TCSS 422: OPERATING SYSTEMS

**Three Easy Pieces
Process API,
Limited Direct Execution,
Scheduling Introduction**

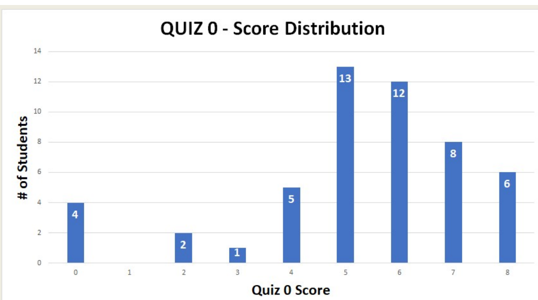
Wes J. Lloyd
Institute of Technology
University of Washington - Tacoma

OBJECTIVES

- Assignment 0 – Introduction to Linux
- Tutorial 1 – C Tutorial: Pointers, Strings, Exec
- Feedback from 4/2

- Introduction to Scheduling – Ch. 7
- Multi-level Feedback Queue Scheduler – Ch. 8

QUIZ 0 - REVIEW



FEEDBACK FROM 4/2

- In the fork code examples, why is both the child and parent executed?

  They're surrounded by "else if" blocks. Isn't only one executed?

- If a time slice is longer than the amount of time a process needs to complete, does the machine still wait for the next timer interrupt to context-switch?

FEEDBACK - 2

- On homework #0: how specific should the commands be?

  Some commands show a lot of extra info.

  Should this be filtered out?

FEEDBACK - 3

- <u>How can we minimize context switching (C/S) overhead?</u>
  - Are processes using their full time slice?
    - ***The time slice should be selected carefully.***
  - HW support (on the CPU) can minimize overhead
    - ***Ex.: CPU should not flush memory page table cache***
  - Avoid having threads BLOCK
    - Blocking induces a context switch
    - When checking LOCK availability:
      - Requesting a lock that is unavailable causes a C/S
      - Perform short lived busy waiting to check for LOCK availability
      - Helps avoid C/S

FEEDBACK - 4

- Preemptive multi-tasking – is the timer interrupt the only method for the OS to regain control of the CPU?
- What are CPU modes?
- Why is there an unused privilege ring (2) between VM and user? What is it for?
- What are system calls?

CHAPTER 7- SCHEDULING: INTRODUCTION

SCHEDULING INTRODUCTION

- For simplicity, consider job scheduling with limitations:
  - Each job requires the same CPU time
  - All jobs arrive at the same time
  - All jobs only use the CPU (no I/O)
  - The run-time of each job is known a priori

SCHEDULING METRICS

- **Metrics**: A standard measure to quantify to what degree a system possesses some property. Metrics provide ***repeatable*** techniques to quantify and compare systems.
- **Measurements** are the numbers derived from the application of metrics
- Scheduling Metric #1: **Turnaround time**
- The time at which the job completes minus the time at which the job arrived in the system

$$T_{turnaround} = T_{completion} - T_{arrival}$$

- How is turnaround time different than execution time?

SCHEDULING METRICS - 2

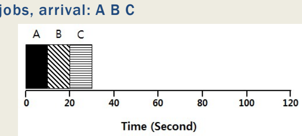
- Scheduling Metric #2: **Fairness**
  - Jain's fairness index
  - Quantifies if jobs receive a fair share of system resources

$$\mathcal{J}(x_1, x_2, \ldots, x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \cdot \sum_{i=1}^{n} {x_i}^2}$$

- n processes
- x_i is time share of each process
- worst case = 1/n
- best case = 1
- Consider n=3, worst case = .333, best case=1
- With n=3 and x_1=.2, x_2=.7, x_3=.1, fairness=.62
- With n=3 and x_1=.33, x_2=.33, x_3=.33, fairness=1

SCHEDULERS

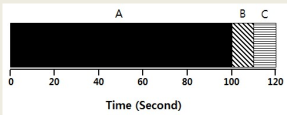
- FIFO: first in, first out
  - Very simple, easy to implement
- Consider
  - 3 x 10sec jobs, arrival: A B C



$$Average\ turnaround\ time = \frac{10 + 20 + 30}{3} = 20\ sec$$

FIFO: CONVOY EFFECT

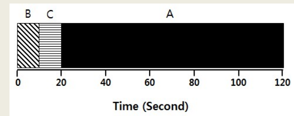
- FIFO with different jobs lengths
- Consider
  - A_{len}=100sec, B_{len}=10sec, C_{len}=10sec



$$Average\ turnaround\ time = \frac{100 + 110 + 120}{3} = 110\ sec$$

SJF: SHORTEST JOB FIRST

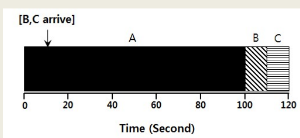
- Given that we know execution times in advance:
  - Run in order of duration, shortest to longest
  - Non preemptive scheduler
  - This is not realistic
  - Arrival: A B C



$$Average\ turnaround\ time = \frac{10 + 20 + 120}{3} = 50\ sec$$

SJF: WITH RANDOM ARRIVAL

- If jobs arrive at any time:
- A @ t=0sec, B @ t=10sec, C @ t=10sec



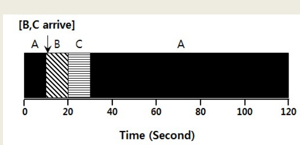
$$Average\ turnaround\ time = \frac{100 + (110 - 10) + (120 - 10)}{3} = 103.33\ sec$$

STCF – SHORTEST TIME TO COMPLETION FIRST

- Add preemption to the Shortest Job First scheduler
  - Also called preemptive shortest job first (PSJF)
- When a new job enters the system:
  - Of all jobs, *Which has the least time left?*
  - PREMPT job execution, and schedule the **new** shortest job
- More realistic, but how do we know execution time in advance?
  - Oracle: All knowing one
  - Only schedule static (fixed size) batch workloads
  - Can we predict execution time?

STCF - 2

- Consider:
  - A_{len}=100 $A_{arrival}$=0
  - B_{len}=10, $B_{arrival}$=10, C_{len}=10, $C_{arrival}$=10



$$Average\ turnaround\ time = \frac{(120 - 0) + (20 - 10) + (30 - 10)}{3} = 50\ sec$$

SCHEDULING METRICS - 3

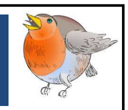
- Scheduling Metric #3: **Response Time**
- Time from when job arrives until it starts execution

$$T_{response} = T_{firstrun} - T_{arrival}$$

- STCF, SJF, FIFO
  - can perform poorly with respect to response time

What scheduling algorithm(s) can help minimize response time?

RR: ROUND ROBIN



- Run each job awhile, then switch to another distributing the CPU evenly (fairly)
- Scheduling Quantum is called a time slice
- Time … a mu… timer interrupt period.

Process	Burst Time
P1	12
P5	5

RR is fair, but performs poorly on metrics such as turnaround time

Round Robin scheduling algorithm Gantt chart

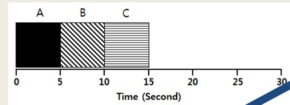
P1	P2	P3	P4	P5	P1	P2	P4	P1

0 5 10 14 19 24 29 32 37 39

Scheduling Quantum = 5 seconds

RR EXAMPLE

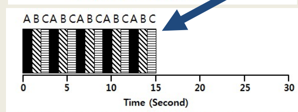
- ABC arrive at time=0, each run for 5 seconds



SJF (Bad for Response Time)

OVERHEAD not considered

$$T_{average\ response} = \frac{0+5+10}{3} = 5sec$$



RR with a time-slice of 1sec (Good for Response Time)

$$T_{average\ response} = \frac{0+1+2}{3} = 1sec$$

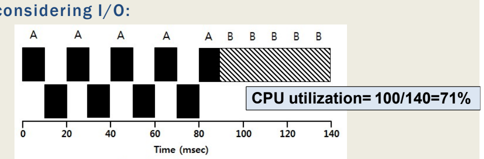
ROUND ROBIN: TRADEOFFS

Short Time Slice	Long Time Slice
Fast Response Time	Slow Response Time
High overhead from context switching	Low overhead from context switching

- Time slice impact:
  - Turnaround time (for earlier example): ts(1,2,3,4,5)=14,14,13,14,10
  - Fairness: round robin is always fair, J=1

SCHEDULING WITH I/O

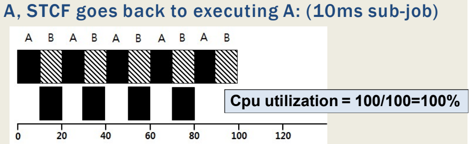
- STCF scheduler
  - A: CPU=50ms, I/O=40ms, 10ms intervals
  - B: CPU=50ms, I/O=0ms
  - Consider A as 10ms subjobs (CPU, then I/O)
- Without considering I/O:



Poor Use of Resources

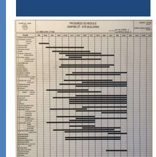
SCHEDULING WITH I/O - 2

- When a job initiates an I/O request
  - A is blocked, waits for I/O to compute, frees CPU
  - STCF scheduler assigns B to CPU
- When I/O completes → raise interrupt
  - Unblock A, STCF goes back to executing A: (10ms sub-job)



Overlap Allows Better Use of Resources

CHAPTER 8 – MULTI-LEVEL FEEDBACK QUEUE (MLFQ) SCHEDULER



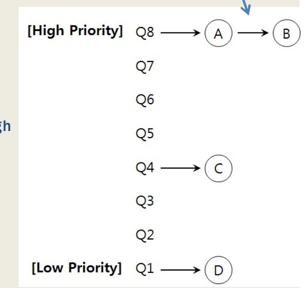
MULTI-LEVEL FEEDBACK QUEUE

- Objectives:
  - Improve turnaround time:
    Run shorter jobs first
  - Minimize response time:
    Important for interactive jobs (UI)
- Achieve without a priori knowledge of job length

MLFQ - 2

Round-Robin within a Queue

- Multiple job queues
- Adjust job priority based on observed behavior
- Interactive Jobs
  - Frequent I/O → keep priority high
  - Interactive jobs require fast response time (GUI/UI)
- Batch Jobs
  - Require long periods of CPU utilization
  - Keep priority low



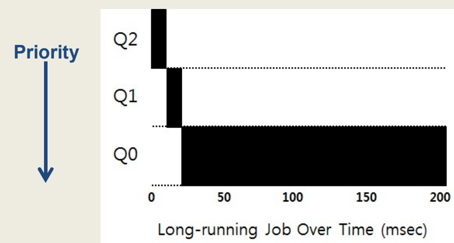
MLFQ: DETERMINING JOB PRIORITY

- New arriving jobs are placed into highest priority queue
- If a job uses its entire time slice, priority is reduced (↓)
  - Jobs appears CPU-bound (“batch” job), not interactive (GUI/UI)
- If a job relinquishes the CPU for I/O priority stays the same

MLFQ approximates SJF

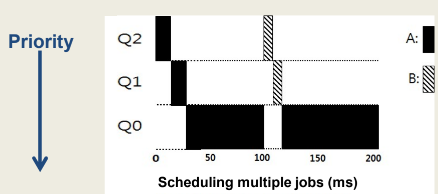
MLFQ: LONG RUNNING JOB

- Three-queue scheduler, time slice=10ms



MLFQ: BATCH AND INTERACTIVE JOBS

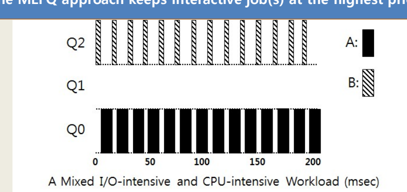
- $A_{arrival\_time}$=0ms, A_{run_time}=200ms,
- B_{run_time} =20ms, $B_{arrival\_time}$ =100ms



MLFQ: BATCH AND INTERACTIVE - 2

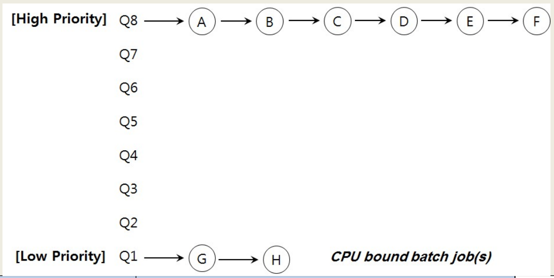
- Continuous interactive job (B) with long running batch job (A)
  - Low response time is good for B
  - A continues to make progress

The MLFQ approach keeps interactive job(s) at the highest priority



MLFQ: ISSUES

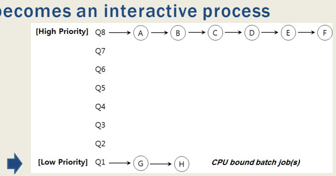
- Starvation



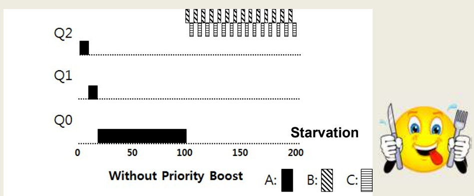
MLFQ: ISSUES - 2

- Gaming the scheduler
  - Issue I/O operation at 99% completion of the time slice
  - Keeps job priority fixed – never lowered
- Job behavioral change
  - CPU/batch process becomes an interactive process

Priority becomes stuck



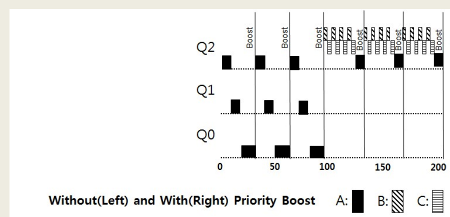
RESPONDING TO BEHAVIOR CHANGE



- Priority Boost
  - Reset all jobs to topmost queue after some time interval S

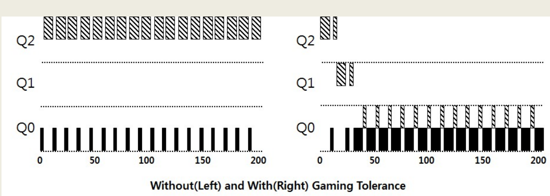
RESPONDING TO BEHAVIOR CHANGE - 2

- With priority boost
  - Prevents starvation



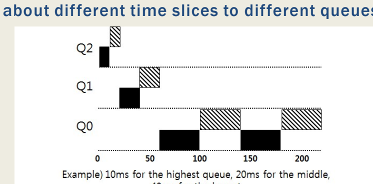
PREVENTING GAMING

- Improved time accounting:
  - Track total job execution time in the queue
  - Each job receives a fixed time allotment
  - When allotment is exhausted, job priority is lowered



MLFQ: TUNING

- Consider the tradeoffs:
  - How many queues?
  - What is a good time slice?
  - How often should we “Boost” priority of jobs?
  - What about different time slices to different queues?



PRACTICAL EXAMPLE

- Oracle Solaris MLFQ implementation
  - 60 Queues → w/ slowly increasing time slice (high to low priority)
  - Provides sys admins with set of editable table(s)
  - Supports adjusting time slices, boost intervals, priority changes, etc.
- Advice
  - Provide OS with hints about the process
  - Nice command → Linux

MLFQ RULE SUMMARY

- The refined set of MLFQ rules:
- **Rule 1:** If Priority(A) > Priority(B), A runs (B doesn't).
- **Rule 2:** If Priority(A) = Priority(B), A & B run in RR.
- **Rule 3:** When a job enters the system, it is placed at the highest priority.
- **Rule 4:** Once a job uses up its time allotment at a given level (regardless of how many times it has given up the CPU), its priority is reduced(i.e., it moves down on queue).
- **Rule 5:** After some time period S, move all the jobs in the system to the topmost queue.

Jackson deploys a 3-level MLFQ scheduler. The time slice is 1 for high priority jobs, 2 for medium priority, and 4 for low priority. This MLFQ scheduler performs a Priority Boost every 6 timer units. When the priority boost fires, the current job is preempted, and the next scheduled job is run in round-robin order.

Job	Arrival Time	Job Length
A	T=0	4
B	T=0	16
C	T=0	8

(11 points) Show a scheduling graph for the MLFQ scheduler for the jobs above.
Draw vertical lines for key events and be sure to label the X-axis times as in the example.
Please draw clearly. An unreadable graph will loose points.

HIGH |
MED |
LOW |
0

QUESTIONS