Announcements

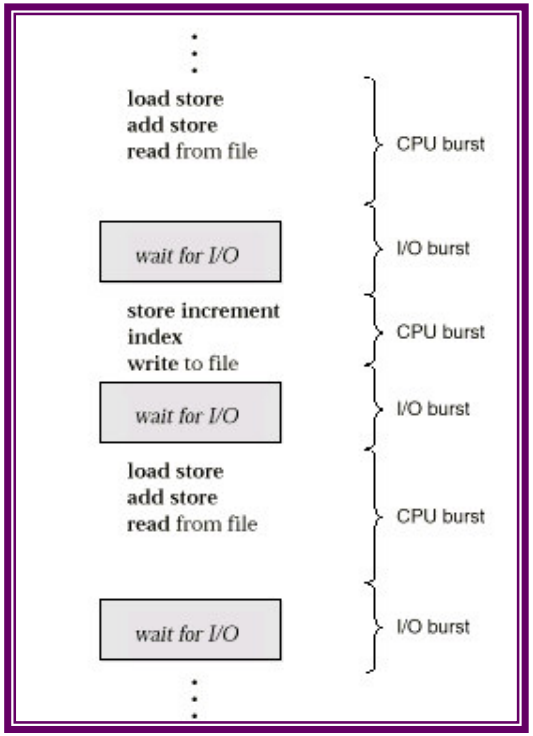
- Subtopics for next lecture?
- Linux/Windows 2000 teams?
- Assignment 1 progress and questions?
- Questions from last lecture?
- Questions on slides from this lecture?
- Invite friends, family at next lecture.
- Revisit ThreadLocal.

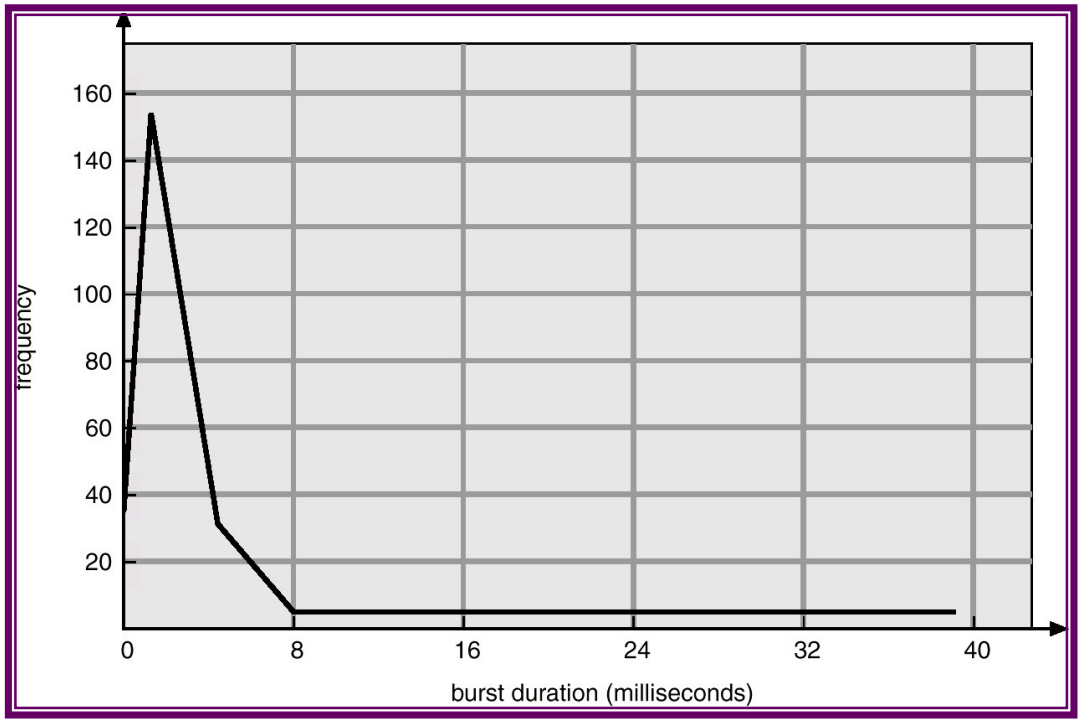
Chapter 6: CPU Scheduling

- Basic Concepts.
- Scheduling Criteria.
- Scheduling Algorithms.
- Multiple-Processor Scheduling.
- Real-Time Scheduling.
- Algorithm Evaluation.

Basic Concepts

- Multiprogramming achieves maximum CPU utilization.
- CPU–I/O Burst Cycle – Process execution consists of a *cycle* of CPU execution and I/O wait. Relative ratio distinguishes I/O- vs. CPU- bound processes.
- CPU burst distribution helps select and/or fine-tune *CPU scheduling* algorithm.





CPU Scheduler

- Selects from among the processes (or kernel threads) 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 (e.g. read(), wait()).
  2. Switches from running to ready state (e.g. timer interrupt).
  3. Switches from waiting to ready (e.g. I/O completed).
  4. Terminates.
- Scheduling under 1 and 4 is *nonpreemptive*: processes willingly relinquish control of CPU.
- All other scheduling is *preemptive*:
  - Under 2: process kicked off CPU. Need choose successor.
  - Under 3: process may kick out another process from CPU.

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the CPU scheduler; steps:
  - Context switch.
  - Switch to user mode.
  - Jump to the proper location in the user code to restart that process.
- *Dispatch latency* – time it takes for the dispatcher to stop one process and restart another.
- CPU scheduler is (semi-automated) policy, dispatcher is pure mechanism.

Scheduling Criteria

- CPU utilization – keep the CPU as busy as possible. (Max)
  - Can *starve* I/O-bound jobs.
- Throughput – # of processes that complete their execution per time unit. (Max)
  - Can starve long jobs.
- Turnaround time – amount of time to execute a particular process from submission to completion. (Min)
  - Can appear unresponsive under time-sharing.
- Waiting time – amount of time a process has been waiting in the ready queue. (Min)
  - Some non-critical jobs don't mind waiting.
- Response time – amount of time it takes from when a request was submitted until the first response is computed and sent to I/O device. (Min)
  - Does not include I/O processing time.
  - Think "ls –R | more".
  - Can't distinguish debugging vs. real output.

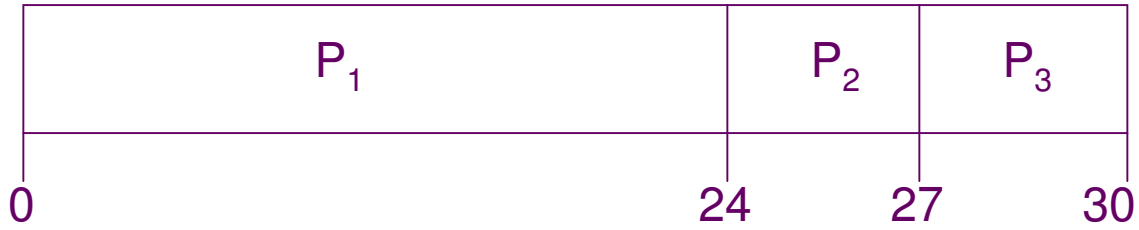
Optimality

- Optimize average measure.
- Optimize minimum or maximum, e.g. minimize max response time.
- Minimize variance (predictable system).
- Examples that follow:
  - Minimize average waiting time.
  - Assume single burst.
  - Assume context switch overhead = 0.

First-Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

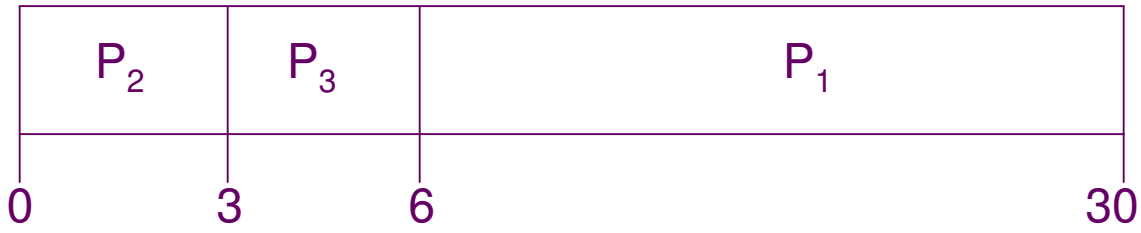
- Suppose that the processes arrive in the order: P_1 , P_2 , P_3. The Gantt Chart for the schedule is:



- 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.
- *Convoy effect*: short process behind long process like motorbikes behind a bus.
- If long process goes into infinite loop, “kill” won’t be able to stop it: if we preempt, stuck process still has priority over “kill”.

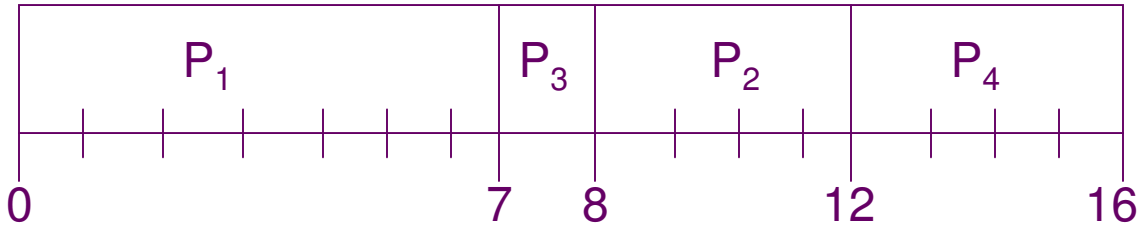
Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- Two schemes:
  - Nonpreemptive – once CPU given to the process it cannot be preempted until it completes its CPU burst.
  - Preemptive – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This is Shortest-Remaining-Time-First (SRTF) scheduling.
- 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	7
P_2	2	4
P_3	4	1
P_4	5	4

- SJF (non-preemptive):

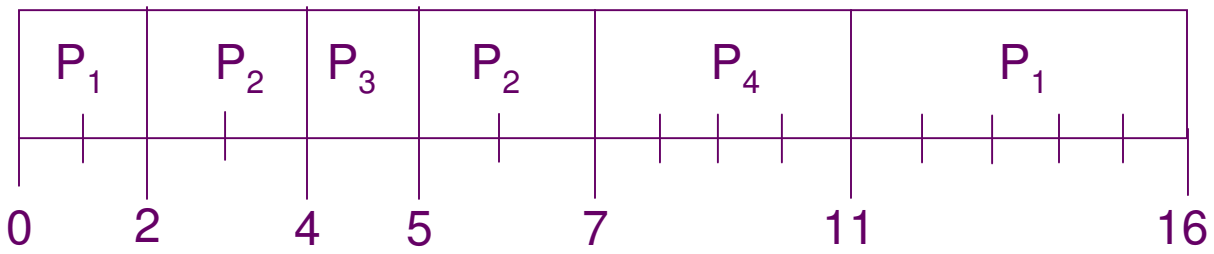


- Average waiting time = (0 + 6 + 3 + 7)/4 = 4.

Example of Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- SJF (preemptive):



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

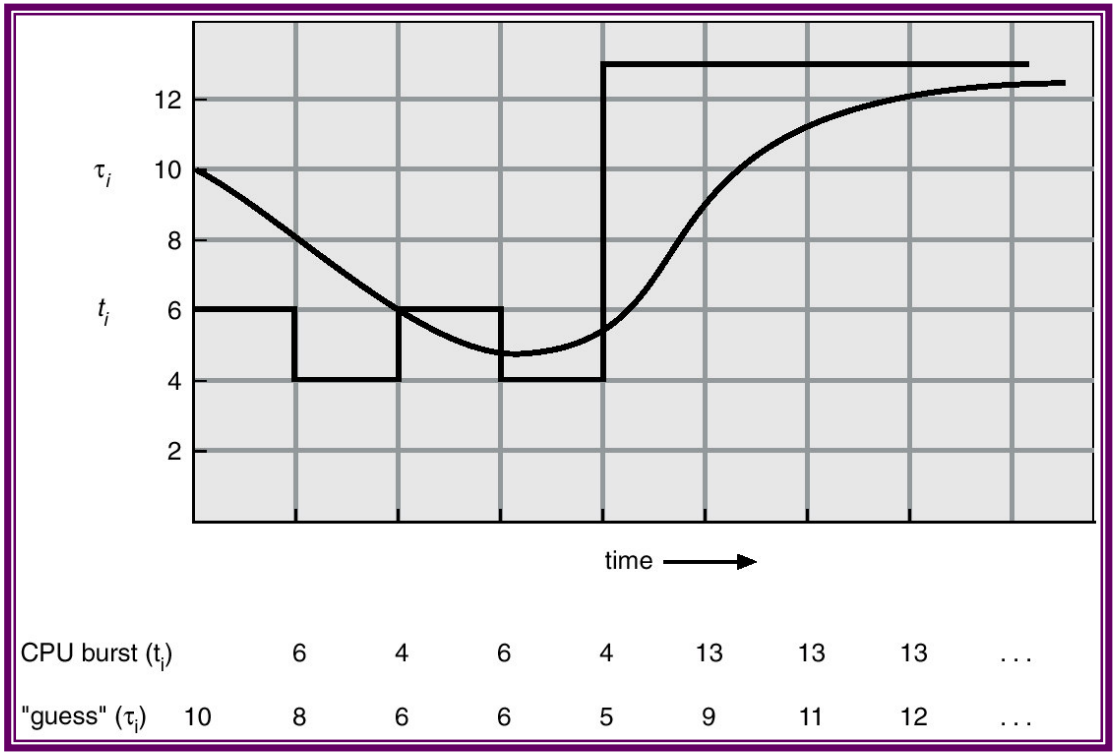
Determining Length of Next CPU Burst

- Can only *estimate* the length.
- Can be done by using the length of previous CPU bursts, using exponential averaging.

1. t_n = actual lenght of n^{th} CPU burst
2. τ_{n+1} = predicted value for the next CPU burst
3. $\alpha, 0 \le \alpha \le 1$
4. Define :

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

Past predictions; history.



Examples of Exponential Averaging

- $\alpha = 0$:
  - $\tau_{n+1} = \tau_n = \tau_0$.
  - Recent history does not count.
- $\alpha = 1$:
  - $\tau_{n+1} = t_n$.
  - Only the actual last CPU burst counts.
- If we expand the formula, we get:

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

Initial guess without past data, e.g. historical system average.

- If $\alpha > 0$ then $(1 - \alpha) < 1$, so each successive term has less weight than its predecessor.

Priority Scheduling

- A priority number (integer) is associated with each process: smallest integer means highest priority.
- The CPU is allocated to the process with the highest priority.
  - Preemptive.
  - Nonpreemptive.
- SJF is a priority scheduler: priority is the predicted next CPU burst time.
- Problem: starvation – low priority processes may never execute.
  - Solution: aging – as time progresses, increase the priority of waiting processes.

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

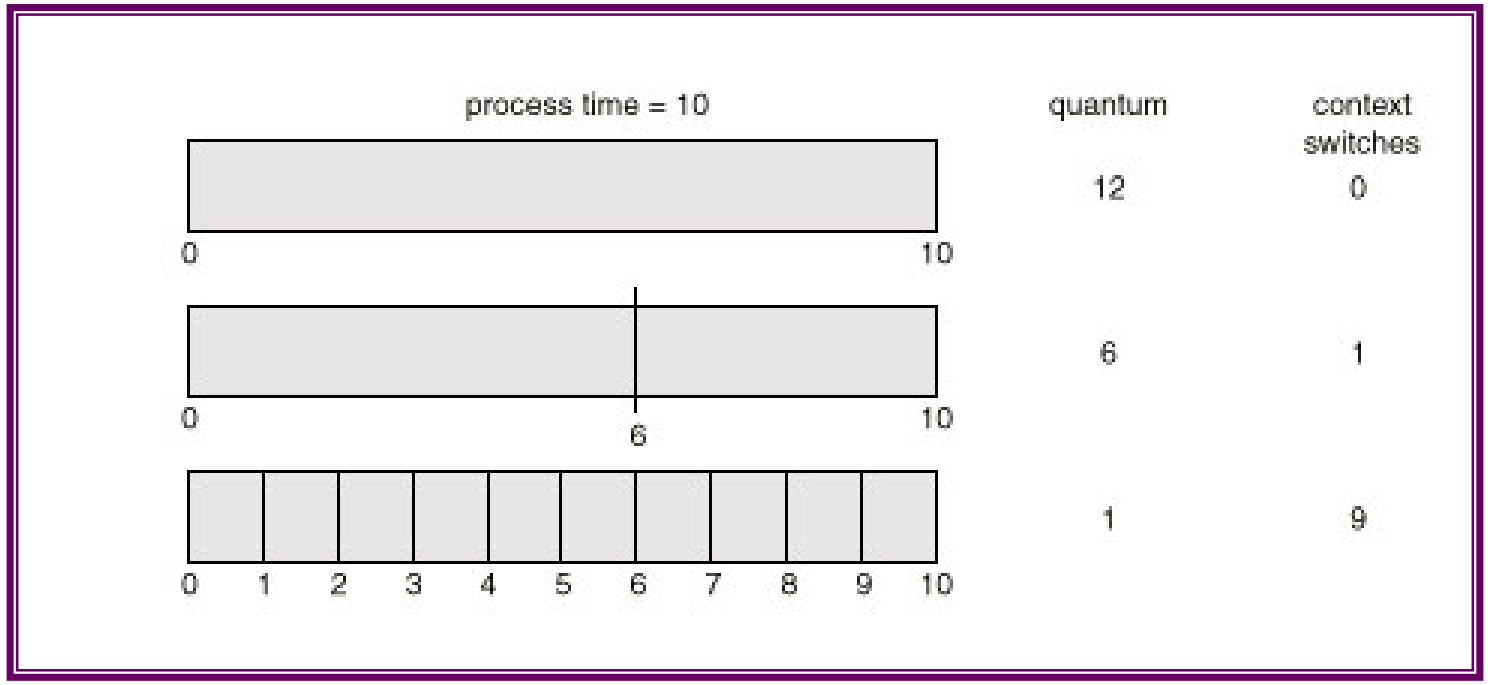
- Nonpreemptive:

- Average waiting time = (0 + 1 + 6 + 16 + 18)/5 = 8.2.

Round Robin (RR)

- Each process:
  - Gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds.
  - After this time has elapsed, the process is preempted and added to the tail 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 $(n\text{-}1)q$ time units.
- Performance:
  - q large $\Rightarrow$ FCFS (FIFO).
  - q small $\Rightarrow$ overhead is too high as q gets closer to context switch duration.

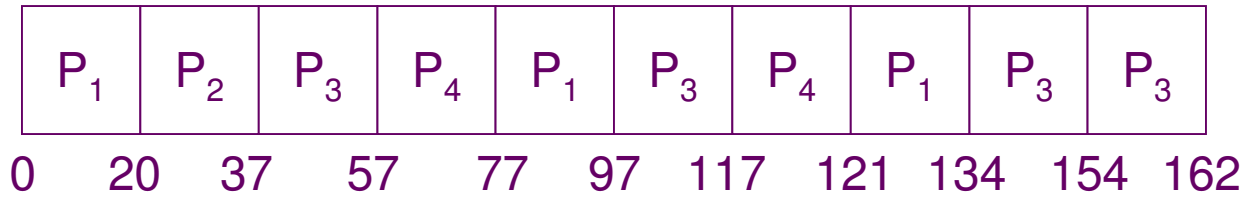
Time Quantum and Context Switch Time



Example of RR with Time Quantum = 20

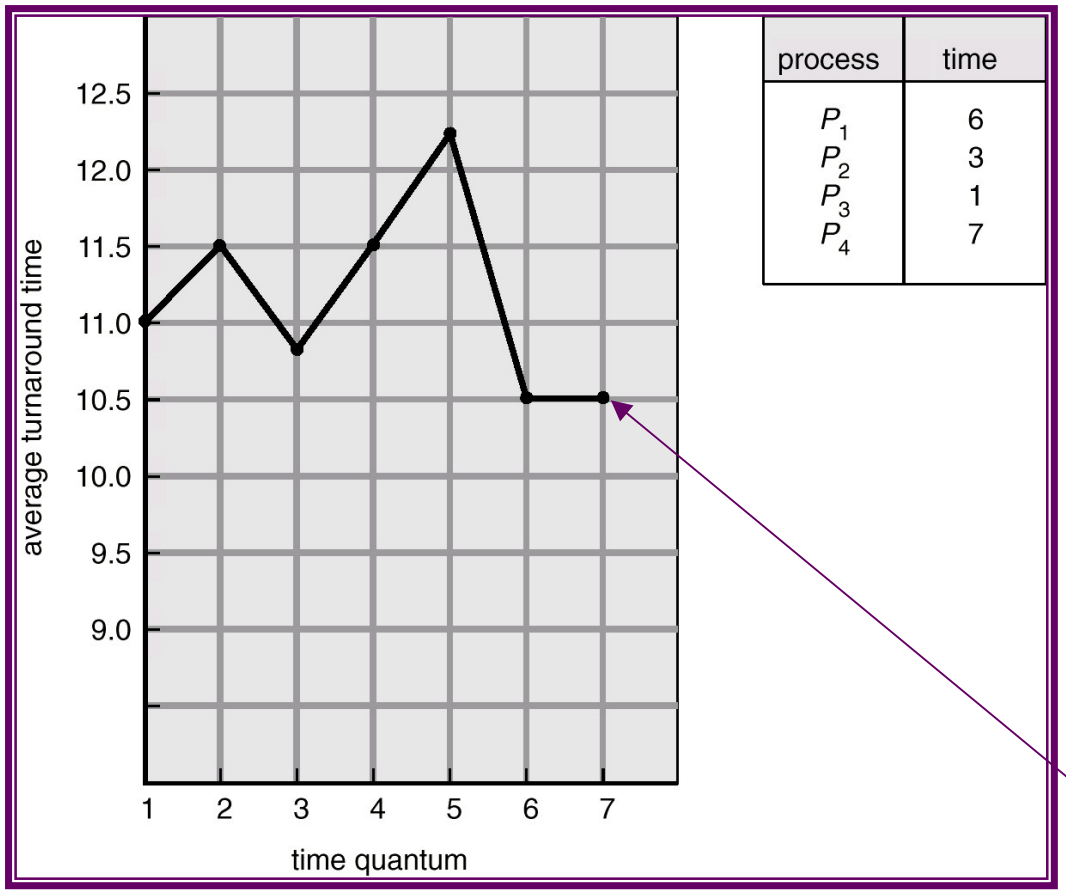
Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- The Gantt chart is:



- Typically better *response* than SJF (though higher average waiting, turnaround time).

Turnaround Time Varies With The Time Quantum



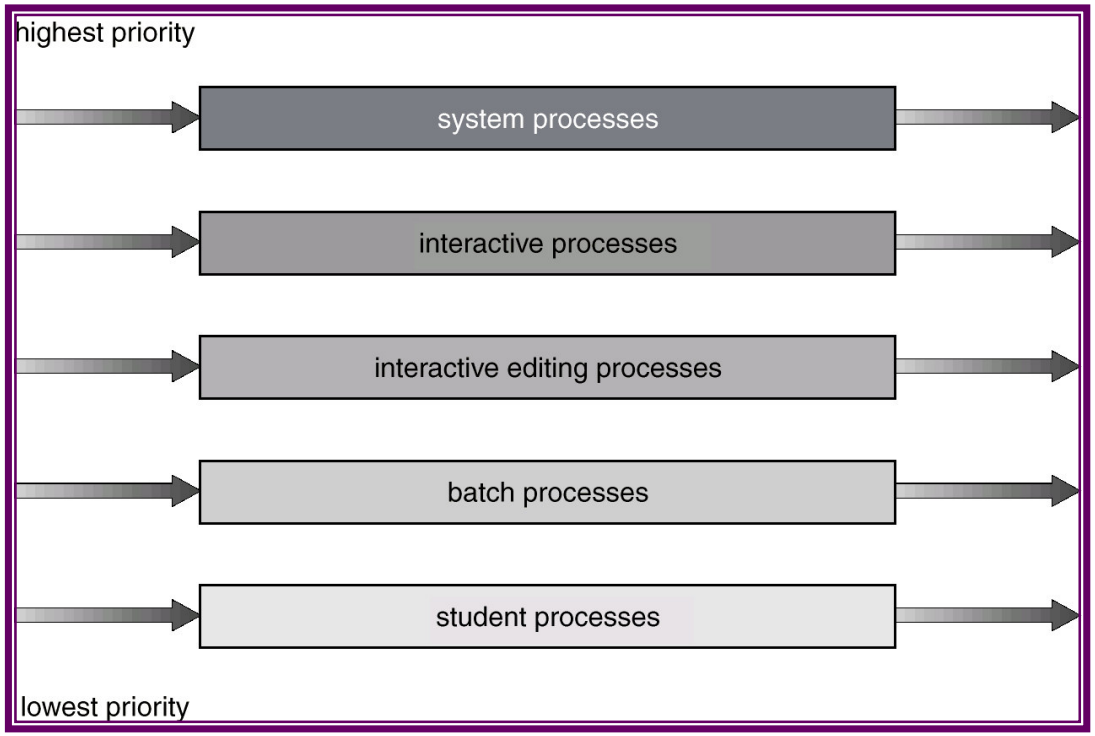
SJF: P_3 P_2 P_1 P_4
Average turnaround =
(1 +
1+3 +
1+3+6 +
1+3+6+7)/4 =8.

Becomes FCFS since max burst=7.

Multilevel Queue

- Ready queue is partitioned into separate queues:
  - Foreground (interactive) processes.
  - Background (batch) processes.
- Each queue has its own scheduling algorithm:
  - Foreground – RR.
  - Background – FCFS.
- Scheduling must be done *between* the queues:
  - 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.
  - Fixed priority scheduling; (i.e., serve all from foreground then from background). May starve background jobs so use aging and allow processes to move between queues.

Multilevel Queue Scheduling



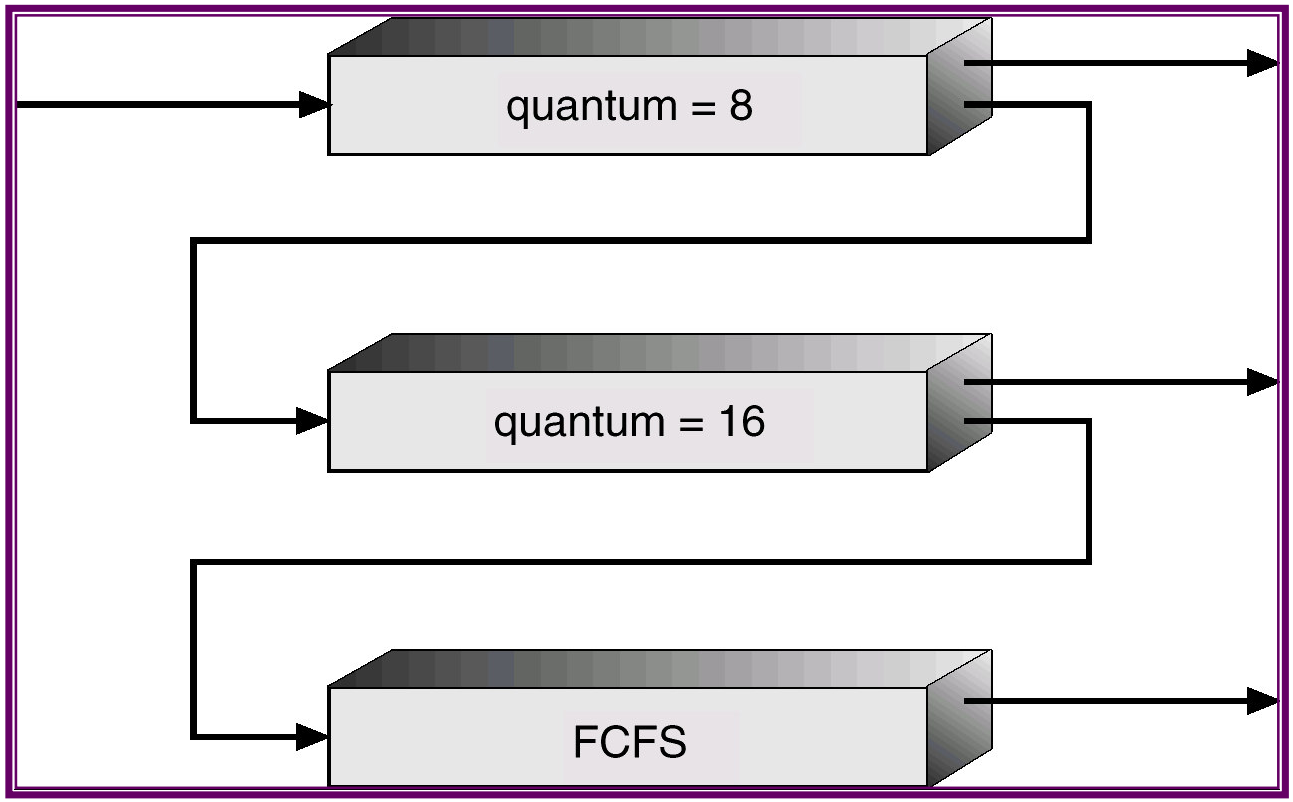
Multilevel Feedback Queue

- 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 time quantum 8 milliseconds.
  - Q_1 – RR time quantum 16 milliseconds.
  - Q_2 – FCFS.
- Scheduling:
  - A new job enters queue Q_0. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, job is preempted and moved to queue Q_1.
  - At Q_1 job is again served (eventually) and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue Q_2.
  - I/O-bound jobs return to Q_0 to finish short CPU-burst and return to waiting for new I/O.

Multilevel Feedback Queues



Multiple-Processor Scheduling

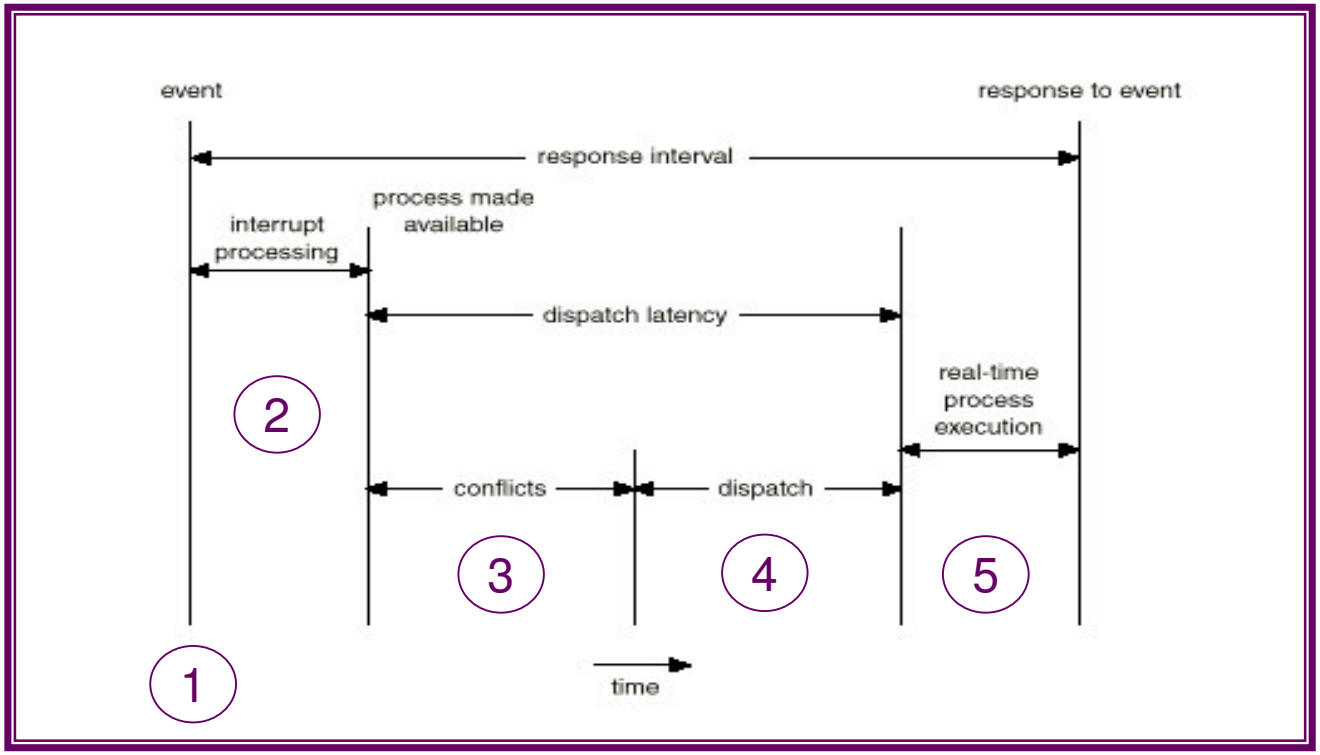
- CPU scheduling more complex when multiple CPUs are available.
- Processor types within a multiprocessor:
  - *Homogeneous:* all same architecture.
  - *Heterogeneous*: some processes incompatible with architecture of some CPUs.
- *Load balancing/sharing*: one ready queue for all processors, idle CPU assigned job at head of queue.
- *Asymmetric multiprocessing*:
  - Only one processor (master scheduler) accesses the system data structures, alleviating the need for protected access to shared data (if self-scheduling from common queue).
  - Easier, implemented first on new hardware.

Real-Time Scheduling

- *Hard real-time* systems:
  - Critical task must complete within a guaranteed time interval.
  - New process admitted with guarantee: *resource reservation*.
- *Soft real-time* computing:
  - Requires priority-like scheduling (e.g. multiple queues).
  - Critical processes receive priority over less fortunate ones.
  - Priority of real-time processes doesn't drop (no demotion).
  - Low dispatch latency.
- Low dispatch latency techniques:
  - Kernel must be preemptible (Solaris 2); often isn't, to keep system data structures safe from corruption (interrupt during partial modification).
  - If shared data is in-use by lower priority process, critical process must wait: *priority inversion*.
    - Solution: priority inheritance: low priority process gets critical process priority until it releases held resources.

Dispatch Latency

1. Hardware interrupt indicating critical event.
2. Basic interrupt handling: interrupt vector, service routine, identify critical process to handle event and get ready to run.
3. Preempt other processes, resolve priority inversion.
4. Dispatch critical process.
5. Critical process computes response to event, takes action.



Algorithm Evaluation

- Deterministic modeling:
  - Given particular predetermined workload, compute performance measure of each algorithm for that workload.
- Queueing models: statistics:
  - Scheduler is math function f() mapping process arrival & burst times to performance measure.
  - Given probability distribution of P, compute distribution of f(P): expected value, variance, etc.
- Simulation:
  - Application that behaves like hardware+OS but given process characteristics as input, not actual processes.
  - Extreme is virtual machine with real OS, i.e. near same effort as implementation (but no hardware glitches).
- Implementation.
- Assignment 1 is *coarse* simulation mix:
  - Input hand-specified as in deterministic modeling.
  - Performance evaluated by simulating system activities under given input.

Evaluation of CPU Schedulers by Simulation