5 CPU Scheduling

Contents

- What is CPU scheduling
- Process behavior characteristics
- Scheduling criteria
- FCFS scheduling
- Round-robin scheduling
- Priority scheduling
- Shortest-job-first scheduling
- Multi-level feedback queue scheduling

CPU Scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them
- CPU scheduling decisions may take place when a process:
  1. switches from running to waiting state (I/O request)
  2. terminates
  3. switches from running to ready state (timer interrupt)
  4. switches from waiting to ready (I/O completion)
- Preemptive vs. nonpreemptive scheduling
  – Scheduling under 1 and 2 is nonpreemptive (i.e. process runs until voluntarily relinquish CPU)
  – Scheduling under all others (including 1 and 2) is preemptive

Dispatcher

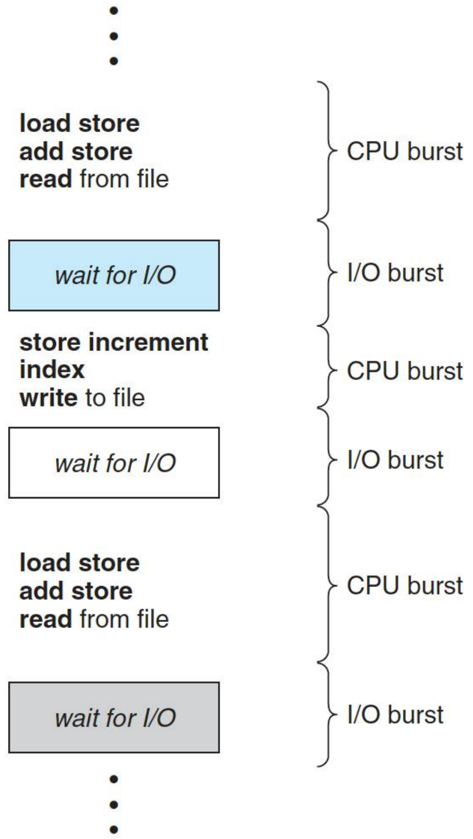
- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- Dispatch latency – time it takes for the dispatcher to stop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

Process Model

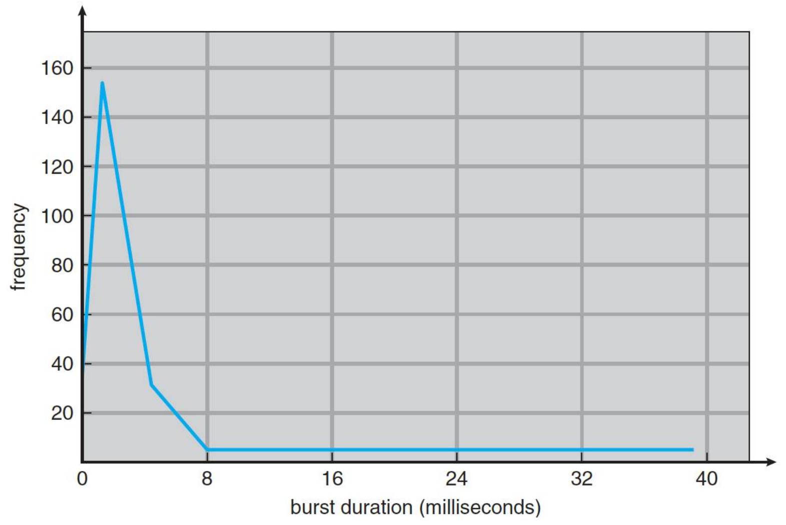
- Process alternates between CPU bursts and I/O bursts
  - CPU burst = CPU execution, I/O burst = I/O wait
  - CPU-bound process: long CPU bursts
  - I/O-bound process: short CPU bursts
  - Problem: *don't know process's type before running*
- An underlying assumption:
  - "response time" is most important for interactive jobs, which will be I/O bound

Characteristics of Process Behavior

Alternating Sequence of CPU and I/O Bursts



Histogram of CPU-burst Times



Goals of “The Perfect CPU Scheduler”

- 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)

- Min response time / turnaround time
  - response time is what the user sees; elapsed time to echo keystroke to editor (acceptable delay ~ 50-150 millisec)
- Max throughput
  - minimize overhead (context switching)
  - efficient use of resources (CPU, disk, cache, …)
- Fairness
  - everyone gets to make progress, no one starves
  - Tension: unfair makes system faster…
- Max CPU utilization

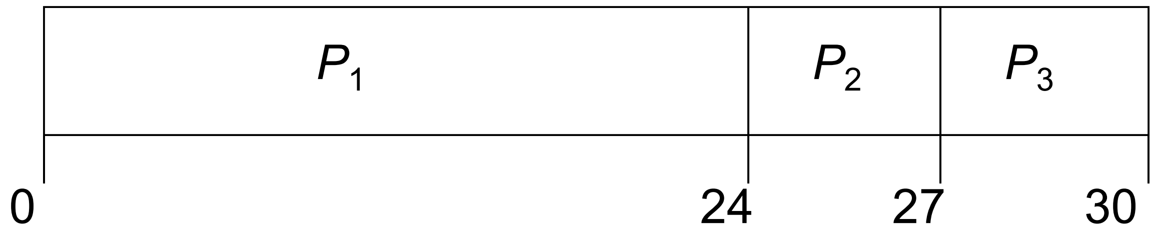
First Come First Served (FCFS or FIFO)

- Simplest scheduling algorithm:
  - Run jobs in order that they arrive at ready queue
  - Uni-programming: Run until done (non-preemptive)
  - Multi-programming: put job at back of queue when blocks on I/O
- Advantage: dirt simple
- Disadvantage
  - wait time depends on arrival order (convoy effects)
  - unfair to later jobs (worst case: long job arrives first)
    - example: three jobs (times: A=100, B=1, C=2) arrive nearly simultaneously – what’s the average completion time?

Example of FCFS Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

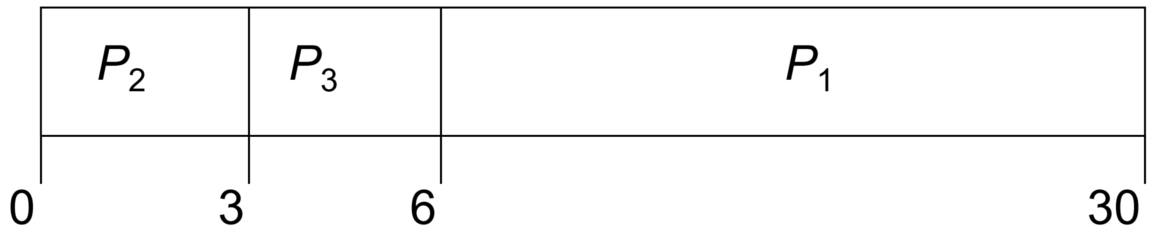
- Suppose that the processes arrive in the order: P_1 , P_2 , P_3 , the Gantt Chart for the schedule is:



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

Example of 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
- Convoy effect: short process behind long process

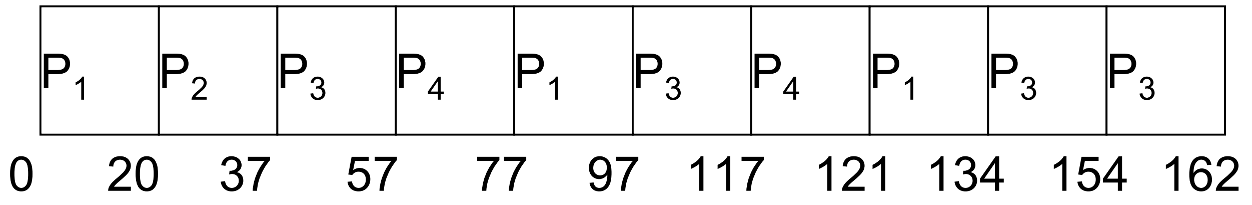
Round Robin (RR)

- Solution to job monopolizing CPU? Interrupt it. (Use a timer)
- Each process gets a small unit of CPU time (*time quantum* or *time slice*), 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.
- Performance
  - q large ®FIFO
  - q small ®q must be large with respect to context switch, otherwise overhead is too high

Example of RR with Time Quantum = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- The Gantt chart is:



Typically, higher average turnaround than SJF, but better *response*

RR Time Quantum Tradeoffs

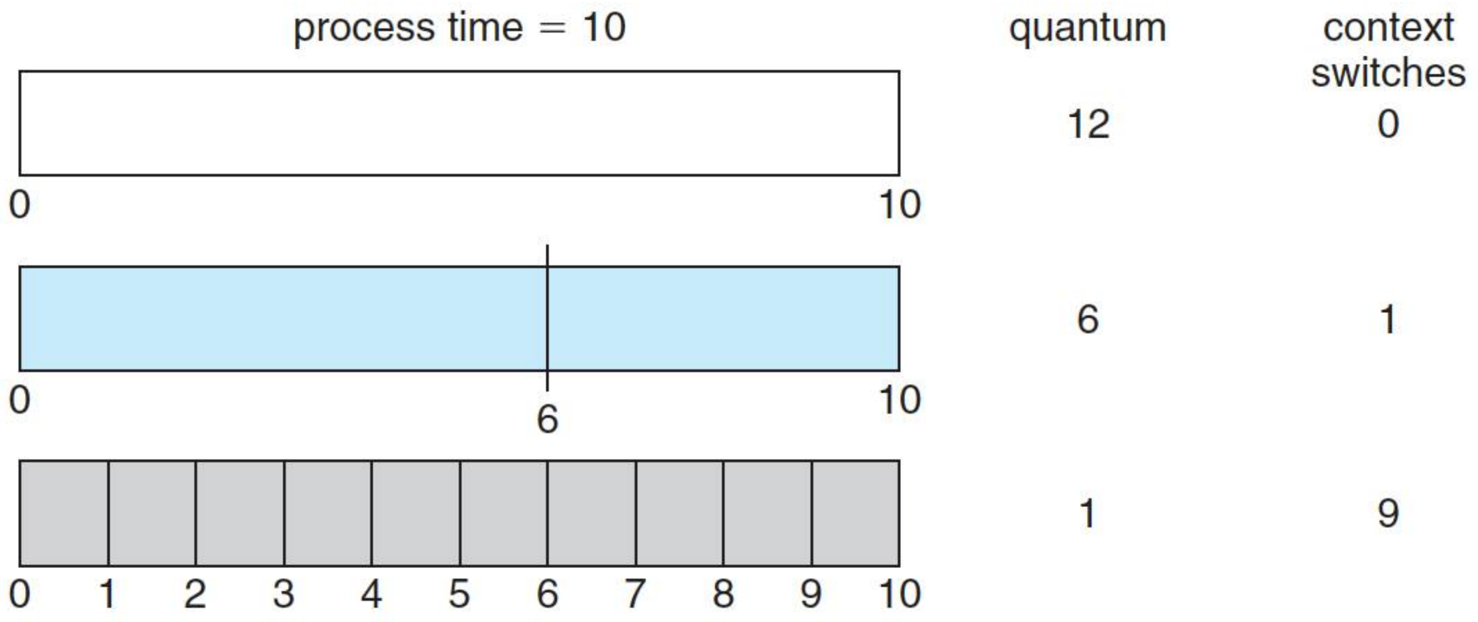
- Performance depends on length of the timeslice
  - Context switching isn't a free operation.
  - If timeslice time is set too high (attempting to amortize context switch cost), you get FCFS. (ie processes will finish or block before their slice is up anyway)

  - If it's set too low you're spending all of your time context switching between threads.

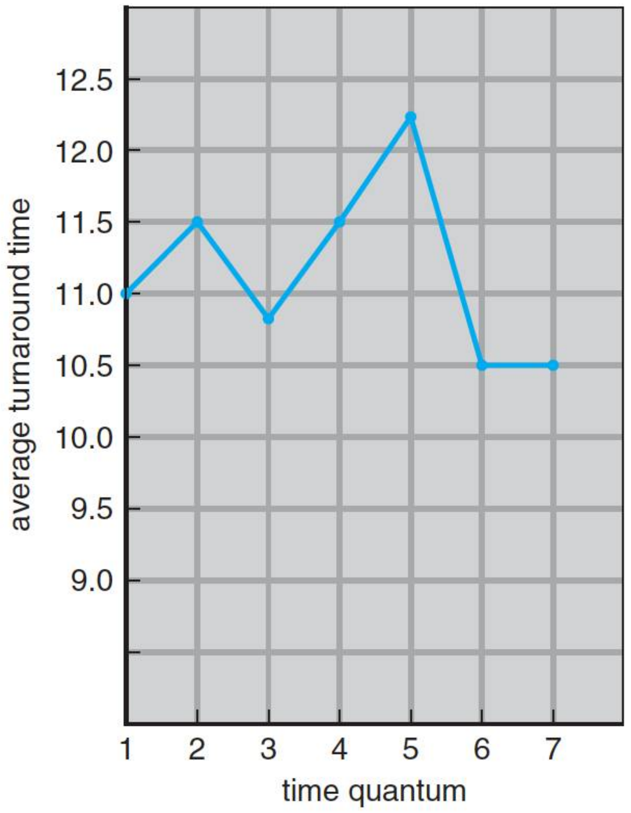
  - Timeslice frequently set to ~100 milliseconds
  - Context switches typically cost < 1 millisecond

  - Moral: context switching is usually negligible (< 1% per timeslice in above example) unless you context switch too frequently and lose all productivity.

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

Priority Scheduling

- A priority number (integer) is associated with each process; the CPU is allocated to the process with the highest priority (smallest integer α highest priority)
- Preemptive vs. nonpreemptive
- Static vs. dynamic or both
- Common use: couple priority to job characteristic
  - Fight starvation? Increase priority as (time last ran)
  - Keep I/O busy? Increase priority for jobs that often block on I/O
- Problem α Starvation – low priority processes may never execute
- Solution α Aging – as time progresses increase the priority of the process

Handling thread dependencies

- Priority inversion e.g., P_1 at high priority, P_2 at low
  - P_2 acquires lock L.
  - Scene 1: P_1 tries to acquire L, fails, spins. P_2 never gets to run.
  - Scene 2: P_1 tries to acquire L, fails, blocks. P_3 enters system at medium priority. P_2 never gets to run.
- Scheduling = deciding who should make progress
  - Obvious: a thread’s importance should increase with the importance of those that depend on it.
  - Naïve priority schemes violate this
- “Priority donation”
  - Thread’s priority scales w/ priority of dependent threads

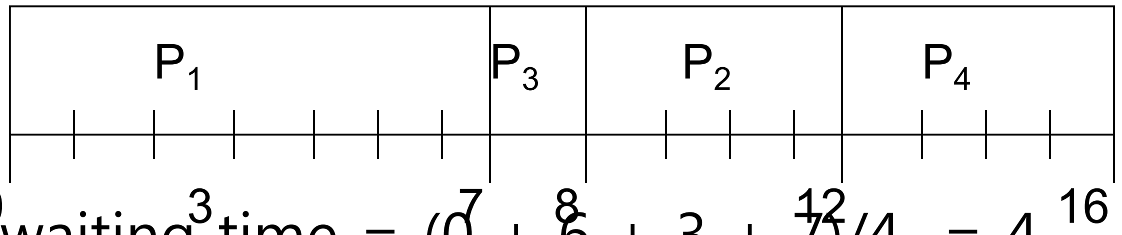
Shortest-Job-First (SJR) Scheduling

- Shortest Time to Completion First (STCF)
- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time
- Two schemes:
  - nonpreemptive – once CPU given to the process it cannot be preempted until completes its CPU burst
  - preemptive – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the Shortest-Remaining-Time-First (SRTF)
- SJF is optimal – gives minimum average waiting time for a given set of processes

Example of Non-Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)

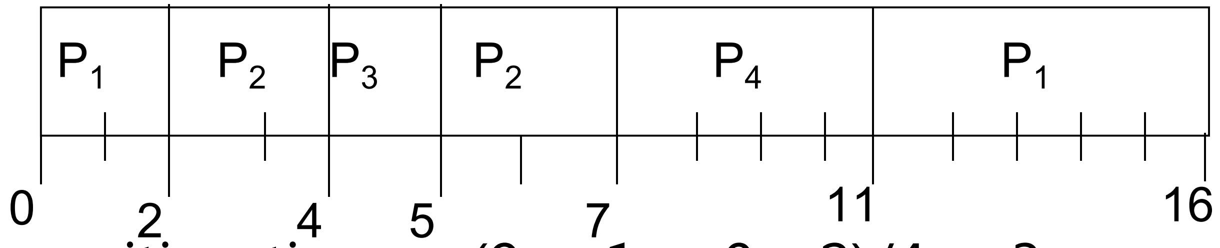


- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

Example of Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive)



- Average waiting time = (9 + 1 + 0 +2)/4 = 3

How to know job length?

- Have user tell us. If they lie, kill the job.
  - Not so useful in practice
- Use the past to predict the future #1:
  - long running job will probably take a long time more
- Use the past to predict the future #2:
  - view job as sequence of sequentially alternating CPU and I/O jobs
  - If previous CPU jobs in the sequence have run quickly, future ones will to ("usually")
  - What to do if past != future?

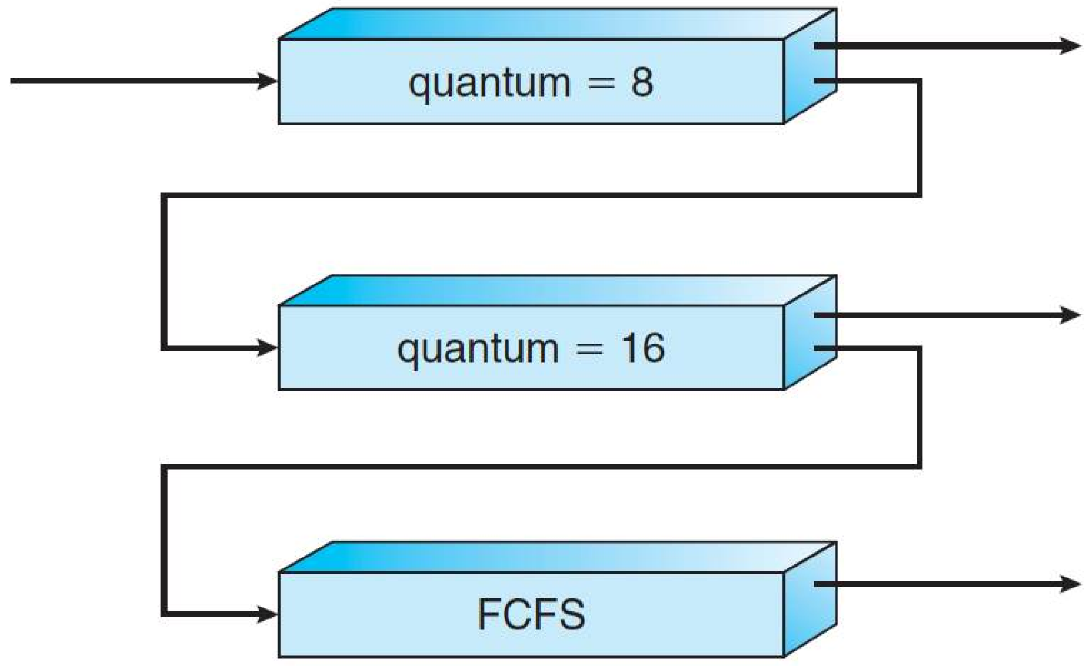
Multilevel Feedback Queue

- 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.

Multilevel Feedback Queues



Summary

- FIFO:
  - + simple
  - - short jobs can get stuck behind long ones; poor I/O
- RR:
  - + better for short jobs
  - - poor when jobs are the same length
- STCF:
  - + optimal (ave. response time, ave. time-to-completion)
  - - hard to predict the future
  - - unfair
- Multi-level feedback:
  - + approximate STCF
  - - unfair to long running jobs