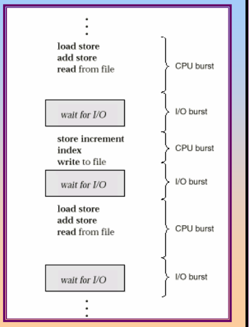
CPU Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
  - Batch systems
  - Interactive systems

Based on original slides by
Silberschatz, Galvin and Gagne

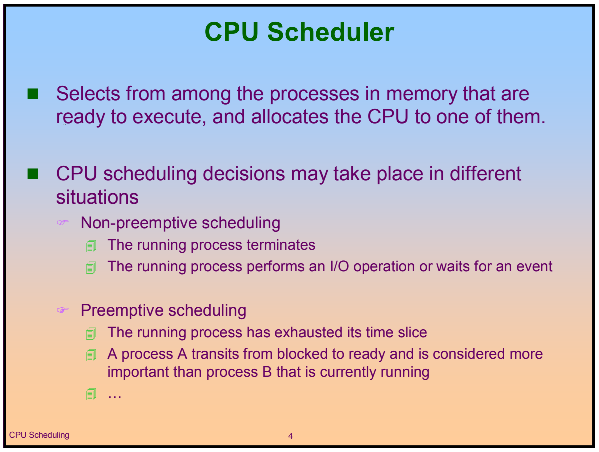
Basic Concepts

- CPU–I/O Burst Cycle
  - Process execution consists of a *cycle* of CPU execution and I/O wait.
  - CPU-Bound Processes
  - I/O-Bound Processes



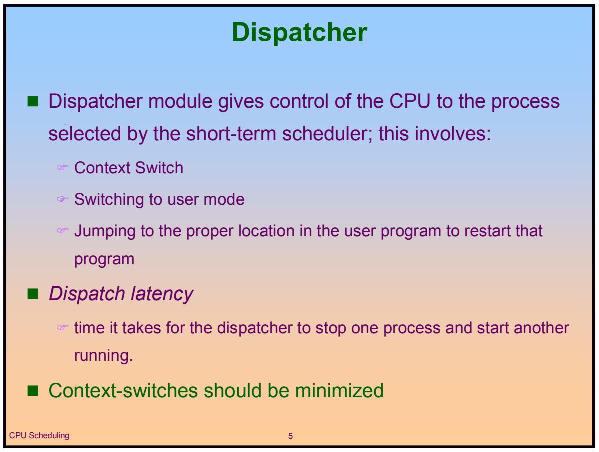
Basic Concepts (2)

- Maximum CPU utilization obtained with multiprogramming
  - Different part of the systems can be active simultaneously allowing parallel execution of processes
  - The scheduling algorithm should mix appropriately CPU-bound and I/O-Bound Processes



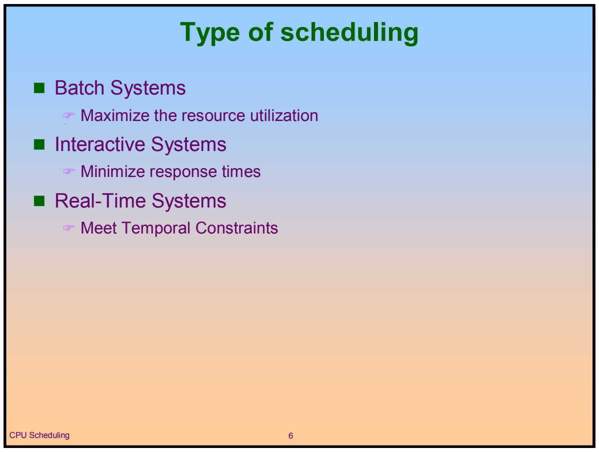
CPU Scheduler

- 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 in different situations
  - Non-preemptive scheduling
    - The running process terminates
    - The running process performs an I/O operation or waits for an event
  - Preemptive scheduling
    - The running process has exhausted its time slice
    - A process A transits from blocked to ready and is considered more important than process B that is currently running
    - …



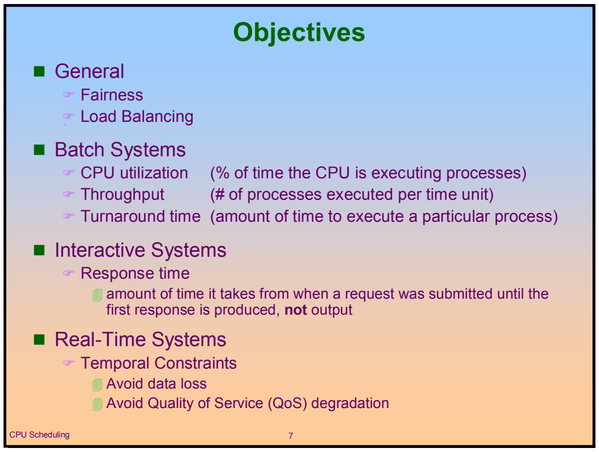
Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - Context Switch
  - 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.
- Context-switches should be minimized



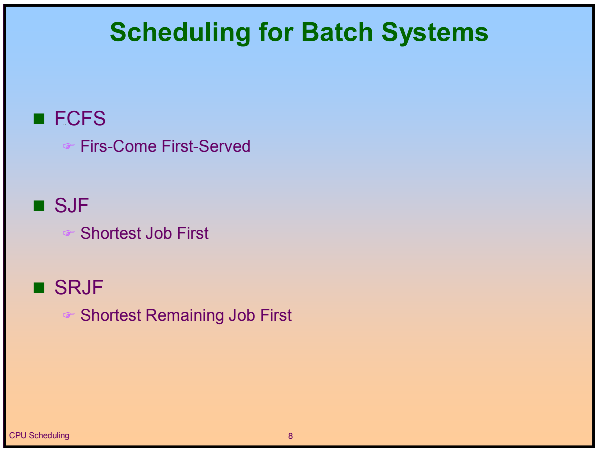
Type of scheduling

- Batch Systems
  - Maximize the resource utilization
- Interactive Systems
  - Minimize response times
- Real-Time Systems
  - Meet Temporal Constraints



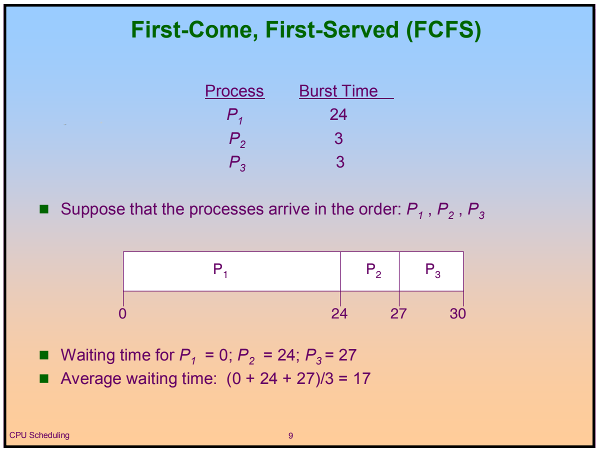
Objectives

- General
  - Fairness
  - Load Balancing
- Batch Systems
  - CPU utilization (% of time the CPU is executing processes)
  - Throughput (# of processes executed per time unit)
  - Turnaround time (amount of time to execute a particular process)
- Interactive Systems
  - Response time
    - amount of time it takes from when a request was submitted until the first response is produced, **not** output
- Real-Time Systems
  - Temporal Constraints
    - Avoid data loss
    - Avoid Quality of Service (QoS) degradation



Scheduling for Batch Systems

- FCFS
  - Firs-Come First-Served
- SJF
  - Shortest Job First
- SRJF
  - Shortest Remaining Job First



First-Come, First-Served (FCFS)

Process	Burst Time
P_1	24
P_2	3
P_3	3

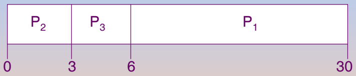
- Suppose that the processes arrive in the order: P_1 , P_2 , P_3

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

FCFS (Cont.)

Suppose that the processes arrive in the order

P_2 , P_3 , P_1 .



- 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

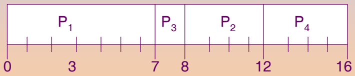
Shortest-Job-First (SJR)

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

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)

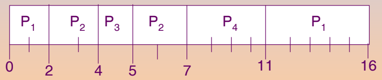


- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

Example of Preemptive SJF (SRJF)

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

Scheduling for Interactive Systems

- Round-Robin (RR)
- Priority-based
- Shortest Process Next
  - Approximated SJF
- Multi-level

Round Robin (RR)

- Each process gets a small unit of CPU time (*time quantum*).
  - After this time has elapsed, the process is preempted and added to the end of the ready queue.
- *n* processes in the ready queue; time quantum = *q*
  - 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.

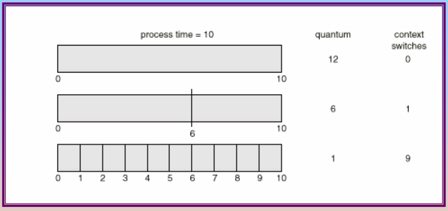
Example of RR with Time Quantum = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

P_1 | P_2 | P_3 | P_4 | P_1 | P_3 | P_4 | P_1 | P_3 | P_3

0 20 37 57 77 97 117 121 134 154 162

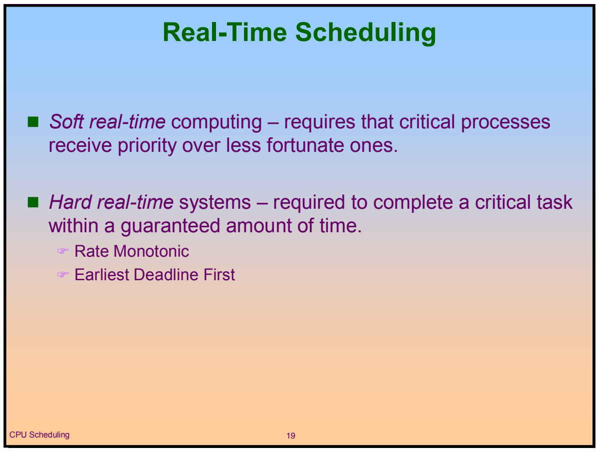
Time Quantum and Perfomance



- Time quantum impacts on
  - Number of Context Switches
  - Average Response Time
- RR is fair by definition
  - All processes are given the same chances

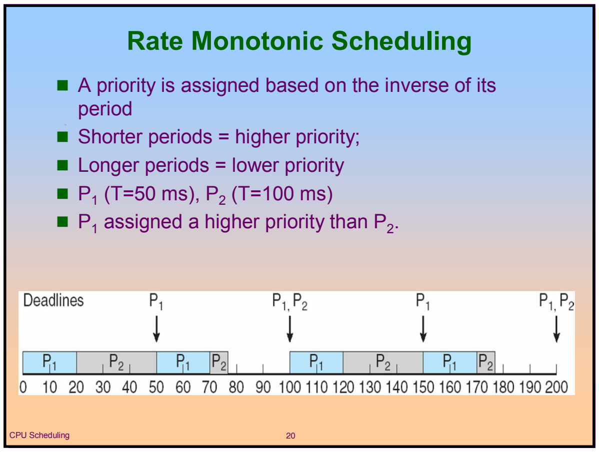
Priority Scheduling

- A priority number (integer) is associated with each process
  - Static vs. Dynamic Priority
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority).
  - Preemptive vs.Non-preemptive
- SJF (SRJF) is a priority scheduling where priority is the predicted (remaining) CPU burst time.
- Problem
  - Starvation – low priority processes may never execute.
- Solution
  - Aging – as time progresses increase the priority of the process.
- Classes of priority



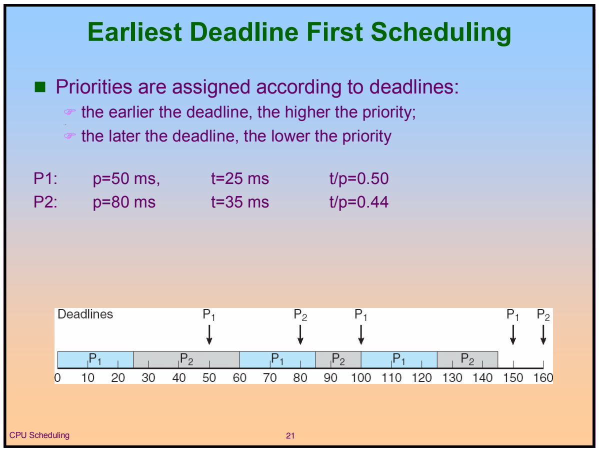
Real-Time Scheduling

- *Soft real-time* computing – requires that critical processes receive priority over less fortunate ones.
- *Hard real-time* systems – required to complete a critical task within a guaranteed amount of time.
  - Rate Monotonic
  - Earliest Deadline First



Rate Monotonic Scheduling

- A priority is assigned based on the inverse of its period
- Shorter periods = higher priority;
- Longer periods = lower priority
- P_1 (T=50 ms), P_2 (T=100 ms)
- P_1 assigned a higher priority than P_2.



Earliest Deadline First Scheduling

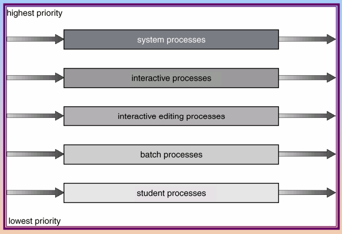
- Priorities are assigned according to deadlines:
  - the earlier the deadline, the higher the priority;
  - the later the deadline, the lower the priority

P1: p=50 ms, t=25 ms t/p=0.50
P2: p=80 ms t=35 ms t/p=0.44

Multilevel Queue

- Ready queue is partitioned into separate queues:
  - foreground (interactive)
  - background (batch)
- 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
    - Example: 80% to foreground in RR; 20% to background in FCFS

Multilevel Queue Scheduling



highest priority

system processes

interactive processes

interactive editing processes

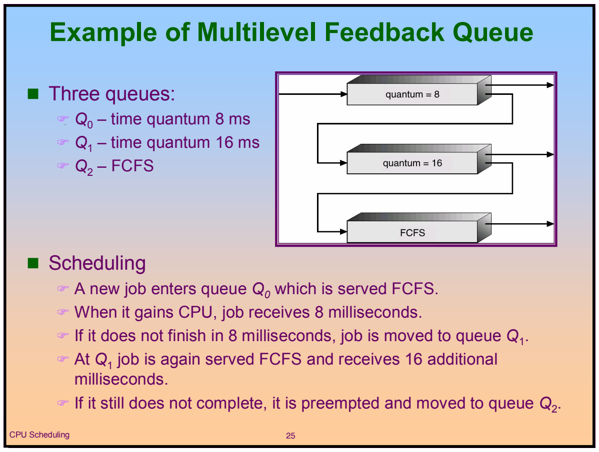
batch processes

student processes

lowest priority

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



Example of Multilevel Feedback Queue

- Three queues:
  - Q_0 – time quantum 8 ms
  - Q_1 – time quantum 16 ms
  - Q_2 – FCFS

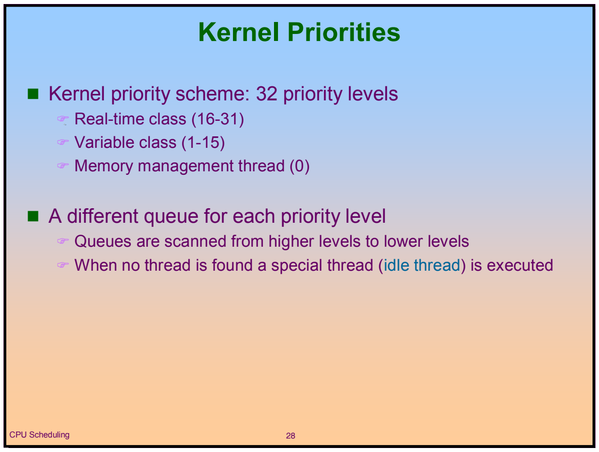
- Scheduling
  - A new job enters queue Q_0 which is served FCFS.
  - When it gains CPU, job receives 8 milliseconds.
  - If it does not finish in 8 milliseconds, job is moved to queue Q_1.
  - At Q_1 job is again served FCFS and receives 16 additional milliseconds.
  - If it still does not complete, it is preempted and moved to queue Q_2.

Operating System Examples

- Windows XP scheduling
- Linux scheduling

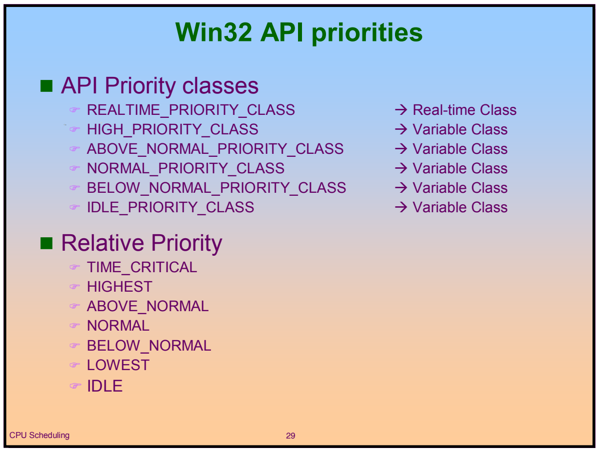
Windows XP Scheduling

- Thread scheduling based on
  - Priority
  - Preemption
  - Time slice
- A thread is execute until one of the following event occurs
  - The thread has terminated its execution
  - The thread has exhausted its assigned time slice
  - The has executed a blocking system call
  - A thread higher-priority thread has entered the ready queue



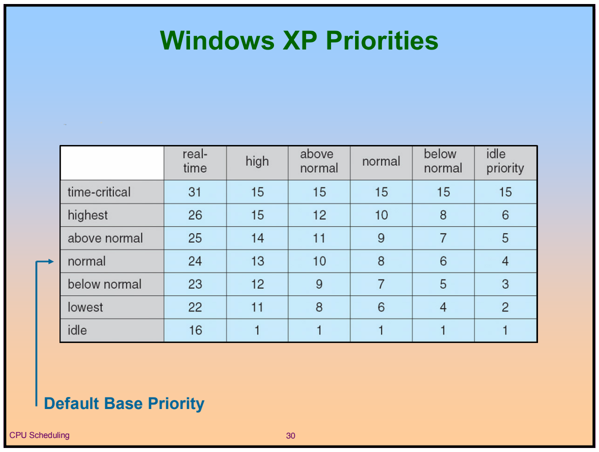
Kernel Priorities

- Kernel priority scheme: 32 priority levels
  - Real-time class (16-31)
  - Variable class (1-15)
  - Memory management thread (0)
- A different queue for each priority level
  - Queues are scanned from higher levels to lower levels
  - When no thread is found a special thread (idle thread) is executed



Win32 API priorities

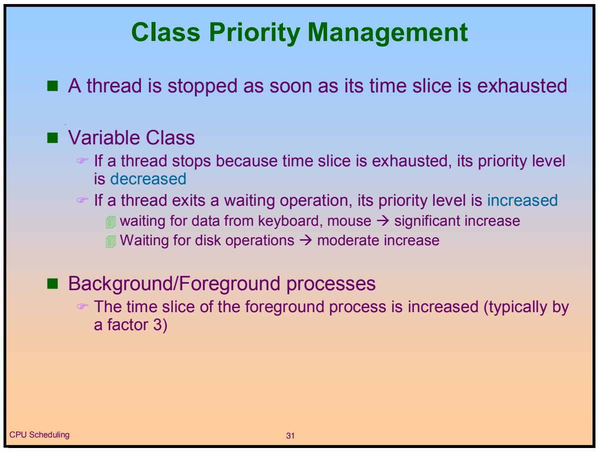
- API Priority classes
  - REALTIME_PRIORITY_CLASS → Real-time Class
  - HIGH_PRIORITY_CLASS → Variable Class
  - ABOVE_NORMAL_PRIORITY_CLASS → Variable Class
  - NORMAL_PRIORITY_CLASS → Variable Class
  - BELOW_NORMAL_PRIORITY_CLASS → Variable Class
  - IDLE_PRIORITY_CLASS → Variable Class
- Relative Priority
  - TIME_CRITICAL
  - HIGHEST
  - ABOVE_NORMAL
  - NORMAL
  - BELOW_NORMAL
  - LOWEST
  - IDLE



Windows XP Priorities

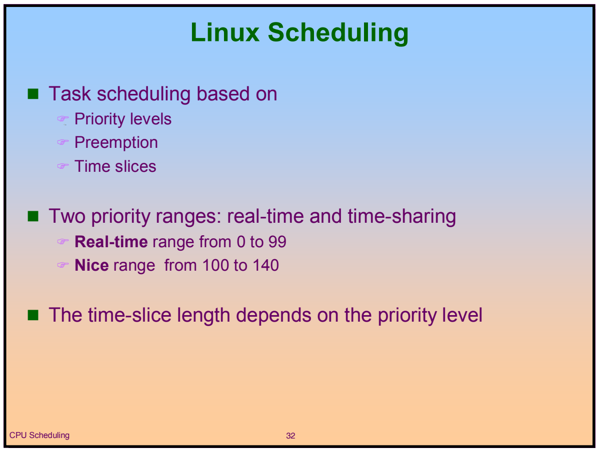
	real-time	high	above normal	normal	below normal	idle priority
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

Default Base Priority



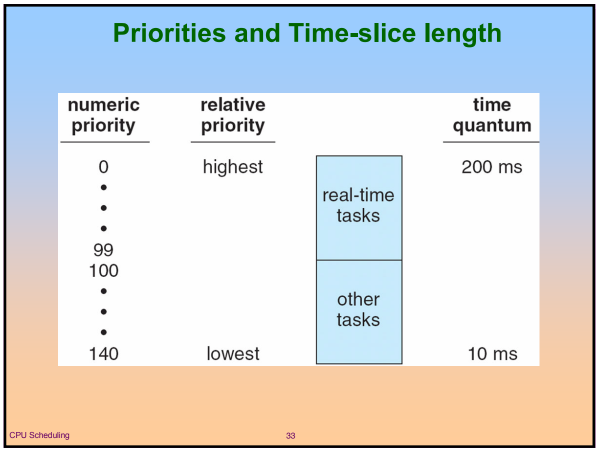
Class Priority Management

- A thread is stopped as soon as its time slice is exhausted
- Variable Class
  - If a thread stops because time slice is exhausted, its priority level is decreased
  - If a thread exits a waiting operation, its priority level is increased
    - waiting for data from keyboard, mouse → significant increase
    - Waiting for disk operations → moderate increase
- Background/Foreground processes
  - The time slice of the foreground process is increased (typically by a factor 3)



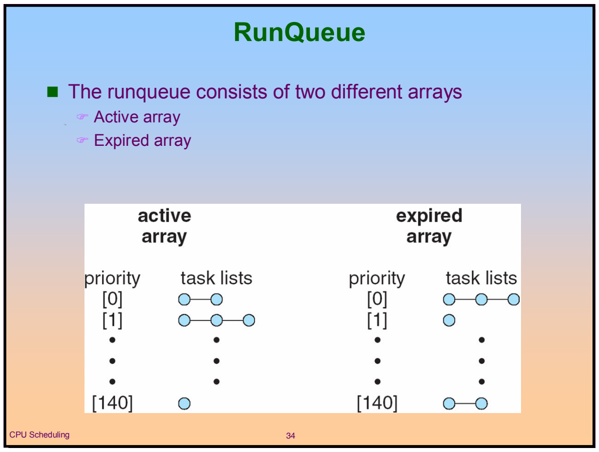
Linux Scheduling

- Task scheduling based on
  - Priority levels
  - Preemption
  - Time slices
- Two priority ranges: real-time and time-sharing
  - **Real-time** range from 0 to 99
  - **Nice** range from 100 to 140
- The time-slice length depends on the priority level



Priorities and Time-slice length

numeric priority	**relative priority**		**time quantum**
0 • • • 99	highest	real-time tasks	200 ms
100 • • • 140	lowest	other tasks	10 ms

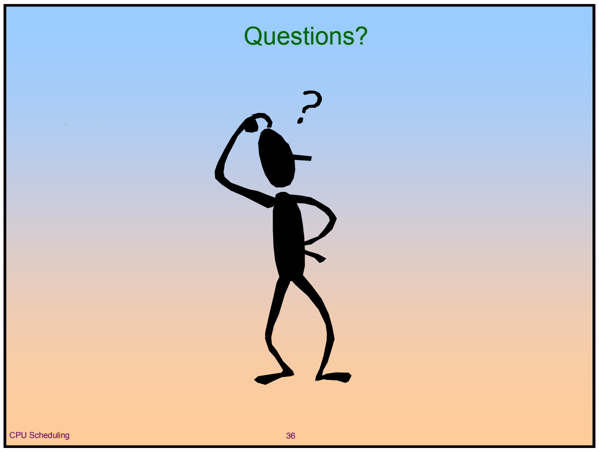


RunQueue

- The runqueue consists of two different arrays
  - Active array
  - Expired array

Priority Calculation

- Real time tasks have static priority
- Time-sharing tasks have dynamic priority
  - Based on nice value ± 5
  - ± 5 depends on how much the task is interactive
    - Tasks with low waiting times are assumed to be scarcely interactive
    - Tasks with large waiting times are assumed to be highly interactive
- Priority re-computation is carried out every time a task has exhausted its time slice



Questions?