CS 143A - Principles of Operating Systems

Lecture 4 - CPU Scheduling

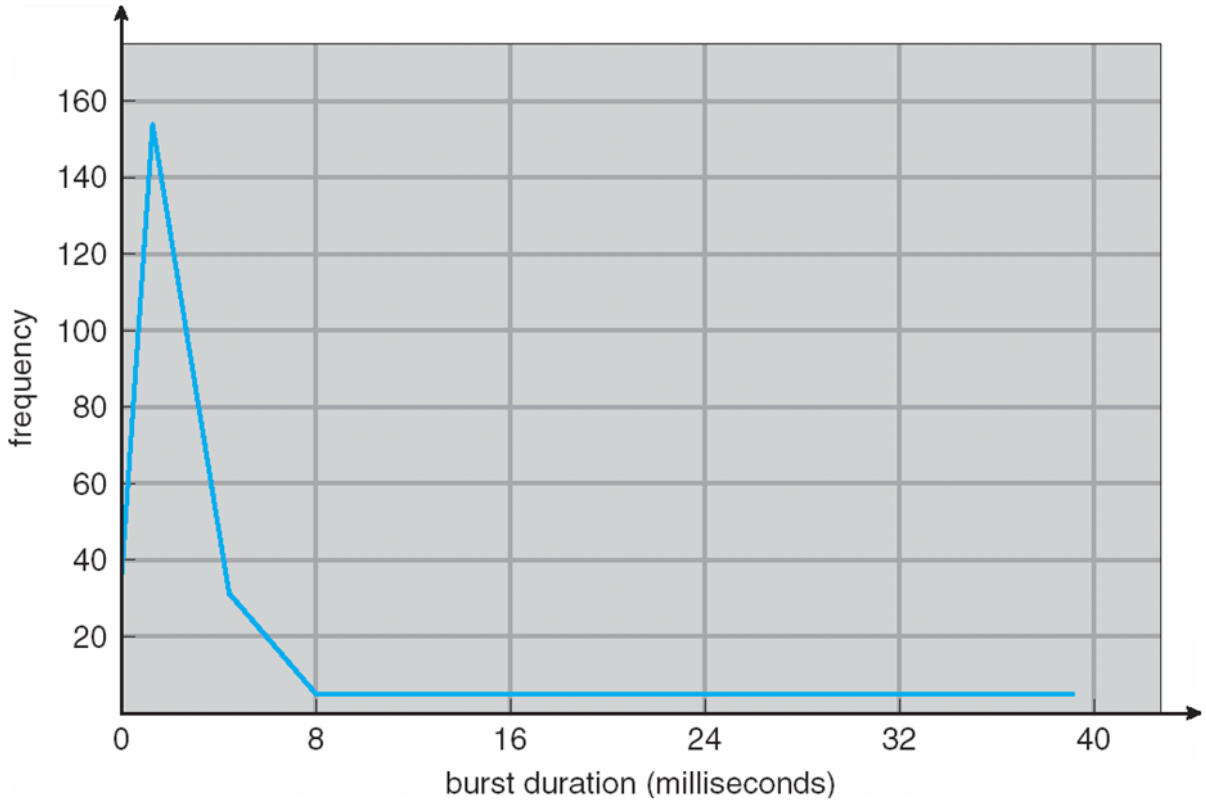
Prof. Nalini Venkatasubramanian

nalini@ics.uci.edu

Outline

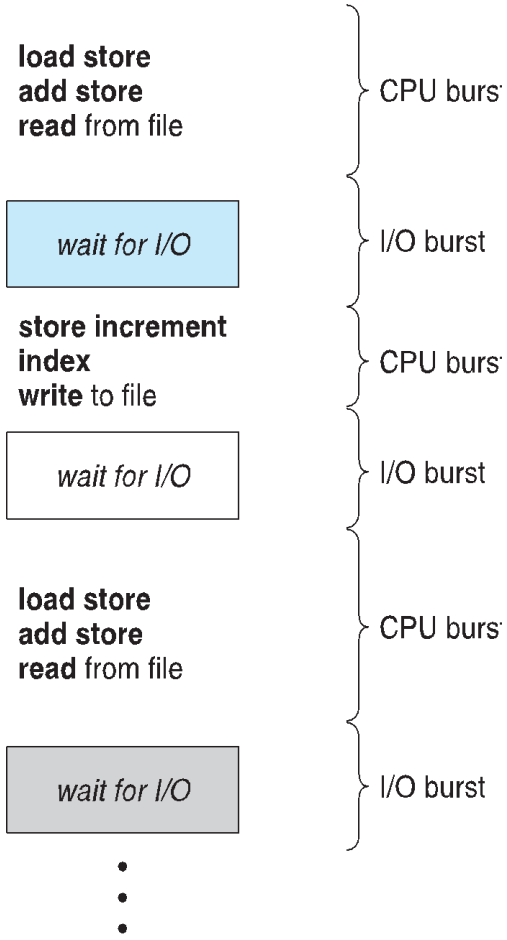
- Basic Concepts
- Scheduling Objectives
- Levels of Scheduling
- Scheduling Criteria
- Scheduling Algorithms
  - FCFS, Shortest Job First, Priority, Round Robin, Multilevel
- Multiple Processor Scheduling
- Real-time Scheduling
- Algorithm Evaluation

CPU Burst Distribution



Basic Concepts

- Maximum CPU utilization obtained with multiprogramming.
- CPU-I/O Burst Cycle
  - Process execution consists of a cycle of CPU execution and I/O wait.



Scheduling Objectives

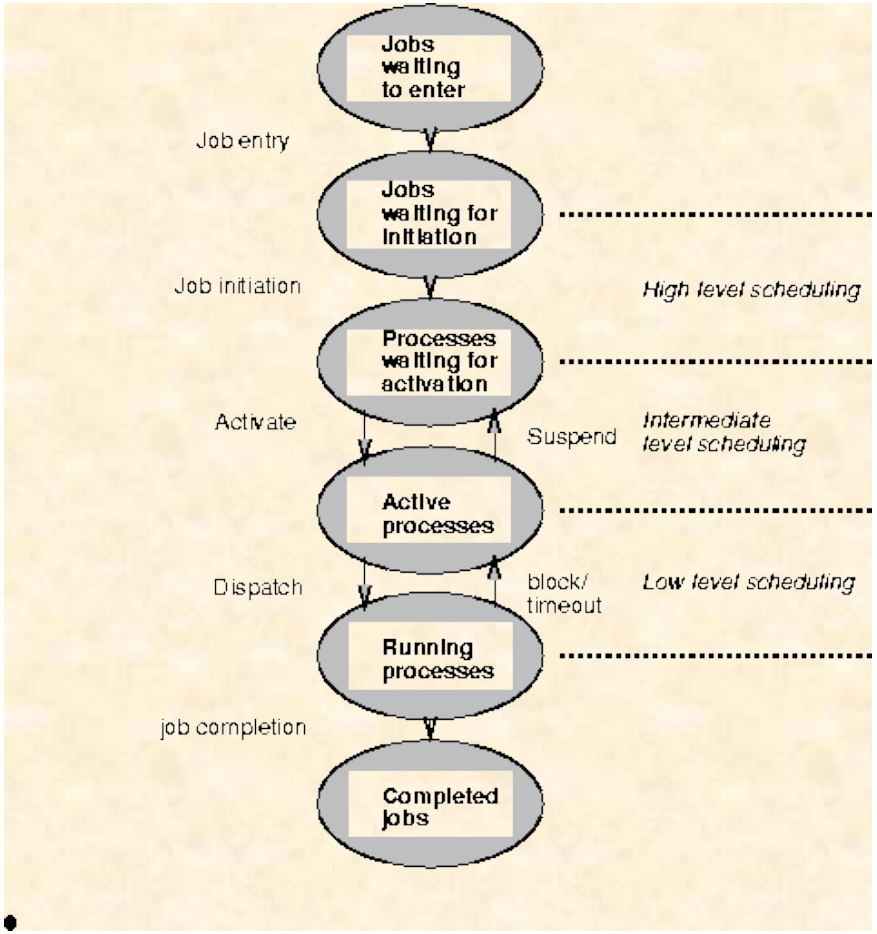
- Enforcement of fairness
  - in allocating resources to processes
- Enforcement of priorities
- Make best use of available system resources
- Give preference to processes holding key resources.
- Give preference to processes exhibiting good behavior.
- Degrade gracefully under heavy loads.

Program Behavior Issues

- I/O boundedness
  - short burst of CPU before blocking for I/O
- CPU boundedness
  - extensive use of CPU before blocking for I/O
- Urgency and Priorities
- Frequency of preemption
- Process execution time
- Time sharing
  - amount of execution time process has already received.

Levels of Scheduling

- High Level / Job Scheduling
  - Selects jobs allowed to compete for CPU and other system resources.
- Intermediate Level / Medium Term Scheduling
  - Selects which jobs to temporarily suspend/resume to smooth fluctuations in system load.
- Low Level (CPU) Scheduling or Dispatching
  - Selects the ready process that will be assigned the CPU.
  - Ready Queue contains PCBs of processes.



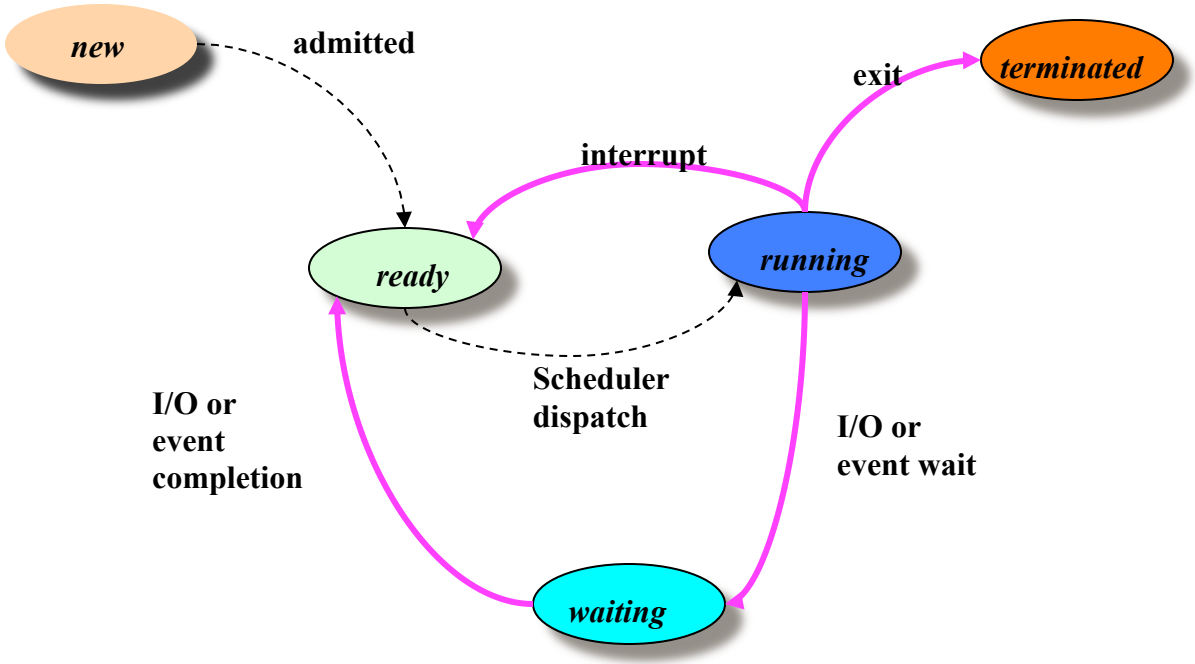
CPU Scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.
  - Non-preemptive Scheduling
    - Once CPU has been allocated to a process, the process keeps the CPU until
      - Process exits OR
      - Process switches to waiting state
  - Preemptive Scheduling
    - Process can be interrupted and must release the CPU.
      - Need to coordinate access to shared data

CPU Scheduling Decisions

- CPU scheduling decisions may take place when a process:
  1. switches from running state to waiting state
  2. switches from running state to ready state
  3. switches from waiting to ready
  4. terminates
- Scheduling under 1 and 4 is non-preemptive.
- All other scheduling is preemptive.

CPU scheduling decisions



Dispatcher

- OS 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 (continue) that program
- Dispatch Latency:
  - time it takes for the dispatcher to stop one process and start another running.
  - Dispatcher must be fast.

Scheduling Criteria

- CPU Utilization
  - Keep the CPU and other resources as busy as possible
- Throughput
  - # of processes that complete their execution per time unit.
- Turnaround time
  - amount of time to execute a particular process from its entry time.

Scheduling Criteria (cont.)

- Waiting time
  - amount of time a process has been waiting in the ready queue.
- Response Time (in a time-sharing environment)
  - amount of time it takes from when a request was submitted until the first response is produced, NOT output.

Optimization Criteria

- Optimize overall system
  - Max CPU Utilization
  - Max Throughput
- Optimize individual processes' performance
  - Min Turnaround time
  - Min Waiting time
  - Min Response time

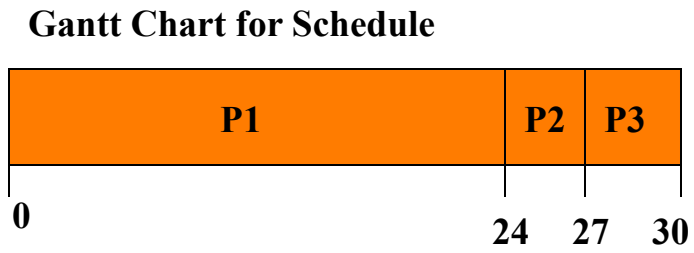
First Come First Serve (FCFS) Scheduling

- Policy: Process that requests the CPU *FIRST* is allocated the CPU *FIRST.*
  - FCFS is a non-preemptive scheduling algorithm.
- Implementation - using FIFO queues
  - incoming process is added to the tail of the queue.
  - Process selected for execution is taken from head of queue.
- Performance metric - Average waiting time in queue.
- Gantt Charts are used to visualize schedules.

First-Come, First-Served(FCFS) Scheduling

- Example

Process	Burst Time
P1	24
P2	3
P3	3

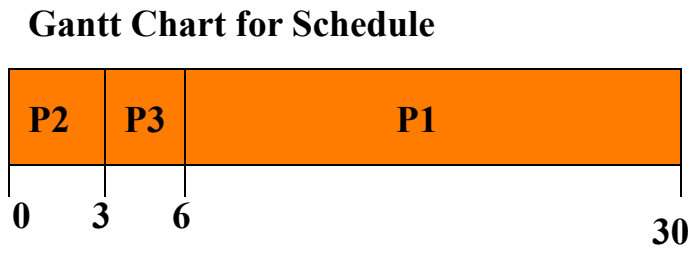


- Suppose the arrival order for the processes is
  - P1, P2, P3
- Waiting time
  - P1 = 0;
  - P2 = 24;
  - P3 = 27;
- Average waiting time
  - (0+24+27)/3 = 17

FCFS Scheduling (cont.)

- Example

Process	Burst Time
P1	24
P2	3
P3	3



- Suppose the arrival order for the processes is
  - P2, P3, P1
- Waiting time
  - P1 = 6; P2 = 0; P3 = 3;
- Average waiting time
  - (6+0+3)/3 = 3 , better..
- *Convoy Effect*:
  - short process behind long process, e.g. 1 CPU bound process, many I/O bound processes.

Shortest-Job-First (SJF) Scheduling

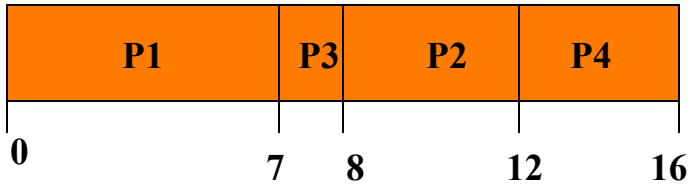
- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- Two Schemes:
  - Scheme 1: Non-preemptive
    - Once CPU is given to the process it cannot be preempted until it completes its CPU burst.
  - Scheme 2: Preemptive
    - If a new CPU process arrives with CPU burst length less than remaining time of current executing process, preempt. Also called Shortest-Remaining-Time-First (SRTF).
  - SJF is optimal - gives minimum average waiting time for a given set of processes.

Non-Preemptive SJF Scheduling

- Example

Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Gantt Chart for Schedule



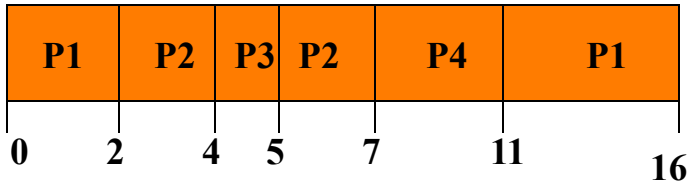
Average waiting time =
(0+6+3+7)/4 = 4

Preemptive SJF Scheduling (SRTF)

- Example

Process	Arrival Time	Burst Time
P1	0	7
P2	2	4
P3	4	1
P4	5	4

Gantt Chart for Schedule

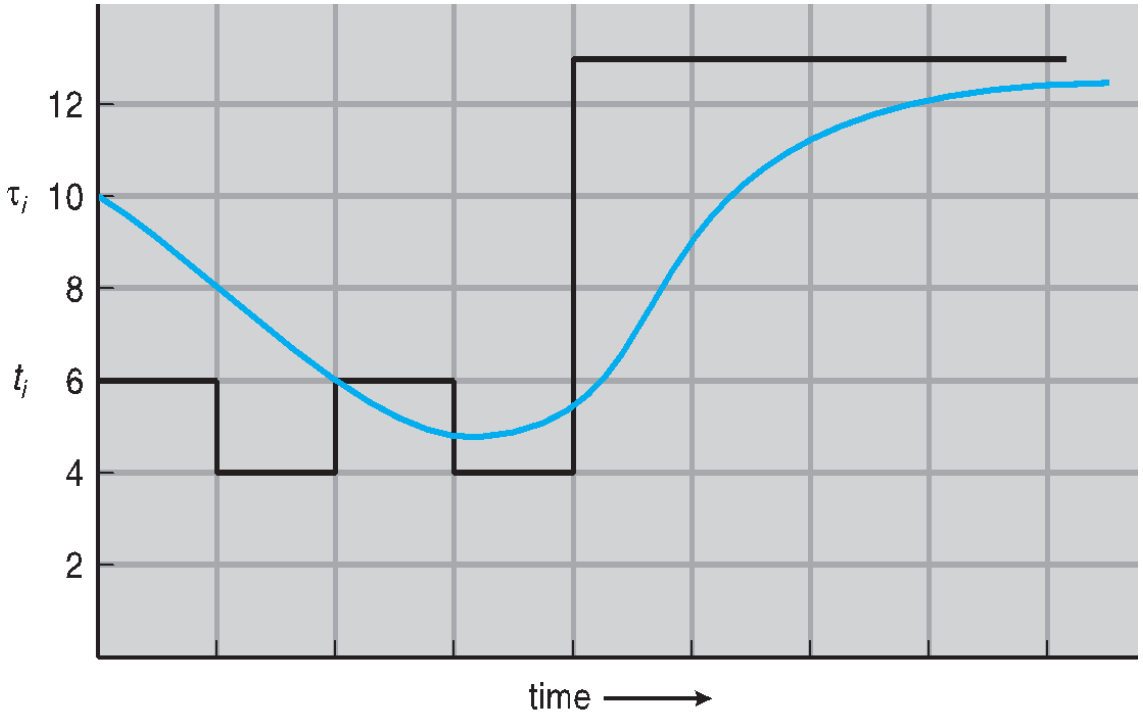


Average waiting time = (9+1+0+2)/4 = 3

Determining Length of Next CPU Burst

- One can only estimate the length of burst.
  - Halting problem: cannot (in general) determine if a program will run forever on some input or complete in finite time
- Estimate next CPU burst by an *exponentially-weighted moving average* over previous ones:
  - t_n = actual length of n^{th} burst
  - τ_{n+1} = predicted value for the next CPU burst
  - α = weight, $0 \leq \alpha \leq 1$
  - Define
    - $\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\, \tau_n$

Prediction of the length of the next CPU burst



CPU burst (t_i)	6	4	6	4	13	13	13	...
"guess" (τ_i)	10	8	6	6	5	9	11	12	...

Exponential Averaging(cont.)

- $\alpha = 0$
  - $\tau_{n+1} = \tau_n$; Recent history does not count
- $\alpha = 1$
  - $\tau_{n+1} = t_n$; Only the actual last CPU burst counts.
- Similarly, expanding the formula:
  - $\tau_{n+1} = \alpha t_n + (1-\alpha)\, \alpha t_{n-1} + \ldots +$
    $(1-\alpha)^j\, \alpha t_{n-j} + \ldots$
    $(1-\alpha)^{(n+1)}\, \tau_0$
    - Each successive term has less weight than its predecessor.

Priority Scheduling

- A priority value (integer) is associated with each process. Can be based on
  - Cost to user
  - Importance to user
  - Aging
  - %CPU time used in last X hours.
- CPU is allocated to process with the highest priority.
  - Preemptive
  - Non-preemptive

Priority Scheduling (cont.)

- SJF is a priority scheme where the priority is the predicted next CPU burst time.
  - Higher priority number → lower priority
  - E.g. priority 0 is higher priority than priority 1
- Problem
  - Starvation!! - Low priority processes may never execute.
- Solution
  - Aging - as time progresses increase the priority of the process.

Round Robin (RR)

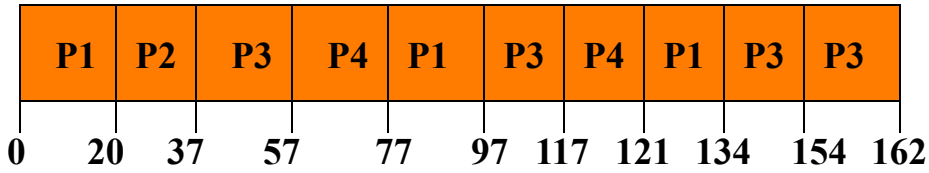
- Each process gets a small unit of CPU time
  - *Time quantum* usually 10-100 milliseconds.
  - After this time has elapsed, the process is preempted, moved to the end of the ready queue, and the process at the head of the ready queue is allocated the CPU.
- *n* processes, time quantum = *q*
  - Each process gets 1/*n* CPU time in chunks of at most *q* at a time.
  - No process waits more than (*n*-1)*q* time units.
  - Performance considerations:
    - Time slice *q* too large – FIFO (FCFS) behavior
    - Time slice *q* too small - Overhead of context switch is too expensive.
    - Heuristic - 70-80% of jobs block within time slice

Round Robin Example

- Time Quantum = 20

Process	Burst Time
P1	53
P2	17
P3	68
P4	24

Gantt Chart for Schedule

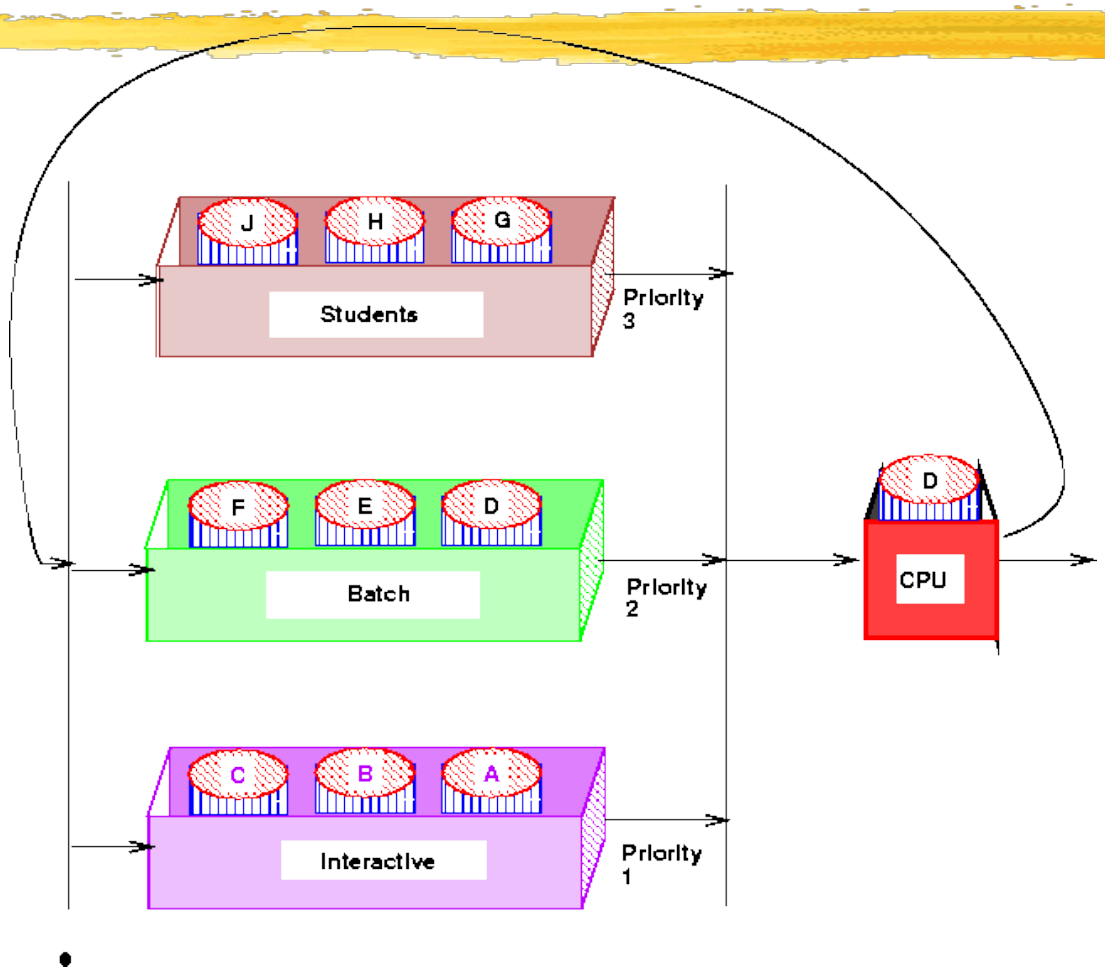


Typically, higher average turnaround time than SRTF, but better response time

Multilevel Queue

- Ready Queue partitioned into separate queues
  - Example: system processes, foreground (interactive), background (batch), student processes…
- Each queue has its own scheduling algorithm
  - Example: foreground (RR), background (FCFS)
- Processes assigned to one queue permanently.
- Scheduling must be done between the queues
  - Fixed priority - serve all from foreground, then from background. Possibility of starvation.
  - Time slice - Each queue gets some CPU time that it schedules - e.g. 80% foreground (RR), 20% background (FCFS)

Multilevel Queues



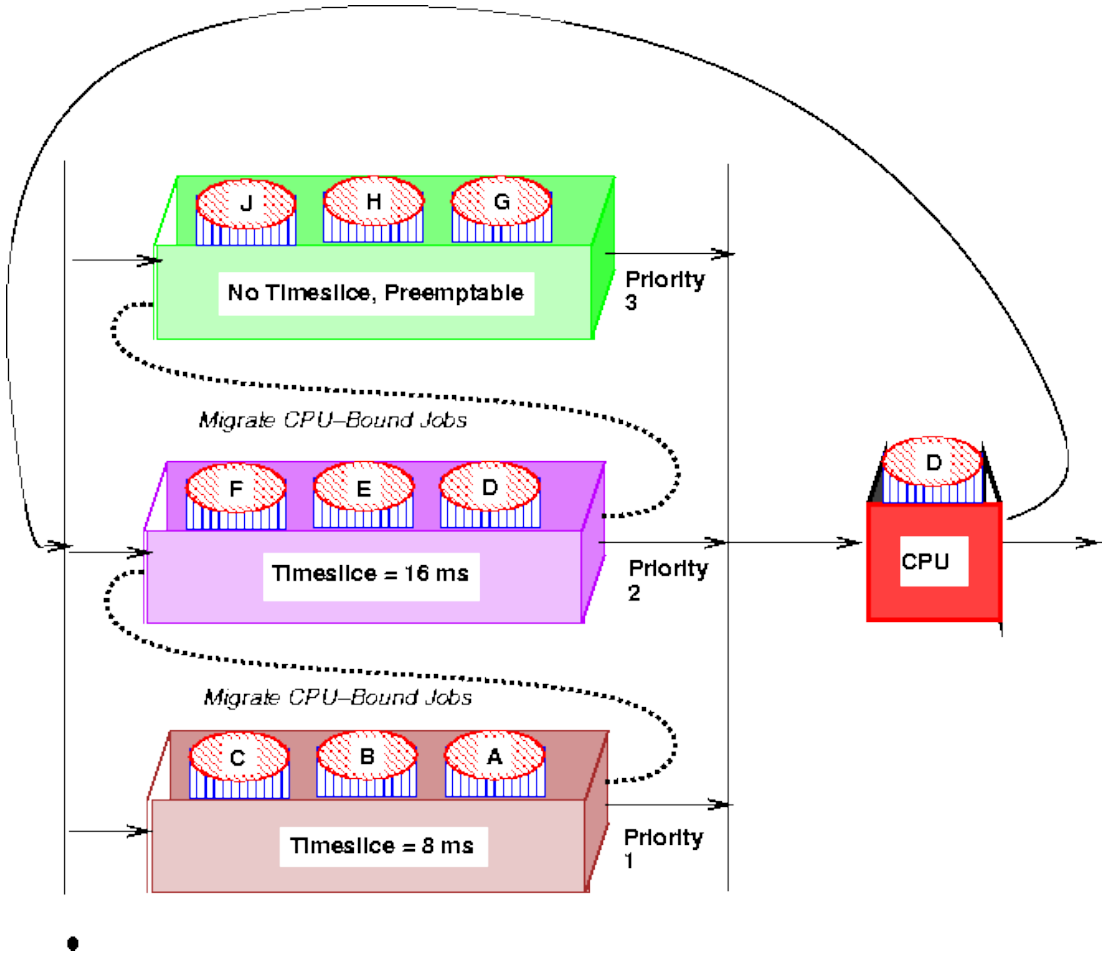
Multilevel Feedback Queue

- Multilevel Queue with priorities
- A process can *move* between the queues.
  - Aging can be implemented this way.
- Parameters for a multilevel feedback queue scheduler:
  - number of queues.
  - scheduling algorithm 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.

Multilevel Feedback Queues

- Example: Three Queues -
  - Q0 - time quantum 8 milliseconds (FCFS)
  - Q1 - time quantum 16 milliseconds (FCFS)
  - Q2 - FCFS
- Scheduling
  - New job enters Q0 - When it gains CPU, it receives 8 milliseconds. If job does not finish, move it to Q1.
  - At Q1, when job gains CPU, it receives 16 more milliseconds. If job does not complete, it is preempted and moved to Q2.

Multilevel Feedback Queues



Multiple-Processor Scheduling

- CPU scheduling becomes more complex when multiple CPUs are available.
  - Have one ready queue accessed by each CPU.
    - Self scheduled - each CPU dispatches a job from ready Q
    - Manager-worker - one CPU schedules the other CPUs
- Homogeneous processors within multiprocessor.
    - Permits Load Sharing
- Asymmetric multiprocessing
    - only 1 CPU runs kernel, others run user programs
    - alleviates need for data sharing

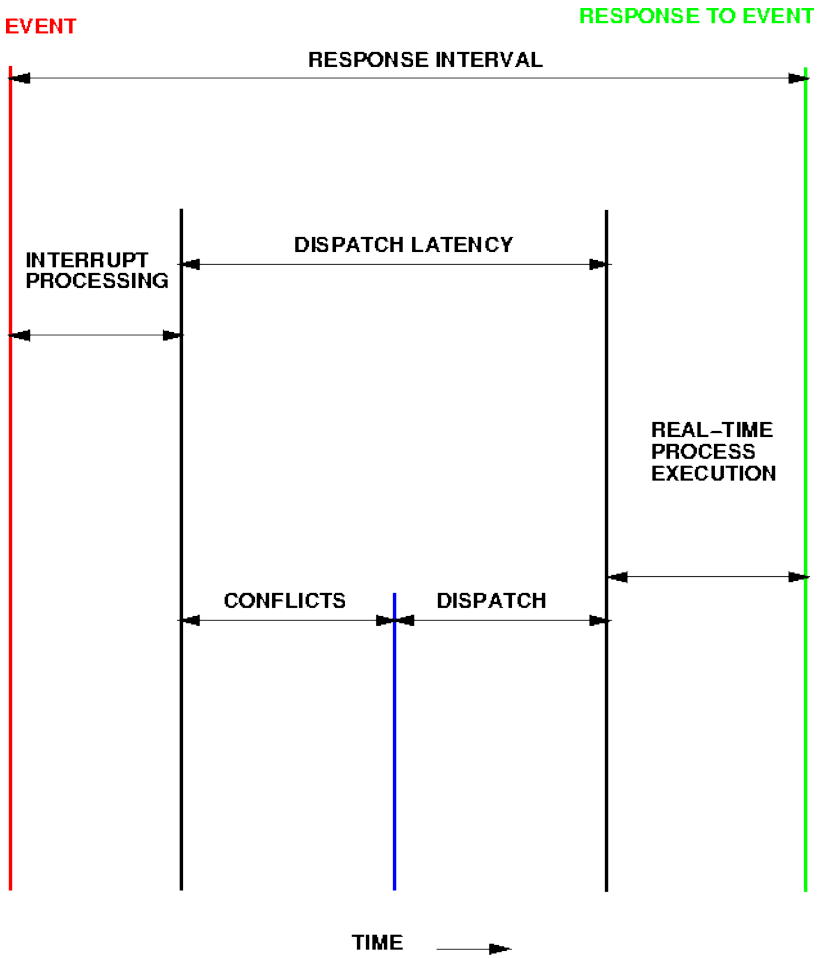
Real-Time Scheduling

- Hard Real-time Computing -
  - required to complete a critical task within a guaranteed amount of time.
- Soft Real-time Computing -
  - requires that critical processes receive priority over less important ones.
- Types of real-time Schedulers
  - Periodic Schedulers - Fixed Arrival Rate
  - Demand-Driven Schedulers - Variable Arrival Rate
  - Deadline Schedulers - Priority determined by deadline
  - ...

Issues in Real-time Scheduling

- Dispatch Latency
  - Problem - Need to keep dispatch latency small, OS may enforce process to wait for system call or I/O to complete.
  - Solution - Make system calls preemptible, determine safe criteria such that kernel can be interrupted.
- Priority Inversion and Inheritance
  - Problem: Priority Inversion
    - Higher Priority Process P0 needs kernel resource currently being used by another lower priority process P1.
    - P0 must wait for P1, but P1 isn't getting scheduled in CPU!
  - Solution: Priority Inheritance
    - Low priority process now inherits high priority until it has completed use of the resource in question.

Real-time Scheduling - Dispatch Latency



Scheduling Algorithm Evaluation

- Deterministic Modeling
  - Takes a particular predetermined workload and defines the performance of each algorithm for that workload.
  - Too specific, requires exact knowledge to be useful.
- Queuing Models and Queuing Theory
  - Use distributions of CPU and I/O bursts. Knowing *arrival* and *service* rates - can compute utilization, average queue length, average wait time, etc…
  - Little's formula: $n = \lambda \times W$
    - n is the average queue length, λ is the avg. arrival rate and W is the avg. waiting time in queue.
- Other techniques: Simulations, Implementation