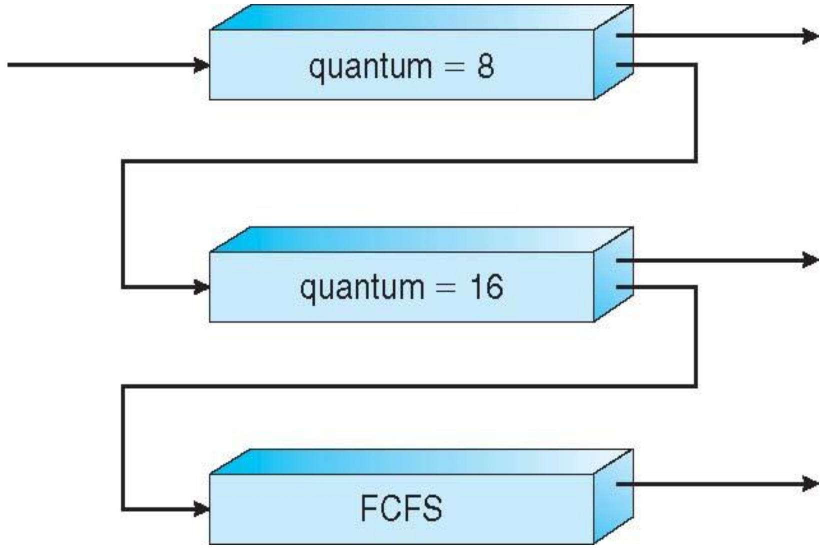
Multilevel Feedback Queue

- A 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 – RR with time quantum 8 milliseconds
  - Q_1 – RR 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



Operating System Examples

- Linux scheduling

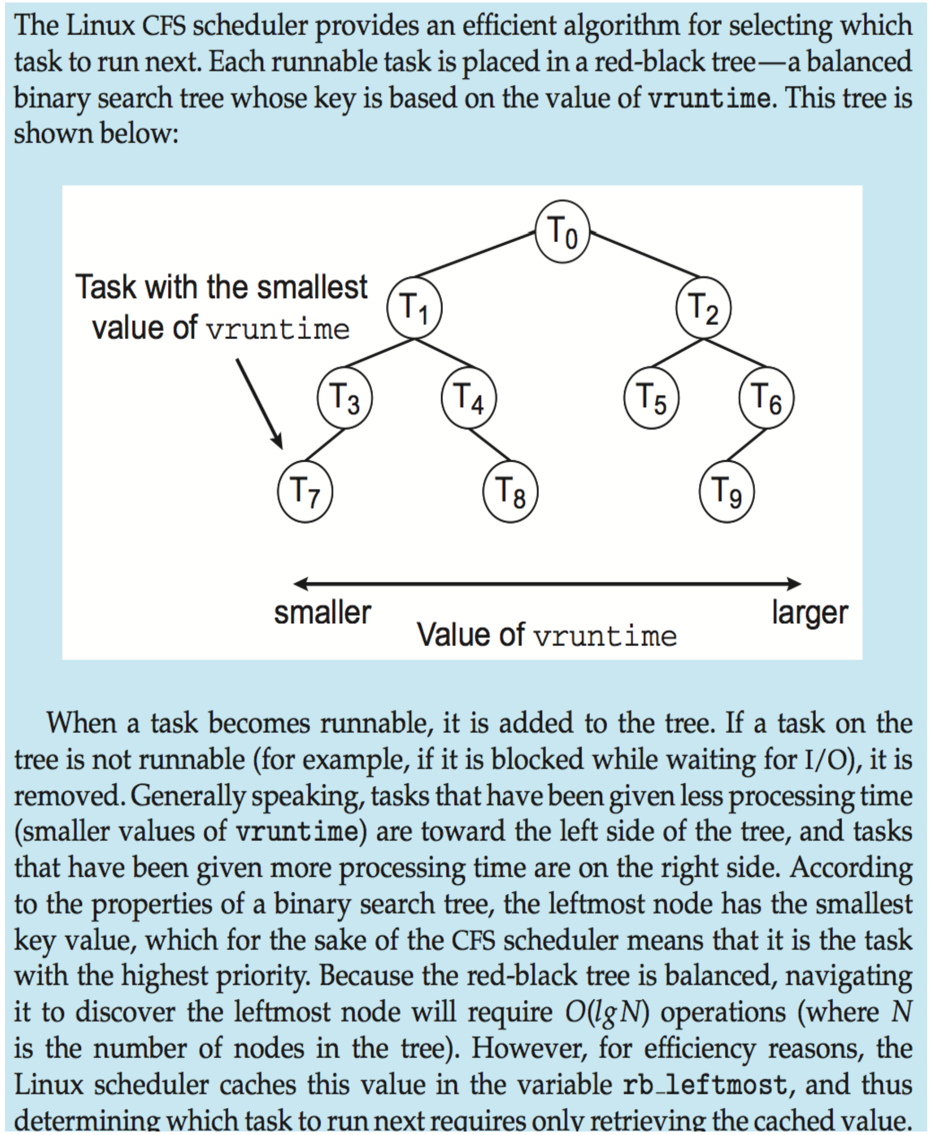
Linux Scheduling Through Version 2.5

- Prior to kernel version 2.5, ran variation of standard UNIX scheduling algorithm
- Version 2.5 moved to constant order *O*(1) scheduling time
    - Preemptive, priority based
    - Two priority ranges: time-sharing and real-time
    - **Real-time** range from 0 to 99 and **nice** value from 100 to 140
    - Map into global priority with numerically lower values indicating higher priority
    - Higher priority gets larger q
    - Task run-able as long as time left in time slice (active)
    - If no time left (expired), not run-able until all other tasks use their slices
    - All run-able tasks tracked in per-CPU runqueue data structure
        - Two priority arrays (active, expired)
        - Tasks indexed by priority
        - When no more active, arrays are exchanged
    - Worked well, but poor response times for interactive processes

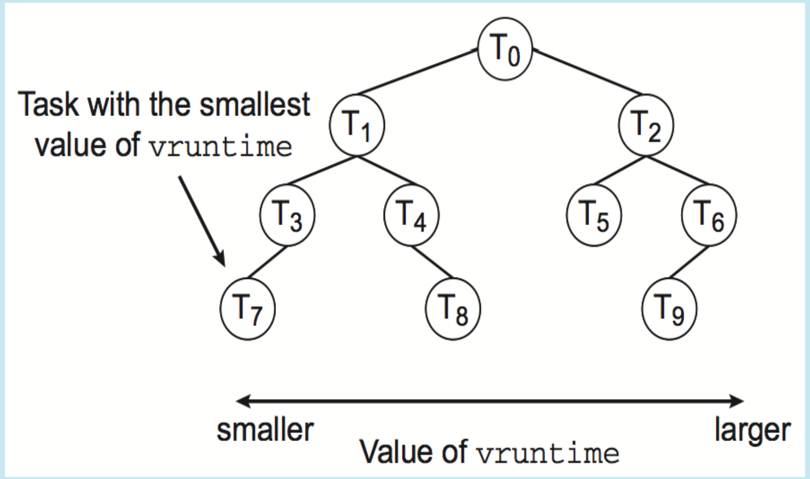
Linux Scheduling in Version 2.6.23 +

- ***Completely Fair Scheduler*** (CFS)
- Scheduling classes
  - Each has specific priority
  - Scheduler picks highest priority task in highest scheduling class
  - Rather than quantum based on fixed time allotments, based on proportion of CPU time
  - 2 scheduling classes included, others can be added
    1. default
    2. real-time
- Quantum calculated based on nice value from -20 to +19
  - Lower value is higher priority
  - Calculates target latency – interval of time during which task should run at least once
  - Target latency can increase if say number of active tasks increases
- CFS scheduler maintains per task virtual run time in variable **`vruntime`**
  - Associated with decay factor based on priority of task – lower priority is higher decay rate
  - Normal default priority yields virtual run time = actual run time
- To decide next task to run, scheduler picks task with lowest virtual run time

CFS Performance



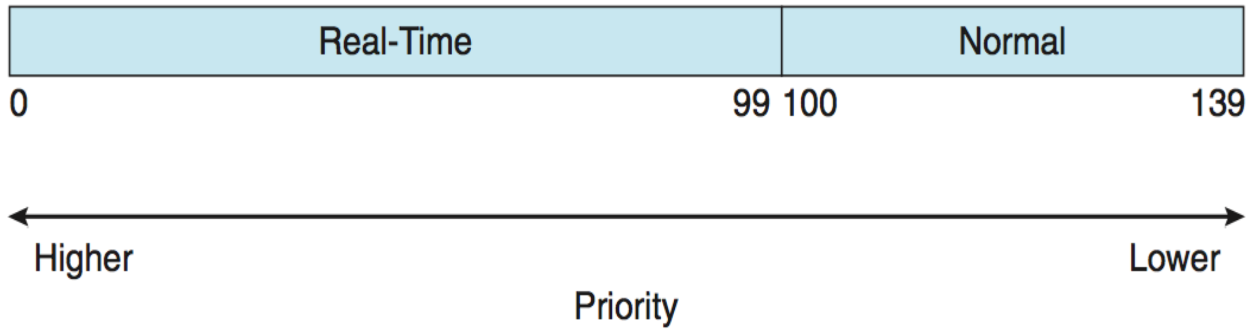
The Linux CFS scheduler provides an efficient algorithm for selecting which task to run next. Each runnable task is placed in a red-black tree—a balanced binary search tree whose key is based on the value of `vruntime`. This tree is shown below:



When a task becomes runnable, it is added to the tree. If a task on the tree is not runnable (for example, if it is blocked while waiting for I/O), it is removed. Generally speaking, tasks that have been given less processing time (smaller values of `vruntime`) are toward the left side of the tree, and tasks that have been given more processing time are on the right side. According to the properties of a binary search tree, the leftmost node has the smallest key value, which for the sake of the CFS scheduler means that it is the task with the highest priority. Because the red-black tree is balanced, navigating it to discover the leftmost node will require $O(lgN)$ operations (where N is the number of nodes in the tree). However, for efficiency reasons, the Linux scheduler caches this value in the variable `rb_leftmost`, and thus determining which task to run next requires only retrieving the cached value.

Linux Scheduling (cont.)

- Real-time scheduling according to POSIX.1b
  - Real-time tasks have static priorities
- Real-time plus normal map into global priority scheme
- Nice value of -20 maps to global priority 100
- Nice value of +19 maps to priority 139



Exit Slips

- Take 1-2 minutes to reflect on this lecture
- On a sheet of paper write:
  - One thing you learned in this lecture
  - One thing you didn’t understand

Next class

- We will discuss:
  - Process Synchronization
- Reading assignment:
  - SGG: Ch. 6

Acknowledgment

- The slides are partially based on the ones from
  - The book site of *Operating System Concepts (Tenth Edition)*: http://os-book.com/