ICS 143 - Principles of Operating Systems

Lecture 5 - CPU Scheduling

Prof. Nalini Venkatasubramanian
nalini@ics.uci.edu

Outline

- 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

Scheduling Objectives

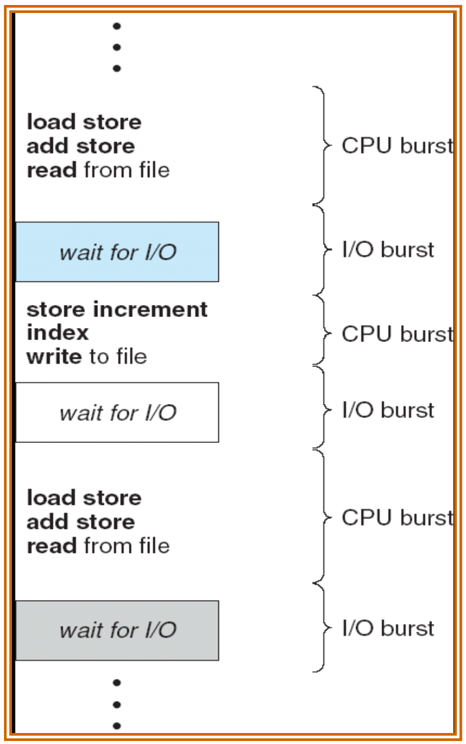
- Enforcement of fairness
    - in allocating resources to processes
- Enforcement of priorities
- Make best use of available system resources
- Give preference to processes holding key resources.
- Give preference to processes exhibiting good behavior.
- Degrade gracefully under heavy loads.

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.

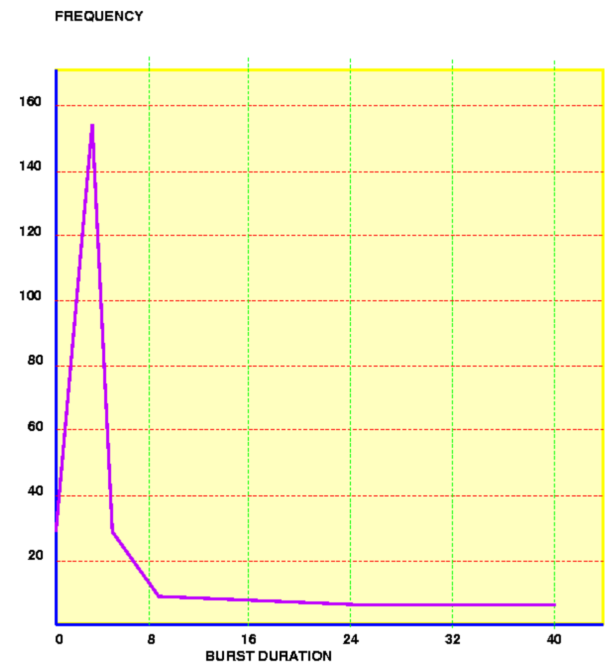
CPU and I/O Bursts

Maximum CPU utilization obtained with multiprogramming.



CPU-I/O Burst Cycle

Process execution consists of a cycle of CPU execution and a cycle of I/O wait.

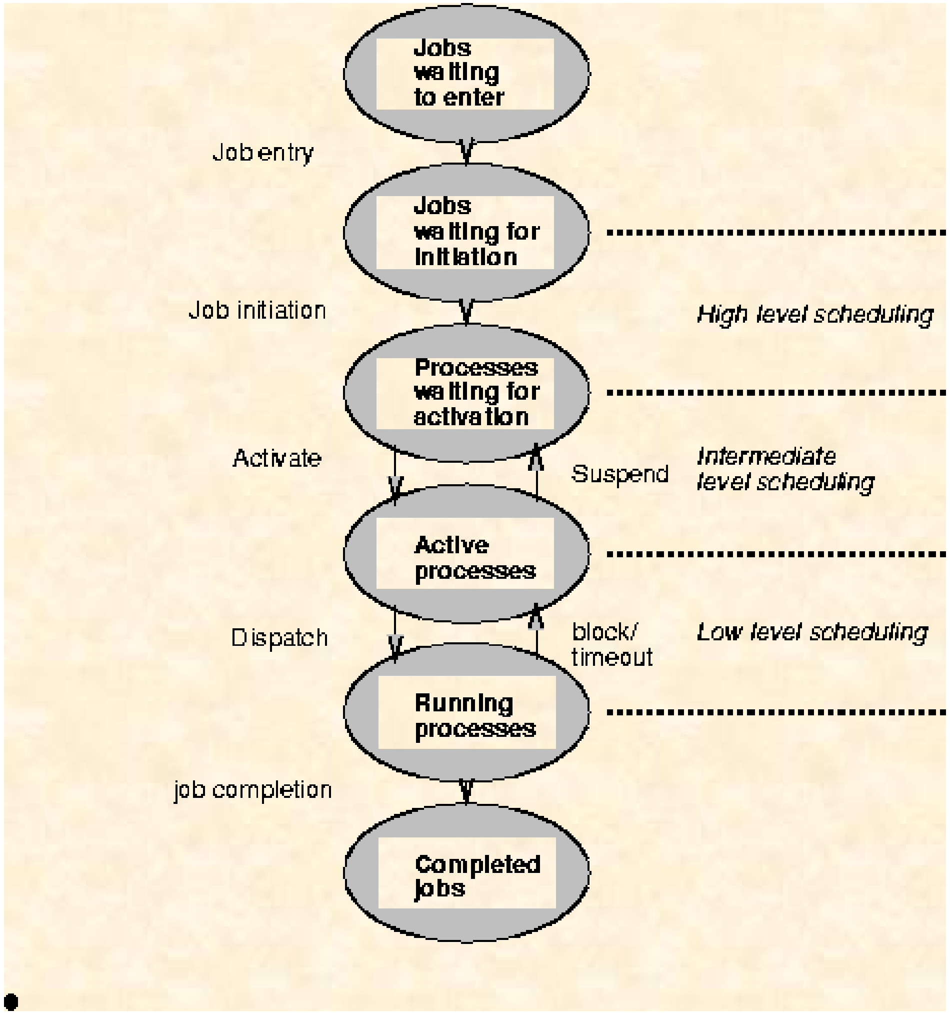


CPU Burst Distribution.

Levels of Scheduling

- High Level Scheduling or Job Scheduling
  - Selects jobs allowed to compete for CPU and other system resources.
- Intermediate Level Scheduling or 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.

Levels of Scheduling(cont.)



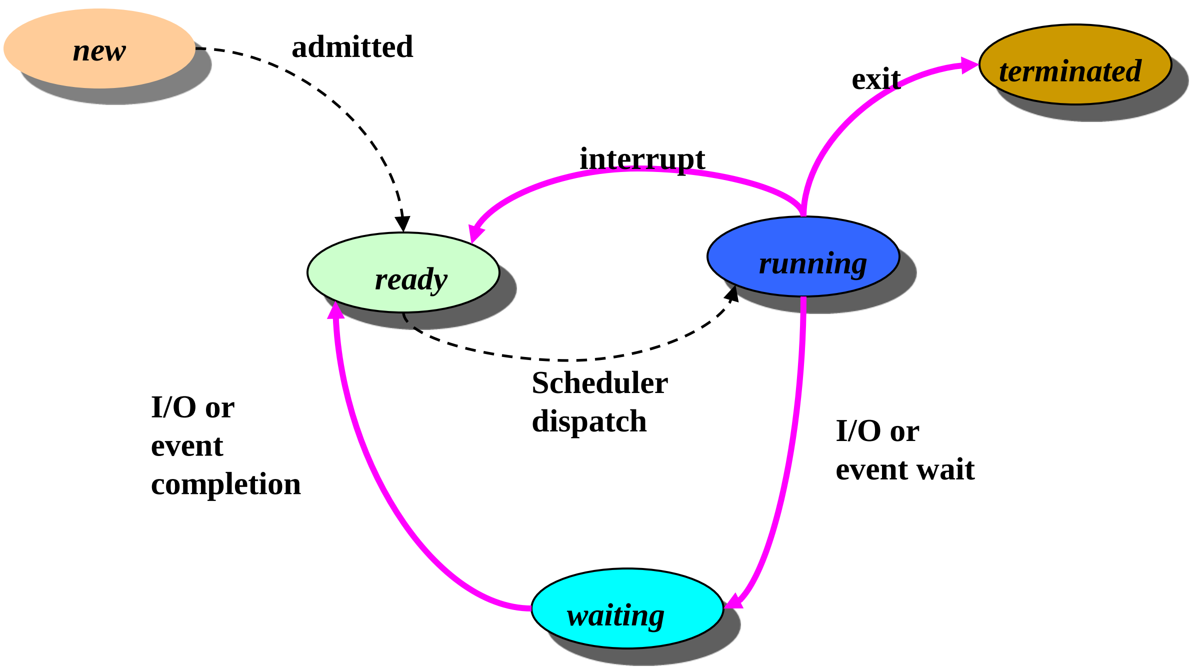
CPU Scheduler

- 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:
  - switches from running state to waiting state
  - switches from running state to ready state
  - switches from waiting to ready
  - terminates
- Scheduling under 1 and 4 is non-preemptive.
- All other scheduling is preemptive.

CPU scheduling decisions



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.
  - 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.
- 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

- Maximize CPU Utilization
- Maximize Throughput
- Minimize Turnaround time
- Minimize Waiting time
- Minimize response time

Observations: Scheduling Criteria

- Throughput vs. response time
  - Throughput related to response time, but not identical:
    - Minimizing response time will lead to more context switching than if you only maximized throughput
  - Two parts to maximizing throughput
    - Minimize overhead (for example, context-switching)
    - Efficient use of resources (CPU, disk, memory, etc)
- Fairness vs. response time
  - Share CPU among users in some equitable way
  - Fairness is not minimizing average response time:
    - Better *average* response time by making system *less* fair

Scheduling Policies

- First-Come First-Serve (FCFS)
- Shortest Job First (SJF)
  - Non-preemptive
  - Pre-emptive
- Priority
- Round-Robin
- Multilevel Queue
- Multilevel Feedback Queue
- Real-time Scheduling

First Come First Serve (FCFS) Scheduling

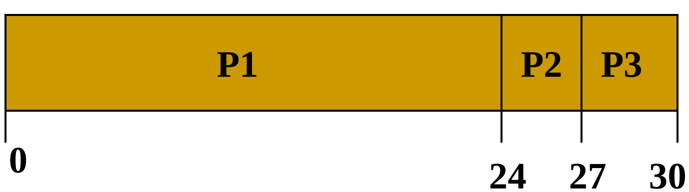
- Policy: Process that requests the CPU *FIRST* is allocated the CPU *FIRST.*
  - FCFS is a non-preemptive algorithm.
- Implementation - using FIFO queues
  - incoming process is added to the tail of the queue.
  - Process selected for execution is taken from head of queue.
- Performance metric - Average 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

Gantt Chart for Schedule



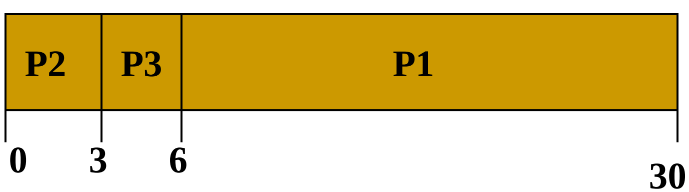
- Suppose the arrival order for the processes is
  - P1, P2, P3
- Waiting time
  - P1 = 0;
  - P2 = 24;
  - P3 = 27;
- Average waiting time
  - (0+24+27)/3 = 17
- Average completion time
  - (24+27+30)/3 = 27

FCFS Scheduling (cont.)

- Example

Process	Burst Time
P1	24
P2	3
P3	3

Gantt Chart for Schedule



- 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..
- Average waiting time
  - (3+6+30)/3 = 13 , better..
- *Convoy Effect*:
  - short process behind long process, e.g. 1 CPU bound process, many I/O bound processes.

Shortest-Job-First(SJF) Scheduling

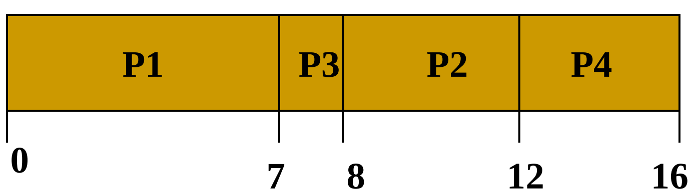
- Associate with each process the length of its next CPU burst.
- Use these lengths to schedule the process with the shortest time.
- Two Schemes:
  - Scheme 1: Non-preemptive
    - Once CPU is given to the process it cannot be preempted until it completes its CPU burst.
  - Scheme 2: Preemptive
    - 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 and SRTF (Example)

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

Non-Preemptive SJF Scheduling

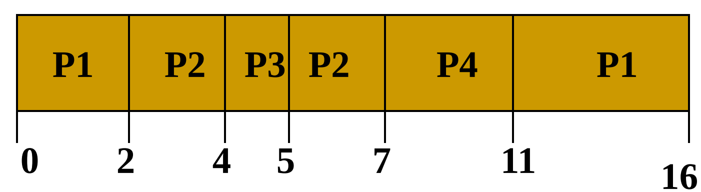
Gantt Chart for Schedule



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

Preemptive SJF Scheduling

Gantt Chart for Schedule



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

SJF/SRTF Discussion

- SJF/SRTF are the best you can do at minimizing average response time
  - Provably optimal (SJF among non-preemptive, SRTF among preemptive)
  - Since SRTF is always at least as good as SJF, focus on SRTF
- Comparison of SRTF with FCFS and RR
  - What if all jobs the same length?
    - SRTF becomes the same as FCFS (i.e. FCFS is best can do if all jobs the same length)
  - What if jobs have varying length?
    - SRTF (and RR): short jobs not stuck behind long ones
- Starvation
  - SRTF can lead to starvation if many small jobs!
  - Large jobs never get to run

SRTF Further discussion

- Somehow need to predict future
  - How can we do this?
  - Some systems ask the user
    - When you submit a job, have to say how long it will take
    - To stop cheating, system kills job if takes too long
  - But: Even non-malicious users have trouble predicting runtime of their jobs
- Bottom line, can't really know how long job will take
  - However, can use SRTF as a yardstick for measuring other policies
  - Optimal, so can't do any better
- SRTF Pros & Cons
  - Optimal (average response time) (+)
  - Hard to predict future (-)
  - Unfair (-)

Determining Length of Next CPU Burst

- One can only estimate the length of burst.
- Use the length of previous CPU bursts and perform exponential averaging.
  - t_n = actual length of nth burst
  - τ_{n+1} =predicted value for the next CPU burst
  - $\alpha = 0,\ \ 0 \le \alpha \le 1$
  - Define
    - $\tau_{n+1} = \alpha\, t_n + (1-\alpha)\,\tau_n$

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

    $(1-\alpha)$^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
  - Nonpreemptive

Priority Scheduling (cont.)

- SJN is a priority scheme where the 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 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 and added to the end of the ready queue.
- *n* processes, time quantum = *q*
  - Each process gets 1/*n* CPU time in chunks of at most *q* time units at a time.
  - No process waits more than (*n*-1)*q* time units.
  - Performance
    - Time slice *q* too large – response time poor
    - Time slice (∞)? -- reduces to FIFO behavior
    - Time slice *q* too small - Overhead of context switch is too expensive. Throughput poor

Example of RR with Time Quantum = 20

- Example:

Process	Burst Time
P_1	53
P_2	8
P_3	68
P_4	24

- The Gantt chart is:

P_1	P_2	P_3	P_4	P_1	P_3	P_4	P_1	P_3	P_3

0 20 28 48 68 88 108 112 125 145 153

- Waiting time
  - P_1=(68-20)+(112-88)=72
  - P_2=(20-0)=20
  - P_3=(28-0)+(88-48)+(125-108)=85
  - P_4=(48-0)+(108-68)=88
- Average waiting time = (72+20+85+88)/4=66¼
- Average completion time = (125+28+153+112)/4 = 104½

- Thus, Round-Robin Pros and Cons:
  - Better for short jobs, Fair (+)
  - Context-switching time adds up for long jobs (-)

Comparisons between FCFS and Round Robin

- Assuming zero-cost context-switching time, is RR always better than FCFS?
- Simple example:

  10 jobs, each take 100s of CPU time
  RR scheduler quantum of 1s
  All jobs start at the same time

- Completion Times:

Job #	FIFO	RR
1	100	991
2	200	992
…	…	…
9	900	999
10	1000	1000

  - Both RR and FCFS finish at the same time
  - Average response time is much worse under RR!
    - Bad when all jobs same length
- Also: Cache state must be shared between all jobs with RR but can be devoted to each job with FIFO
  - Total time for RR longer even for zero-cost switch!

Earlier Example with Different Time Quantum

Best FCFS:

P_2 [8]	P_4 [24]	P_1 [53]	P_3 [68]
0–8	8–32	32–85	85–153

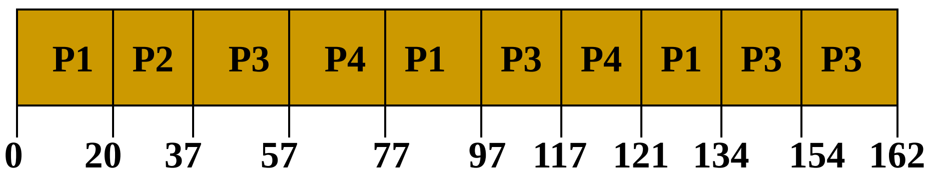
	Quantum	P_1	P_2	P_3	P_4	Average
Wait Time	Best FCFS	32	0	85	8	31¼
	Q = 1	84	22	85	57	62
	Q = 5	82	20	85	58	61¼
	Q = 8	80	8	85	56	57¼
	Q = 10	82	10	85	68	61¼
	Q = 20	72	20	85	88	66¼
	Worst FCFS	68	145	0	121	83½
Completion Time	Best FCFS	85	8	153	32	69½
	Q = 1	137	30	153	81	100½
	Q = 5	135	28	153	82	99½
	Q = 8	133	16	153	80	95½
	Q = 10	135	18	153	92	99½
	Q = 20	125	28	153	112	104½
	Worst FCFS	121	153	68	145	121¾

Round Robin Example

Time Quantum = 20

Process	Burst Time
P1	53
P2	17
P3	68
P4	24

Gantt Chart for Schedule



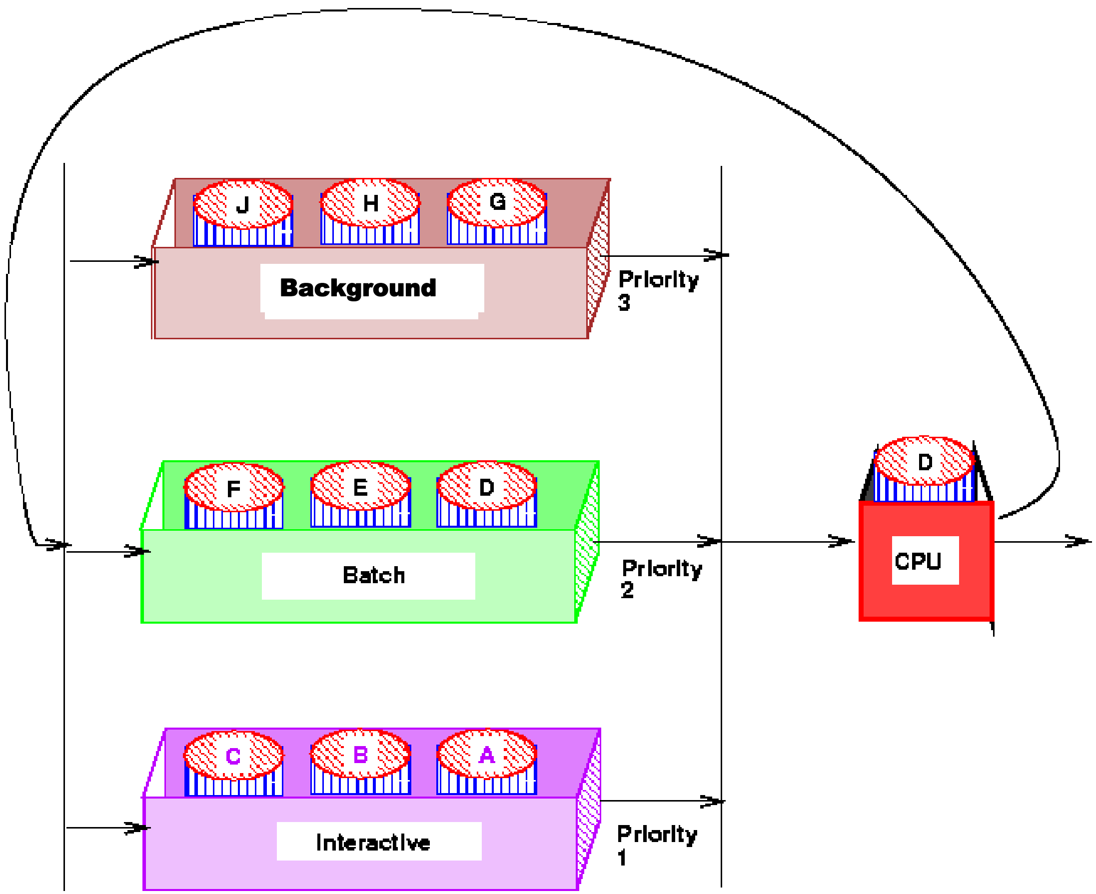
Typically, higher average turnaround time than SRTF, but better response

- Initially, UNIX timeslice (q) = 1 sec
  - Worked OK when UNIX was used by few (1-2) people.
  - What if three compilations going on? 3 seconds to echo each keystroke!
- In practice, need to balance short-job performance and long-job throughput
  - q must be large wrt context switch, o/w overhead is too high
  - Typical time slice today is between 10ms – 100ms
  - Typical context switching overhead is 0.1 – 1 ms
  - Roughly 1% overhead due to context switching
- Another Heuristic - 70 – 80% of jobs block within timeslice

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



-

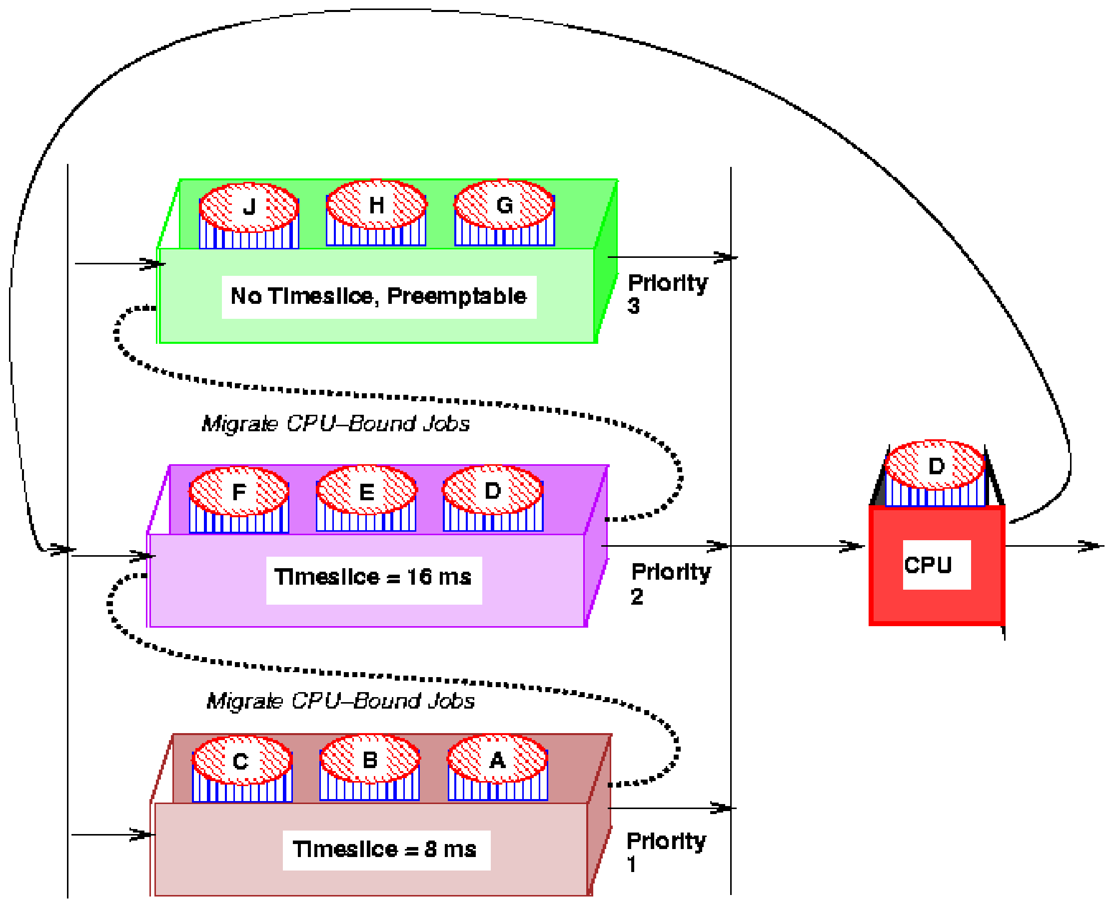
Multilevel Feedback Queue

- 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 (RR)
  - Q1 - time quantum 16 milliseconds (RR)
  - 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 queue 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
    - Master-Slave - 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

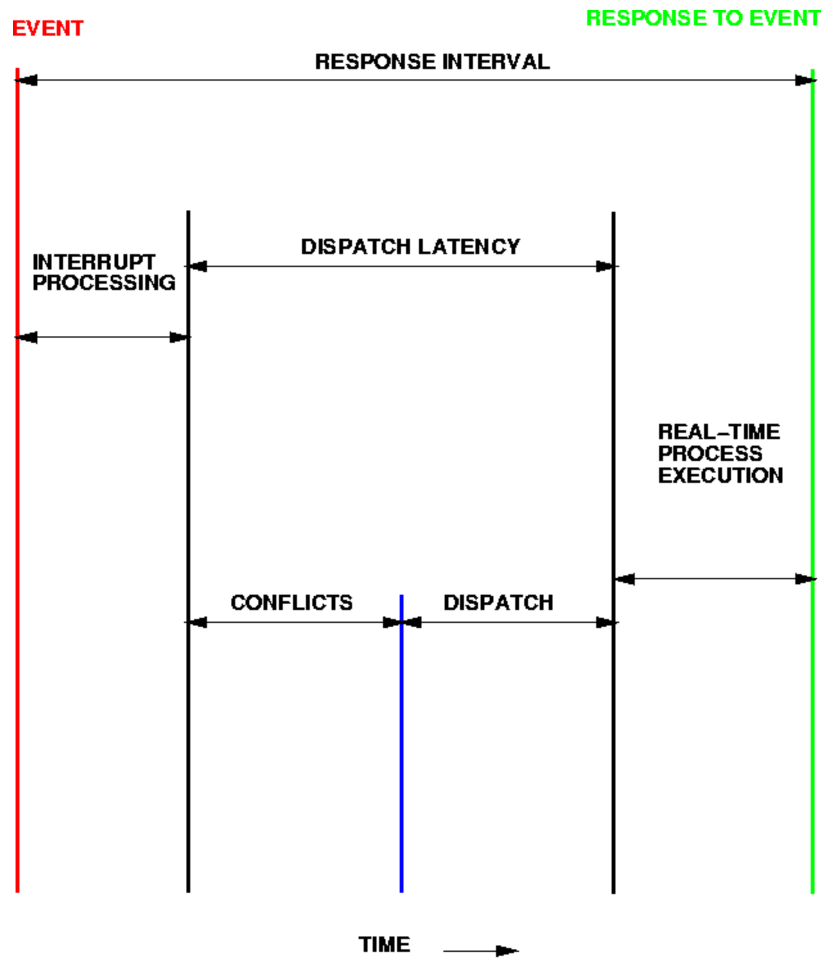
Real-Time Scheduling

- 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 fortunate 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 needs kernel resource currently being used by another lower priority process..higher priority process must wait.
  - 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



-

Additional scheduling techniques:

Lottery Scheduling

Proportional Share Scheduling

Energy efficient task scheduling:

Communication aware task scheduling:

Examples of real-time scheduling algorithms:

Rate monotonic (RM). Tasks are periodic. Policy is shortest-period-first, so it always runs the ready task with shortest period.

Earliest deadline (EDF). This algorithm schedules the task with closer deadline first.

Least laxity (LLF). lesser "flexibility" to be scheduled in time. Laxity is the difference between the time to deadline and the remaining computation time to finish.

Maximum-urgency-first (MUF). mixed-scheduling algorithm combines the best features of the others: Provides predictability under overload conditions and a scheduling bound of 100 percent for its critical set. The static part of a task's *urgency* is a user-defined *criticality* (high/ low), which has higher precedence than its dynamic part.

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 - $n = \lambda \times W$ where 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