Operating Systems
ECE344

Lecture 7: Scheduling

Ding Yuan

Announcement & Reminder

- Midterm exam
  - Will grade them this Friday
  - Will post the solution online before next lecture
  - Will briefly go over the common mistakes next Monday

Scheduling Overview

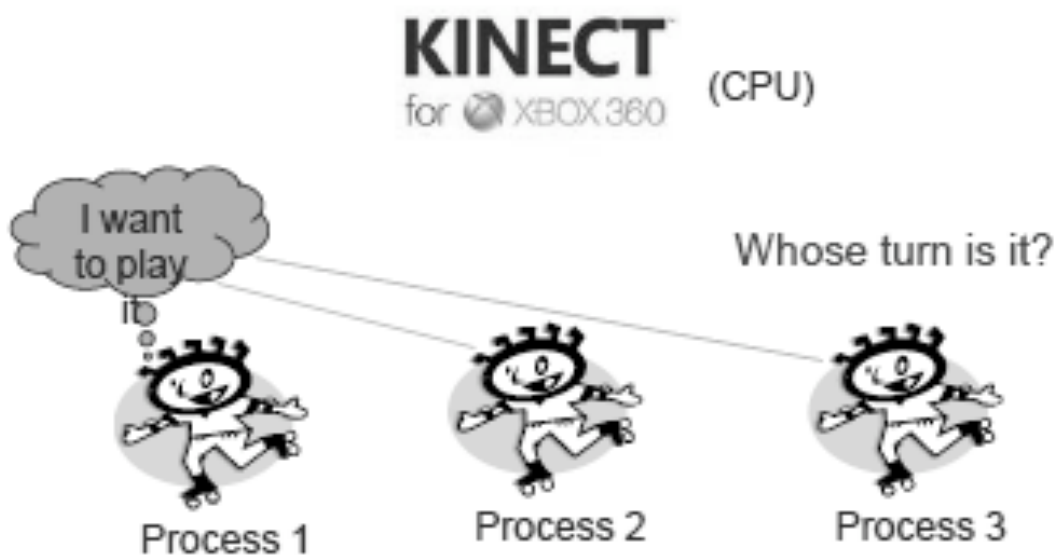
- In discussing process management and synchronization, we talked about context switching among processes/threads on the ready queue
- But we have glossed over the details of exactly which thread is chosen from the ready queue
- Making this decision is called scheduling
- In this lecture, we'll look at:
  - The goals of scheduling
  - Various well-known scheduling algorithms
  - Standard Unix scheduling algorithm

Multiprogramming

- In a multiprogramming system, we try to increase CPU utilization and job throughput by overlapping I/O and CPU activities
  - Doing this requires a combination of mechanisms and policy
- We have covered the mechanisms
  - Context switching, how it happens
  - Process queues and process states
- Now we'll look at the policies
  - **Which** process (thread) to run, for **how long**, etc.
- We'll refer to schedulable entities as jobs (standard usage) – could be processes, threads, people, etc.

Scheduling

- Deciding which process/thread should occupy the resource (CPU, disk, etc.)

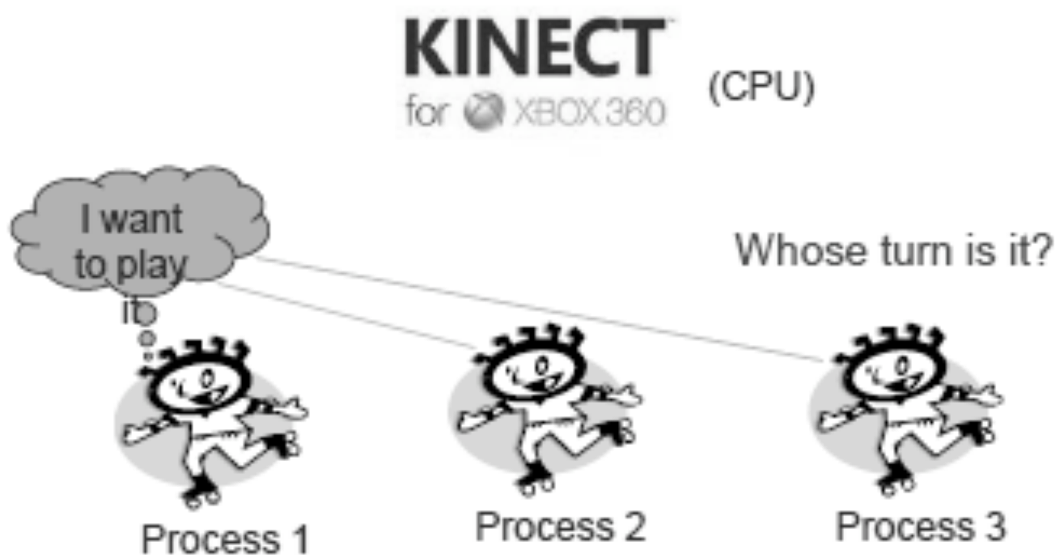


When to schedule?

- A new job starts
- The running job exits
- The running job is blocked
- I/O interrupt (some processes will be ready)
- Timer interrupt
  - Every 10 milliseconds (Linux 2.4)
  - Every 1 millisecond (Linux 2.6)
  - Why is the change?
  - Read this if you are interested (not required for exam): *http://kerneltrap.org/node/5411*

What are the scheduling objectives?

- Anyone?



Scheduling Objectives

- Fair (nobody cries)
- Priority (lady first)
- Efficiency (make best use of equipment)
- Encourage good behavior (good boy/girl)
- Support heavy load (degrade gracefully)
- Adapt to different environment (interactive, real-time, multi-media, etc.)

Performance Criteria

- Throughput: # of jobs that complete in unit time
- Turnaround time (also called elapse time)
  - Amount of time to execute a particular process from the time it entered
- Waiting time
  - amount of time process has been waiting in ready queue
- Meeting deadlines: avoid bad consequences

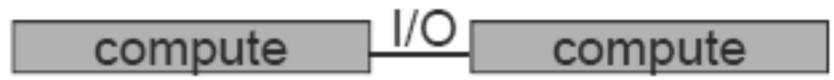
Different Systems, Different Focuses

- Batch Systems (e.g., billing, accounts receivable, accounts payable, etc.)
  - Max throughput, max CPU utilization
- Interactive Systems (e.g., our PC)
  - Min. response time
- Real-time system (e.g., airplane)
  - Priority, meeting deadlines
    - Example: on airplane, Flight Control has strictly higher priority than Environmental Control

Program Behaviors Considered in Scheduling

- Is it I/O bound? Example?

- Is it CPU bound? Example?



- Batch or interactive environment
- Priority
- Frequency of page fault
- Frequency of preemption

Midterm Exam

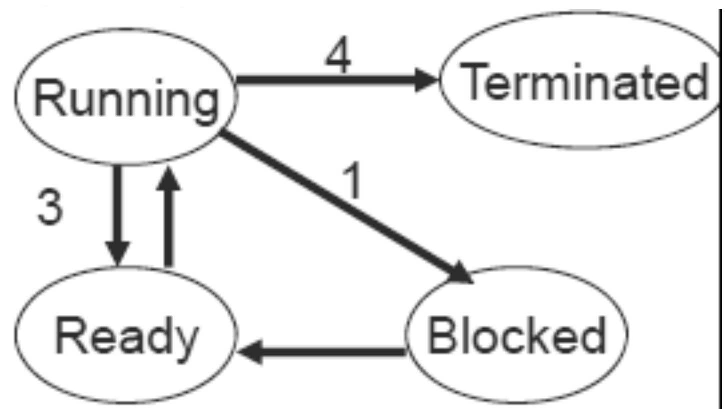
- Grades available in Portal
- Mean: 69
- Median: 72
- Regrade: submit your request before Mar/11
  - send me an email
- If you get < 50, I encourage you to send me an email to discuss how I can help you to do better

Review of last lecture

- Scheduling
  - What is scheduling?
  - When to schedule?
  - Objectives?

Preemptive vs. Non-preemptive

- Non-preemptive scheduling
  - The running process keeps the CPU until it voluntarily gives up the CPU
    - Process exits
    - Switch to blocked state
    - 1 and 4 only (no 3 unless calls yield)



- Preemptive scheduling
  - The running process can be interrupted and must release the CPU

Scheduling Algorithms

- First Come First Serve (FCFS)
- Short Job First (SJF)

Batch Systems

- Priority Scheduling
- Round Robin
- Multi-Queue & Multi-Level Feedback

Interactive Systems

- Earliest Deadline First Scheduling

Real-time Systems

First Come First Serve (FCFS)

- Also called first-in first-out (FIFO)
  - Jobs are scheduled in order of arrival to ready queue
  - “Real-world” scheduling of people in lines (e.g., supermarket)
  - Typically non-preemptive (no context switching at market)
  - Jobs treated equally, no starvation

FCFS Example

Process	Duration	Order	Arrival Time
P1	24	1	0
P2	3	2	0
P3	4	3	0

P1 waiting time: 0
P2 waiting time: 24
P3 waiting time: 27

The average waiting time:
(0+24+27)/3 = 17

Problems with FCFS

- Average waiting time can be large if small jobs wait behind long ones (high turnaround time)
  - Non-preemptive
  - *You have a basket, but you're stuck behind someone with a cart*
- *Solution?*
  - Express lane (12 items or less)

Shortest Job First (SJF)

- Shortest Job First (SJF)
  - Choose the job with the smallest expected duration first
    - Person with smallest number of items to buy
  - Requirement: the job duration needs to be known in advance
  - Used in Batch Systems
  - Optimal for Average Waiting Time if all jobs are available simultaneously (provable). Why?
  - Real life analogy?
    - Express lane in supermarket
    - Shortest important task first

-- The 7 Habits of Highly Effective People

Non-preemptive SJF: Example

Process	Duration	Order	Arrival Time
P1	6	1	0
P2	8	2	0
P3	7	3	0
P4	3	4	0

P4 waiting time: 0
P1 waiting time: 3
P3 waiting time: 9
P2 waiting time: 16

The total time is: 24

The average waiting time (AWT): (0+3+9+16)/4 = 7

Comparing to FCFS

Process	Duration	Order	Arrival Time
P1	6	1	0
P2	8	2	0
P3	7	3	0
P4	3	4	0

P1 waiting time: 0
P2 waiting time: 6
P3 waiting time: 14
P4 waiting time: 21

The total time is the same (why?)
The average waiting time (AWT):
(0+6+14+21)/4 = 10.25
(comparing to 7)

SJF is not always optimal

- Is SJF optimal if not all the jobs are available simultaneously?

Process	Duration	Order	Arrival Time
P1	10	1	0
P2	2	2	2

P1 waiting time: 0
P2 waiting time: 8

The average waiting time (AWT):
(0+8)/2 = 4

Preemptive SJF

- Also called Shortest Remaining Time First
  - Schedule the job with the shortest remaining time required to complete
- Requirement: again, the duration needs to be known in advance

Preemptive SJF: Same Example

Process	Duration	Order	Arrival Time
P1	10	1	0
P2	2	2	2

P1 waiting time: 4-2 =2
P2 waiting time: 0

The average waiting time (AWT):
(0+2)/2 = 1

No CPU waste!!!

A Problem with SJF

- Starvation
  - In some condition, a job is waiting forever
  - Example:
    - Process A with duration of 1 hour, arrives at time 0
    - But every 1 minute, a short process with duration of 2 minutes arrive
    - Result of SJF: A never gets to run

Scheduling Algorithms

- First Come First Serve (FCFS)
- Short Job First (SJF)

Batch Systems

- Priority Scheduling
- Round Robin
- Multi-Queue & Multi-Level Feedback

Interactive Systems

- Earliest Deadline First Scheduling

Real-time Systems

Priority Scheduling

- Each job is assigned a priority
- FCFS within each priority level
- Select highest priority job over lower ones
- Rationale: higher priority jobs are more mission-critical
  - Example: DVD movie player vs. send email
- Real life analogy?
  - Boarding at airports
- Problems:
  - May not give the best AWT
  - indefinite blocking or starving a process

Set Priority

- Two approaches
  - Static (for systems with well-known and regular application behaviors)
  - Dynamic (otherwise)
- Priority may be based on:
  - Importance
  - Percentage of CPU time used in last X hours
    - Should a job have higher priority if it used more CPU in the past? Why?

Priority Schedulring: Example

Process	Duration	Priority	Arrival Time
P1	6	4	0
P2	8	1	0
P3	7	3	0
P4	3	2	0

P2 waiting time: 0
P4 waiting time: 8
P3 waiting time: 11
P1 waiting time: 18

The average waiting time (AWT):
(0+8+11+18)/4 = 9.25

(worse than SJF)

Priority in Unix

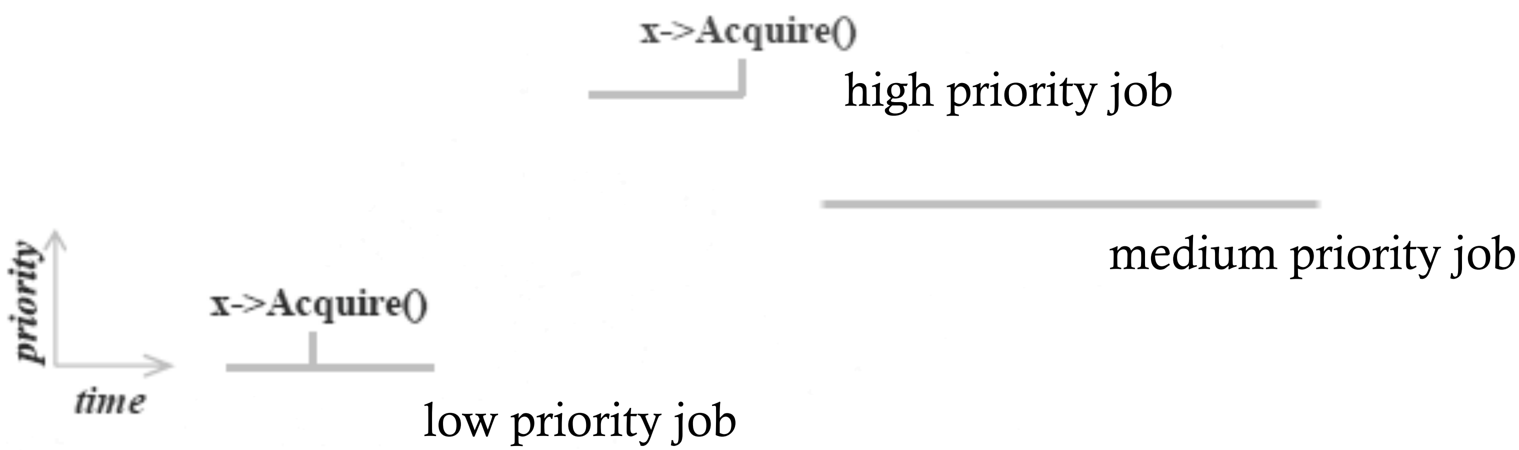
```
dMacBookPro-ajksiq-2:~ ding$ ps -l 22667 22638 1772 24097 23321 1744
  UID   PID  PPID CPU PRI NI      VSZ    RSS WCHAN  STAT   TT       TIME COMMAND
  502  1744    95   0  63  0   460588  19712 -      S      ??   23:54.27 /Users/Ding/Downloads/iTerm.a
pp/Contents/MacOS/iTerm -psn_0_1872329
  502  1772    95   0  46  0   730728  81508 -      S      ??   22:51.61 /Applications/Microsoft Offic
e 2008/Microsoft Word.app/Contents/MacOS/Microsoft Word -psn_0_1876426
  502 22638    95   0  31  0   419924   4420 -      S      ??    0:01.64 /Users/Ding/Library/Printers/
Brother DCP-7060D.app/Contents/MacOS/PrinterProxy -psn_0_13421772
  502 22667    95   0  62  0   897612 216952 -      S      ??   58:39.34 /Applications/Safari.app/Cont
ents/MacOS/Safari -psn_0_13458645
  502 23321    95   0  63  0   742388 183984 -      S      ??    2:56.82 /Applications/iTunes.app/Cont
ents/MacOS/iTunes -psn_0_13692174
  502 24097    95   0  46  0   638352 141540 -      S      ??    2:44.58 /Applications/Microsoft Offic
e 2008/Microsoft PowerPoint.app/Contents/MacOS/Microsoft PowerPoint -psn_0_14105971
  502 24058 24057   0  31  0   600252    940 -      S    s000    0:00.01 -bash
  502 24069 24058   0  31  0   599928   1008 -      S+   s000    0:00.10 ssh yuan@anubis.eecg.toronto.
edu
  502 16280 16279   0  31  0   600252    800 -      S    s001    0:00.12 -bash
  502 16340 16339   0  31  0   600252    680 -      S+   s002    0:00.02 -bash
```

Nobody wants to *Be "nice" on Unix*

```
NICE(1)                          FSF                          NICE(1)

NAME
       nice - run a program with modified scheduling priority

SYNOPSIS
       nice [OPTION] [COMMAND [ARG]...]

DESCRIPTION
       Run COMMAND with an adjusted scheduling priority.  With no
       COMMAND, print the current scheduling priority.  ADJUST is
       10  by default.  Range goes from -20 (highest priority) to
       19 (lowest).

       -ADJUST
              increment priority by ADJUST first

       -n, --adjustment=ADJUST
              same as -ADJUST

       --help display this help and exit

       --version
lines 1-25
```

More on Priority Scheduling

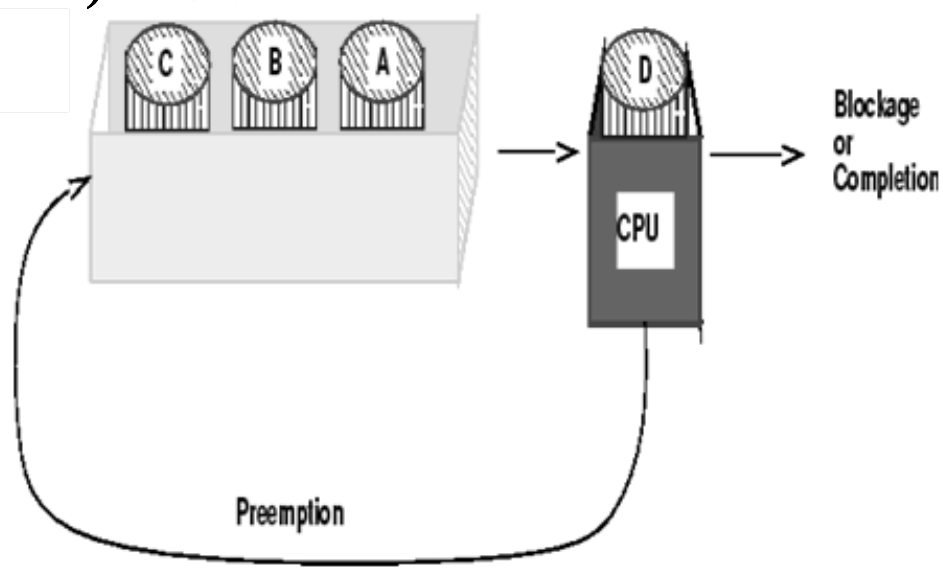
- For real-time (predictable) systems, priority is often used to isolate a process from those with lower priority. *Priority inversion*: high priority task is indirectly preempted by medium/low priority tasks
  - A solution: priority inheritance



Round-robin

- One of the oldest, simplest, most commonly used scheduling algorithm
- Select process/thread from ready queue in a round-robin fashion (take turns)
- Real life analogy?

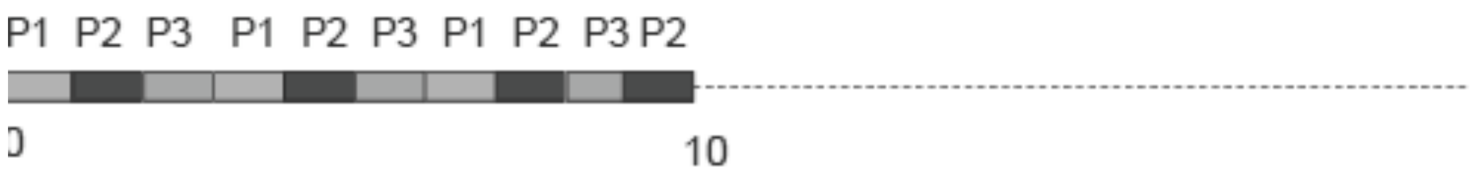
Problem:
- *Do not consider priority*
- *Context switch overhead*



Round-Robin: example

Process	Duration	Order	Arrival Time
P1	3	1	0
P2	4	2	0
P3	3	3	0

Suppose time quantum is: 1 unit, P1, P2 & P3 never block



P1 waiting time: 4
P2 waiting time: 6
P3 waiting time: 6

The average waiting time (AWT):
(4+6+6)/3 = 5.33

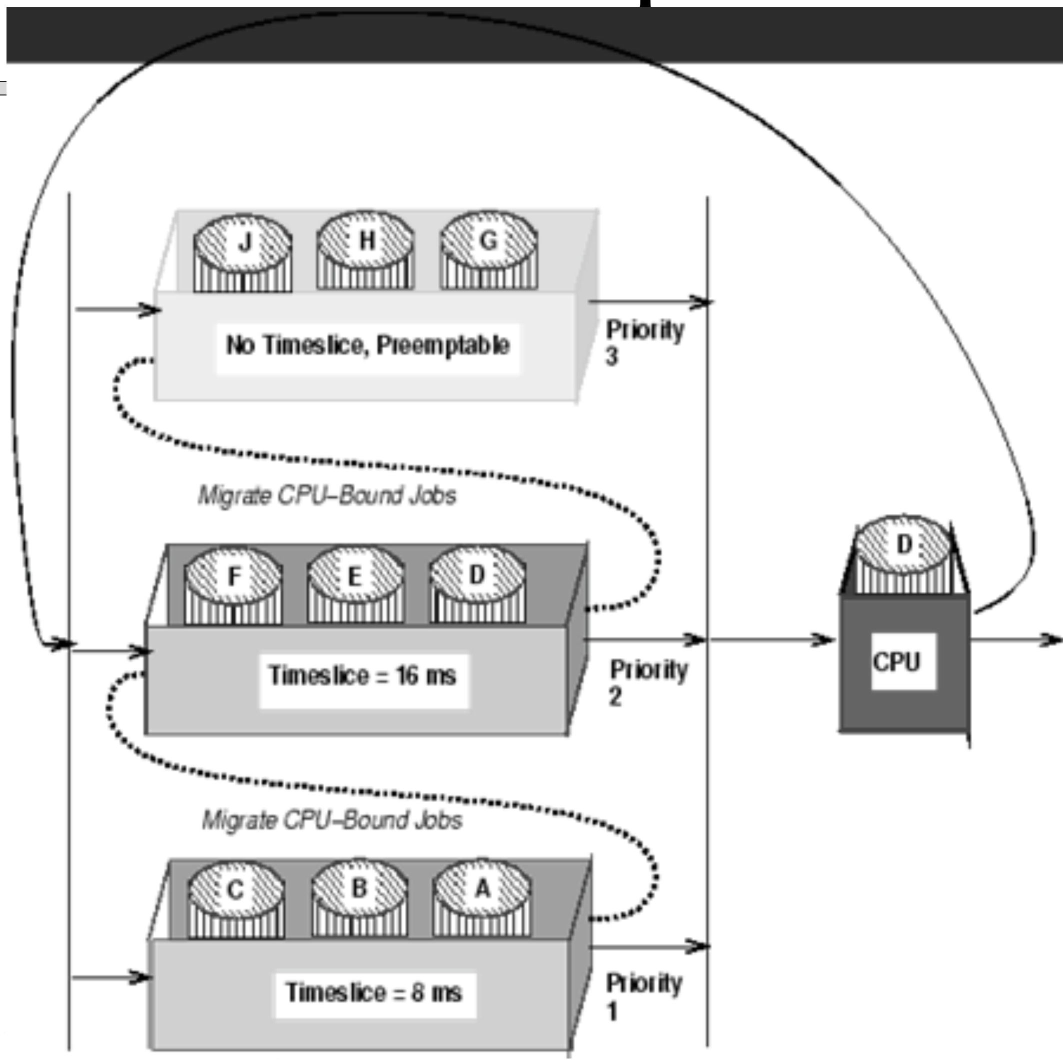
Time Quantum

- Time slice too large
  - FIFO behavior
  - Poor response time
- Time slice too small
  - Too many context switches (overheads)
  - Inefficient CPU utilization
- Heuristics: 70-80% of jobs block within time-slice
- Typical time-slice: 5 – 100 ms
  - *Wait: isn't timer-interrupt frequency 1ms on Linux 2.6?*

Combining Algorithms

- Scheduling algorithms can be combined
  - Have multiple queues
  - Use a different algorithm for each queue
  - Move processes among queues
- Example: Multiple-level feedback queues (MLFQ)
  - Multiple queues representing different job types
    - Interactive, CPU-bound, batch, etc.
  - Queues have priorities
  - Jobs can move among queues based upon execution history
    - Feedback: switch from interactive to CPU-bound behavior

Example


J
H
G
No Timeslice, Preemptable
Priority 3
Migrate CPU–Bound Jobs
F
E
D
Timeslice = 16 ms
Priority 2
D
CPU
Migrate CPU–Bound Jobs
C
B
A
Timeslice = 8 ms
Priority 1

Unix Scheduler

- The Unix scheduler uses a MLFQ
  - ~170 priority levels
- Priority scheduling across queues, RR within a queue
  - The process with the highest priority always runs
  - Processes with the same priority are scheduled RR
- Processes dynamically change priority
  - Increases over time if process blocks before end of quantum
  - Decreases over time if process uses entire quantum

Motivation of Unix Scheduler

- The idea behind the Unix scheduler is to reward interactive processes over CPU hogs
- Interactive processes (shell, editor, etc.) typically run using short CPU bursts
  - *They do not finish quantum before waiting for more input*
- Want to minimize response time
  - Time from keystroke (putting process on ready queue) to executing keystroke handler (process running)
  - Don't want editor to wait until CPU hog finishes quantum
- This policy delays execution of CPU-bound jobs
  - But that's ok

Scheduling Algorithms

- First Come First Serve (FCFS)
- Short Job First (SJF)

Batch Systems

- Priority Scheduling
- Round Robin
- Multi-Queue & Multi-Level Feedback

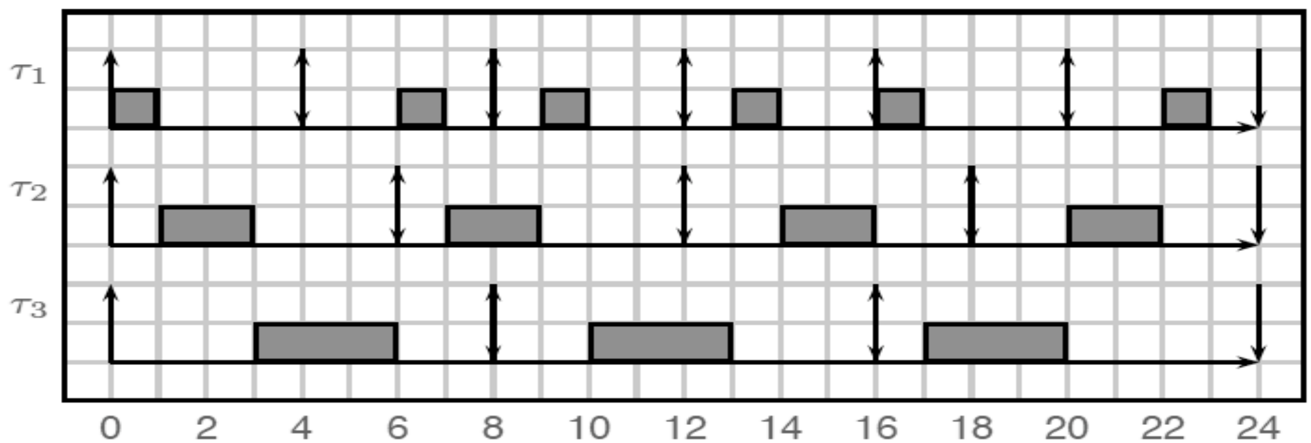
Interactive Systems

- Earliest Deadline First Scheduling

Real-time Systems

Earlieas Deadline First (EDF)

- Each job has an arrival time and a *deadline* to finish
  - Real life analogy?
- Always pick the job with the earliest deadline to run



- Optimal algorithm (provable): if the jobs can be scheduled (by any algorithm) to all meet the deadline, EDF is one of such schedules

Scheduling Summary

- Scheduler (dispatcher) is the module that gets invoked when a context switch needs to happen
- Scheduling algorithm determines which process runs, where processes are placed on queues
- Many potential goals of scheduling algorithms
  - Utilization, throughput, wait time, response time, etc.
- Various algorithms to meet these goals
  - FCFS/FIFO, SJF, Priority, RR
- Can combine algorithms
  - Multiple-level feedback queues
  - Unix example