# TCSS 422: OPERATING SYSTEMS

## Common Scheduling Algorithms, Multi-level Feedback Queue (MLFQ) Scheduler, Proportional Share Schedulers

Wes J. Lloyd
School of Engineering and Technology
University of Washington - Tacoma



1

# TEXT BOOK COUPON

- 10% off textbook code: **WRITING10** (*through Friday Apr 18*)
- https://www.lulu.com/shop/andrea-arpaci-dusseau-and-remzi-arpaci-dusseau/operating-systems-three-easy-pieces-hardcover-version-110/hardcover/product-15gjeeky.html?q=three+easy+pieces+operating+systems&page=1&pageSize=4
- With coupon textbook is only $35.78 + tax & shipping



2

# TCSS 422 – OFFICE HRS – SPRING 2025

- <u>Office Hours plan for Spring (by Zoom):</u>
- **Monday 11:30am - 12:30p GTA Xinghan**
- **Tuesday 11:30am - 12:30p GTA Xinghan**
- **Wednesday 11:00am - 12:00p Instructor Wes**
- **Friday 12:00pm - 1:00p Wes or Xinghan**

- **Office hours this Friday April 18th**
  - Xinghan

- Instructor is available after class at 6pm in CP 229 each day



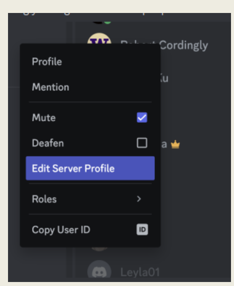
3

# TCSS 422 DISCORD SERVER

- Please join the TCSS 422 A – Spring 2025 Discord Server

- https://discord.gg/Jh5Cp8TMYn

- Under Edit Server Profile:
  Please update your 'Server Nickname' to your real name or UW NET ID
  THANK YOU






4

# OBJECTIVES – 4/17

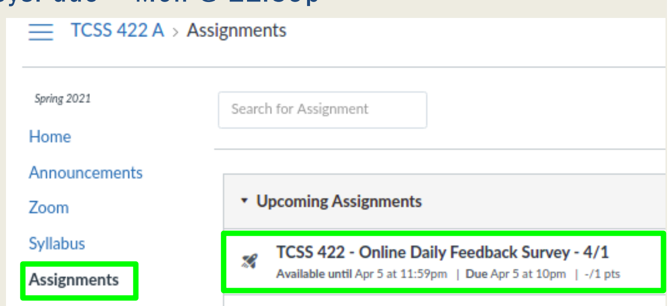
- Questions from 4/15
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Quiz 1 – Active Reading Chapter 9
- Chapter 7: Scheduling Introduction
- Chapter 8: Multi-level Feedback Queue
  - MLFQ Scheduler
  - Job Starvation
  - Gaming the Scheduler
  - Examples
- Chapter 9: Proportional Share Schedulers





# ONLINE DAILY FEEDBACK SURVEY

- Daily Feedback Quiz in Canvas – Available After Each Class
- Extra credit available for completing surveys ***ON TIME***
- Tuesday surveys: due by ~ Wed @ 11:59p
- Thursday surveys: due ~ Mon @ 11:59p



## TCSS 422 - Online Daily Feedback Survey - 4/1

### Quiz Instructions

**Question 1** 0.5 pts

On a scale of 1 to 10, please classify your perspective on material covered in today's class:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| Mostly Review To Me | | | | Equal New and Review | | | | | Mostly New to Me |

**Question 2** 0.5 pts

Please rate the pace of today's class:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|
| Slow | | | | Just Right | | | | | Fast |



7

# MATERIAL / PACE

- Please classify your perspective on material covered in today's class (35 of 63 respondents – 55.5%) :
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 6.88 (↑ - previous 6.47)**

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.05 (↑ - previous 4.83)**



8

## FEEDBACK FROM 4/15

- ***What is your recommendation for preparing for the quizzes and midterm?***



9

## OBJECTIVES – 4/17

- Questions from 4/15
- **Assignment 0**
- C Tutorial - Pointers, Strings, Exec in C
- Quiz 1 – Active Reading Chapter 9
- Chapter 7: Scheduling Introduction
- Chapter 8: Multi-level Feedback Queue
  - MLFQ Scheduler
  - Job Starvation
  - Gaming the Scheduler
  - Examples
- Chapter 9: Proportional Share Schedulers



10

## OBJECTIVES – 4/17

- Questions from 4/15
- Assignment 0
- **C Tutorial - Pointers, Strings, Exec in C**
- Quiz 1 – Active Reading Chapter 9
- Chapter 7: Scheduling Introduction
- Chapter 8: Multi-level Feedback Queue
  - MLFQ Scheduler
  - Job Starvation
  - Gaming the Scheduler
  - Examples
- Chapter 9: Proportional Share Schedulers

| April 17, 2025 | TCSS422: Operating Systems [Spring 2025]<br>School of Engineering and Technology, University of Washington - Tacoma | L6.11 |
|---|---|---|

11

## OBJECTIVES – 4/17

- Questions from 4/15
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- **Quiz 1 – Active Reading Chapter 9**
- Chapter 7: Scheduling Introduction
- Chapter 8: Multi-level Feedback Queue
  - MLFQ Scheduler
  - Job Starvation
  - Gaming the Scheduler
  - Examples
- Chapter 9: Proportional Share Schedulers

| April 17, 2025 | TCSS422: Operating Systems [Spring 2025]<br>School of Engineering and Technology, University of Washington - Tacoma | L6.12 |
|---|---|---|

12

# OBJECTIVES – 4/17

- Questions from 4/15
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Quiz 1 – Active Reading Chapter 9
- **Chapter 7: Scheduling Introduction**
- Chapter 8: Multi-level Feedback Queue
  - MLFQ Scheduler
  - Job Starvation
  - Gaming the Scheduler
  - Examples
- Chapter 9: Proportional Share Schedulers

| April 17, 2025 | TCSS422: Operating Systems [Spring 2025]<br>School of Engineering and Technology, University of Washington - Tacoma | L6.13 |
|---|---|---|

13

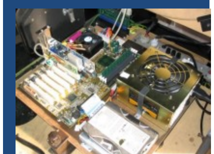
# CATCH UP FROM LECTURE 5

- Switch to Lecture 5 Slides
- Slides L5.48 to L5.52 (thru STCF scheduler)

| April 17, 2025 | TCSS422: Operating Systems [Spring 2025]<br>School of Engineering and Technology, University of Washington - Tacoma | L6.14 |
|---|---|---|

14

# CHAPTER 7- SCHEDULING: INTRODUCTION





15

# CHAPTER 7

- Chapter 7: Scheduling Introduction
  - Scheduling metrics
    - Turnaround time, Jain's Fairness Index, Response time
  - FIFO, SJF, STCF, RR schedulers



16

# 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 time period.

| Process | Burst Time |
|---|---|
| P1 | 12 |

**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

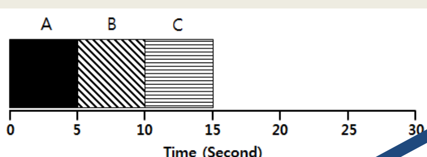


17

# RR EXAMPLE

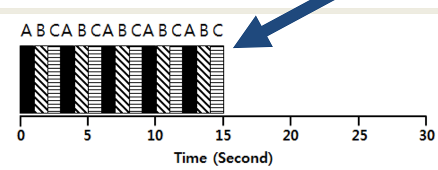
- ABC arrive at time=0, each run for 5 seconds

**OVERHEAD not considered**




SJF (Bad for Response Time)

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




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

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



18

# ROUND ROBIN: TRADEOFFS

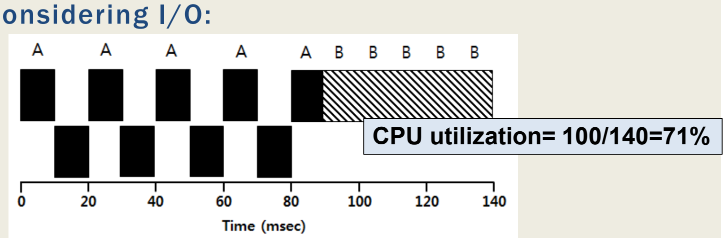
| Short Time Slice | | Long Time Slice |
|---|---|---|
| Fast Response Time | ⟷ | Slow Response Time |
| High overhead from context switching | | Low overhead from context switching |

- 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

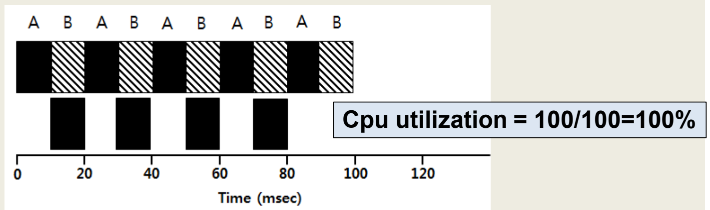
- STCF scheduler
  - A: CPU=50ms, I/O=40ms, 10ms intervals
  - B: CPU=50ms, I/O=0ms
  - Consider A as 10ms subjobs (CPU, then I/O)
- Without considering I/O:




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



21

Respond at **pollev.com/weslloyd**
Text **WESLLOYD** to **22333** once to join, then **1, 2, 3, 4, 5...**

## Which scheduler, thus far, best address fairness and average response time of jobs?

- First In - First Out (FIFO) **1**
- Shortest Job First (SJF) **2**
- Shortest Time to Completion First (STCF) **3**
- Round Robin **4**
- None of the Above **5**
- All of the Above **6**

Total Results: 0

Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

22

## QUESTION: SCHEDULING FAIRNESS

- Which scheduler, this far, best addresses fairness and average response time of jobs?

- First In – First Out (FIFO)
- Shortest Job First (SJF)
- Shortest Time to Completion First (STCF)
- Round Robin (RR)
- None of the Above
- All of the Above



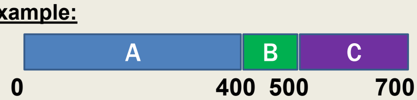
## SCHEDULING METRICS

- Consider Three jobs (A, B, C) that require: $time_A$=400ms, $time_B$=100ms, and $time_C$=200ms

- All jobs arrive at time=0 in the sequence of A B C.

- Draw a scheduling graph to help compute the <u>**average response time (ART)**</u> and <u>**average turnaround time (ATT)**</u> scheduling metrics for the FIFO scheduler.

<u>**Example:**</u>



## What is the Average Response Time of the FIFO scheduler?

Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

25

## What is the Average Turnaround Time of the FIFO scheduler?

Powered by Poll Everywhere

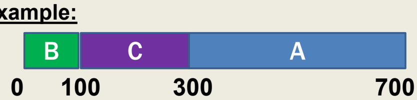
Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

26

# SCHEDULING METRICS

- Consider Three jobs (A, B, C) that require: $time_A$=400ms, $time_B$=100ms, and $time_C$=200ms
- All jobs arrive at time=0 in the sequence of A B C.
- Draw a scheduling graph to help compute the **average response time (ART)** and **average turnaround time (ATT)** scheduling metrics for the SJF scheduler.

**Example:**




| April 17, 2025 | TCSS422: Operating Systems [Spring 2025]<br>School of Engineering and Technology, University of Washington - Tacoma | L6.27 |
|---|---|---|

27

# What is the Average Response Time of the Shortest Job First Scheduler?

Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

28

## What is the Average Turnaround Time of the Shortest Job First Scheduler?

" 7.75 milli "

" 2ms "

" Too long :( "

" 1000 "

Powered by Poll Everywhere

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**

29

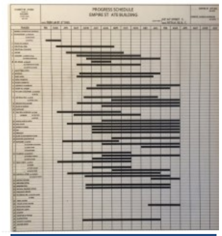
## OBJECTIVES – 4/17

- Questions from 4/15
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Quiz 1 – Active Reading Chapter 9
- Chapter 7: Scheduling Introduction
- Chapter 8: Multi-level Feedback Queue
  - **MLFQ Scheduler**
  - Job Starvation
  - Gaming the Scheduler
  - Examples
- Chapter 9: Proportional Share Schedulers

| April 17, 2025 | TCSS422: Operating Systems [Spring 2025]<br>School of Engineering and Technology, University of Washington - Tacoma | L6.30 |
|---|---|---|

30

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





31

## 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

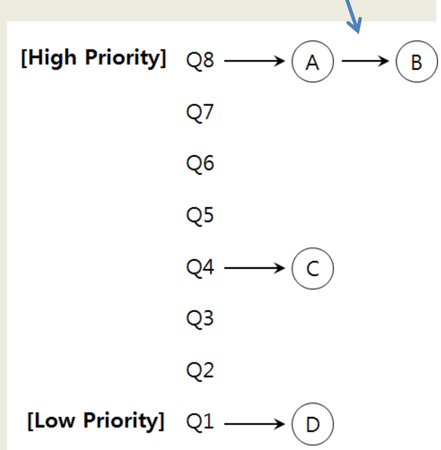


32

# 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






33

# 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**

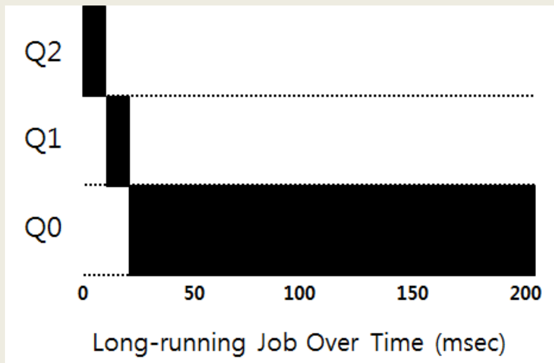


34

# MLFQ: LONG RUNNING JOB

- Three-queue scheduler, time slice=10ms

Priority



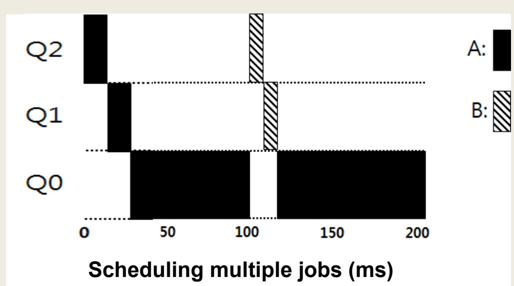



35

# MLFQ: BATCH AND INTERACTIVE JOBS

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

Priority



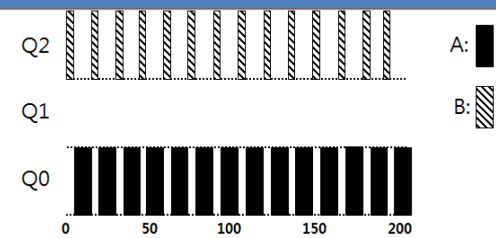



36

## 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

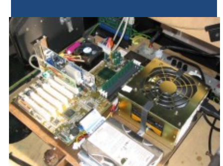
**The MLFQ approach keeps interactive job(s) at the highest priority**




A Mixed I/O-intensive and CPU-intensive Workload (msec)



## WE WILL RETURN AT 5:05PM



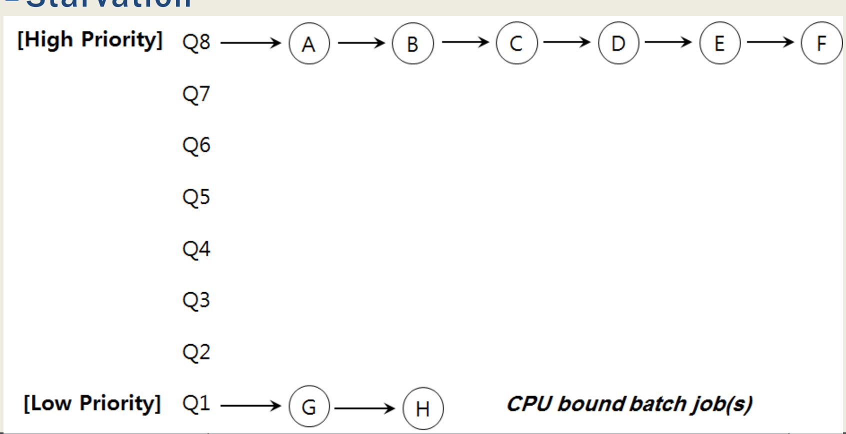
# OBJECTIVES – 4/17

- Questions from 4/15
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Quiz 1 – Active Reading Chapter 9
- Chapter 7: Scheduling Introduction
- Chapter 8: Multi-level Feedback Queue
  - MLFQ Scheduler
  - **Job Starvation**
  - Gaming the Scheduler
  - Examples
- Chapter 9: Proportional Share Schedulers



# MLFQ: ISSUES

- Starvation



# OBJECTIVES – 4/17

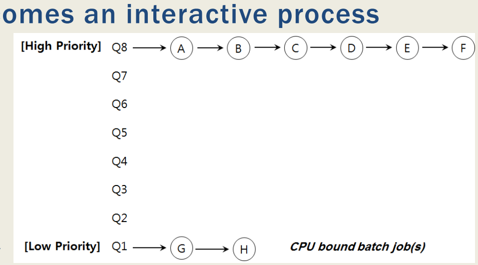
- Questions from 4/15
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Quiz 1 – Active Reading Chapter 9
- Chapter 7: Scheduling Introduction
- Chapter 8: Multi-level Feedback Queue
  - MLFQ Scheduler
  - Job Starvation
  - **Gaming the Scheduler**
  - Examples
- Chapter 9: Proportional Share Schedulers

| April 17, 2025 | TCSS422: Operating Systems [Spring 2025]<br>School of Engineering and Technology, University of Washington - Tacoma | L6.41 |
|---|---|---|

41

# 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

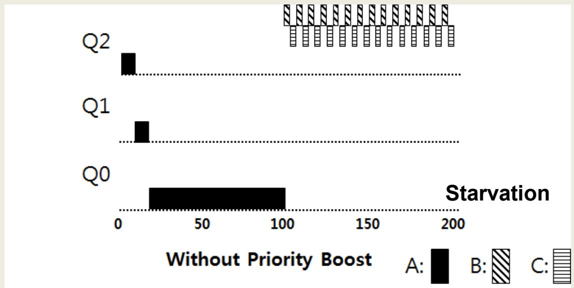



**Priority becomes stuck**

| April 17, 2025 | TCSS422: Operating Systems [Spring 2025]<br>School of Engineering and Technology, University of Washington - Tacoma | L6.42 |
|---|---|---|

42

# RESPONDING TO BEHAVIOR CHANGE



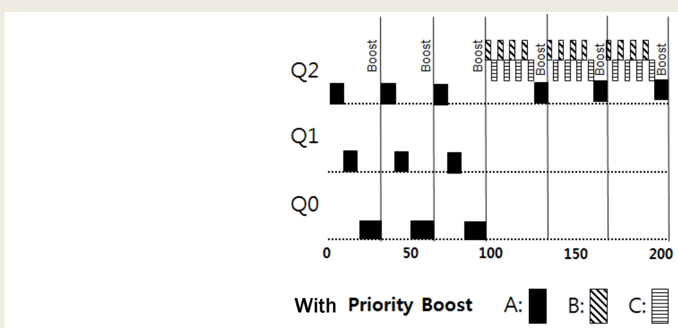

- Priority Boost
  - Reset all jobs to topmost queue after some time interval S

| April 17, 2025 | TCSS422: Operating Systems [Spring 2025]<br>School of Engineering and Technology, University of Washington - Tacoma | L6.43 |
|---|---|---|

43

# RESPONDING TO BEHAVIOR CHANGE - 2

- With priority boost
  - Prevents starvation




| April 17, 2025 | TCSS422: Operating Systems [Spring 2025]<br>School of Engineering and Technology, University of Washington - Tacoma | L6.44 |
|---|---|---|

44

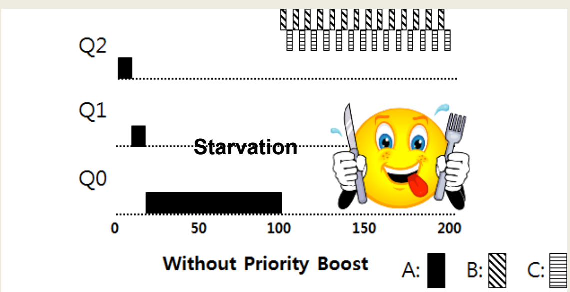
# KEY TO UNDERSTANDING MLFQ – PB

- Without priority boost:
- **Rule 1:** If Priority(A) > Priority(B), A runs (B doesn't).
- **Rule 2:** If Priority(A) = Priority(B), A & B run in RR.
- **KEY**: If time quantum of a higher queue is filled, then we don't run any jobs in lower priority queues!!!



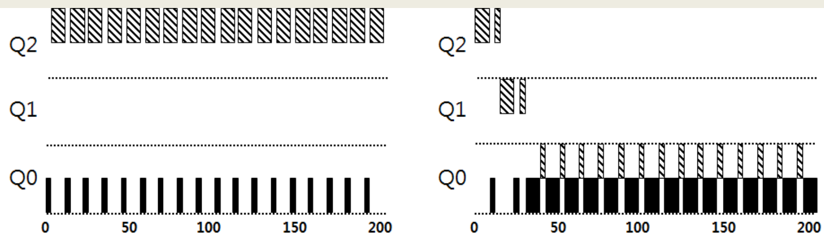
# STARVATION EXAMPLE

- **Consider 3 queues:**
- Q2 – HIGH PRIORITY – Time Quantum 10ms
- Q1 – MEDIUM PRIORITY – Time Quantum 20 ms
- Q0 – LOW PRIORITY – Time Quantum 40 ms
- Job A: 200ms no I/O
- Job B: 5ms then I/O
- Job C: 5ms then I/O
- Q2 fills up, starves Q1 & Q0
- A makes no progress



# 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

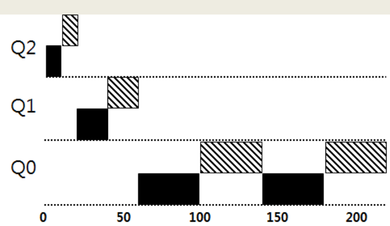



Without(Left) and With(Right) Gaming Tolerance



# 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?




Example) 10ms for the highest queue, 20ms for the middle, 40ms for the lowest

# 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.

# OBJECTIVES – 4/17

- Questions from 4/15
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Quiz 1 – Active Reading Chapter 9
- Chapter 7: Scheduling Introduction
- Chapter 8: Multi-level Feedback Queue
  - MLFQ Scheduler
  - Job Starvation
  - Gaming the Scheduler
  - **Examples**
- Chapter 9: Proportional Share Schedulers



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.

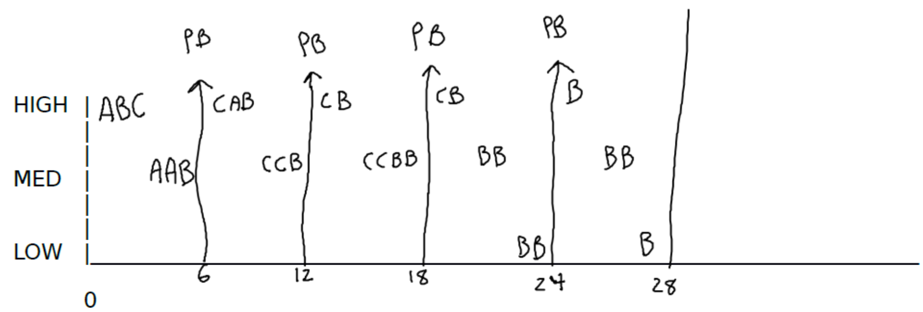
| Job | Arrival Time | Job Length |
| --- | --- | --- |
| A | T=0 | 4 |
| B | T=0 | 16 |
| C | T=0 | 8 |

(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.

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 | |
| B | T=0 | |
| C | T=0 | |

(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.

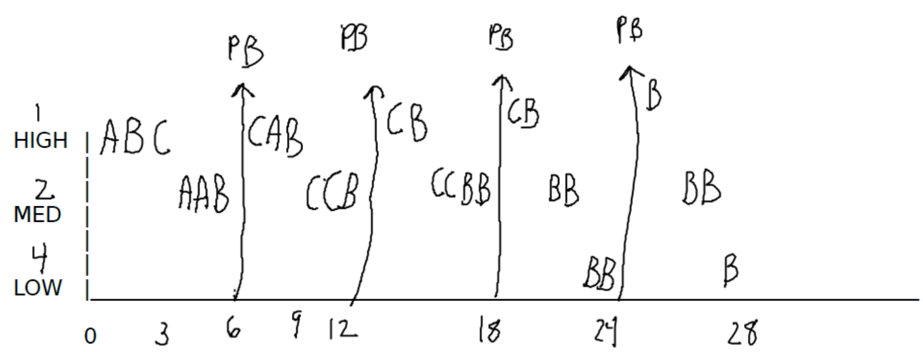



53

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 | |
| B | T=0 | |
| C | T=0 | |

(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.




54

# EXAMPLE

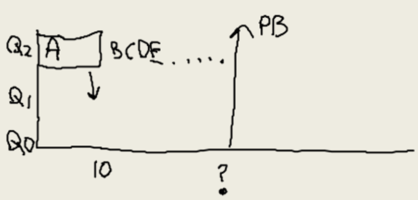
- Question:
- Given a system with a total quantum length of 10 ms ***for all jobs*** to run before priority is lowered in the highest queue, how often would you have to boost jobs back to the highest priority level to guarantee that a single long-running (and potentially starving) job gets at least 5% of the CPU?

| April 17, 2025 | TCSS422: Operating Systems [Spring 2025]<br>School of Engineering and Technology, University of Washington - Tacoma | L6.55 |
|---|---|---|

55

# EXAMPLE

- Question:
- Given a system with a quantum length of 10 ms ***for all jobs*** to run before priority is lowered in the highest queue, how often would you have to boost jobs back to the highest priority level to guarantee that a single long-running (and potentially starving) job gets at least 5% of the CPU?




$.05\ PB = 10$

$PB = \frac{10}{.05} = 200$ ms

| April 17, 2025 | TCSS422: Operating Systems [Spring 2025]<br>School of Engineering and Technology, University of Washington - Tacoma | L6.56 |
|---|---|---|

56

# EXAMPLE

- Question:
- Given a system with a quantum length of 10 ms ***for all jobs*** in its highest queue, how often would you have to boost jobs back to the highest priority level to guarantee that a single long-running (and potentially starving) job gets at least 5% of the CPU?

- Some combination of n jobs runs for a total of 10 ms per cycle without relinquishing the CPU
  - E.g. 2 jobs = 5 ms ea; 3 jobs = 3.33 ms ea, 10 jobs = 1 ms ea
  - n jobs always uses full time quantum in highest queue (10 ms)
  - Batch jobs starts, runs for full quantum of 10ms, pushed to lower queue
  - All other jobs run and context switch totaling the quantum per cycle
  - If 10ms is 5% of the CPU, when must the priority boost be ???
  - **ANSWER** → ***Priority boost should occur every 200ms***



# OBJECTIVES – 4/17

- Questions from 4/15
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Quiz 1 – Active Reading Chapter 9
- Chapter 7: Scheduling Introduction
- Chapter 8: Multi-level Feedback Queue
  - MLFQ Scheduler
  - Job Starvation
  - Gaming the Scheduler
  - Examples
- **Chapter 9: Proportional Share Schedulers**

# CHAPTER 9 - PROPORTIONAL SHARE SCHEDULER



59

## OBJECTIVES – 4/17

- Chapter 9: Proportional Share Schedulers
  - Lottery scheduler
  - Ticket mechanisms
  - Stride scheduler
  - Linux Completely Fair Scheduler



60

# PROPORTIONAL SHARE SCHEDULER

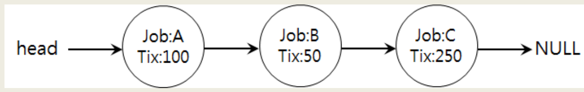
- Also called fair-share scheduler
  or lottery scheduler
  - Guarantees each job receives some percentage of CPU time based on share of "tickets"
  - Each job receives an allotment of tickets
  - % of tickets corresponds to potential share of a resource
  - Can conceptually schedule any resource this way
    - CPU, disk I/O, memory



# LOTTERY SCHEDULER

- Simple implementation
  - Just need a random number generator
    - Picks the winning ticket
  - Maintain a data structure of jobs and tickets (list)
  - Traverse list to find the owner of the ticket
  - Consider sorting the list for speed

# LOTTERY SCHEDULER IMPLEMENTATION




```
// counter: used to track if we've found the winner yet
int counter = 0;

// winner: use some call to a random number generator to
// get a value, between 0 and the total # of tickets
int winner = getrandom(0, totaltickets);

// current: use this to walk through the list of jobs
node_t *current = head;

// loop until the sum of ticket values is > the winner
while (current) {
        counter = counter + current->tickets;
        if (counter > winner)
                break; // found the winner
        current = current->next;
}
// 'current' is the winner: schedule it...
```

| April 17, 2025 | TCSS422: Operating Systems [Spring 2025]<br>School of Engineering and Technology, University of Washington - Tacoma | L6.63 |
|---|---|---|

63

# OBJECTIVES – 4/17

- **Chapter 9: Proportional Share Schedulers**
  - **Lottery scheduler**
  - **Ticket mechanisms**
  - **Stride scheduler**
  - **Linux Completely Fair Scheduler**

| April 17, 2025 | TCSS422: Operating Systems [Spring 2025]<br>School of Engineering and Technology, University of Washington - Tacoma | L6.64 |
|---|---|---|

64

## TICKET MECHANISMS

- Ticket currency / exchange
  - User allocates tickets in any desired way
  - OS converts user currency into global currency
- Example:
  - There are 200 global tickets assigned by the OS

| | |
|---|---|
| **User A** | → *500* (A's currency) to A1 → *50* (global currency) |
| | → *500* (A's currency) to A2 → *50* (global currency) |
| **User B** | → *10* (B's currency) to B1 → *100* (global currency) |



## TICKET MECHANISMS - 2

- Ticket transfer
  - Temporarily hand off tickets to another process
- Ticket inflation
  - Process can temporarily raise or lower the number of tickets it owns
  - If a process needs more CPU time, it can boost tickets.

# LOTTERY SCHEDULING

- Scheduler picks a **winning** ticket
  - Load the job with the winning ticket and run it
- Example:
  - Given 100 tickets in the pool
    - Job A has 75 tickets: 0 - 74
    - Job B has 25 tickets: 75 – 99

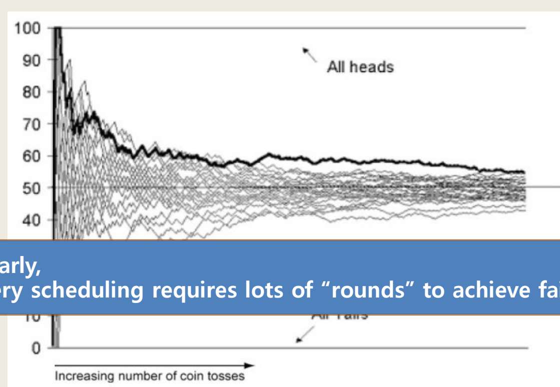
| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Scheduler's winning tickets:** | 63 | 85 | 70 | 39 | 76 | 17 | 29 | 41 | 36 | 39 | 10 | 99 | 68 | 83 | 63 |
| **Scheduled job:** | A | B | A | A | B | A | A | A | A | A | A | B | A | B | A |

- But what do we know about probability of a coin flip?



# COIN FLIPPING

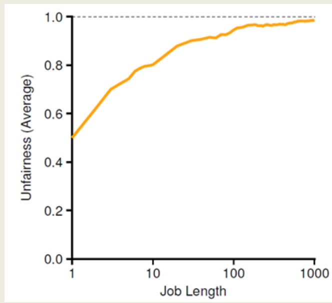
- Equality of distribution (fairness) requires a lot of flips!




Similarly,
Lottery scheduling requires lots of "rounds" to achieve fairness.

# LOTTERY FAIRNESS

- With two jobs
  - Each with the same number of tickets (t=100)




When the job length is not very long, average unfairness can be **quite severe**.



# LOTTERY SCHEDULING CHALLENGES

- What is the best approach to assign tickets to jobs?
  - Typical approach is to assume users know best
  - Users are provided with tickets, which they allocate as desired

- How should the OS automatically distribute tickets upon job arrival?
  - What do we know about incoming jobs a priori ?
  - Ticket assignment is really an open problem…

# OBJECTIVES – 4/17

- Chapter 9: Proportional Share Schedulers
  - Lottery scheduler
  - Ticket mechanisms
  - **Stride scheduler**
  - Linux Completely Fair Scheduler



# STRIDE SCHEDULER

- Addresses statistical probability issues with lottery scheduling
- Instead of guessing a random number to select a job, simply count…

# STRIDE SCHEDULER - 2

- Jobs have a "stride" value
  - A stride value describes the counter pace when the job should give up the CPU
  - Stride value is **inverse in proportion** to the job's number of tickets (more tickets = smaller stride)
- Total system tickets = 10,000
  - Job A has 100 tickets → $A_{stride}$ = 10000/100 = 100 stride
  - Job B has 50 tickets → $B_{stride}$ = 10000/50 = 200 stride
  - Job C has 250 tickets → $C_{stride}$ = 10000/250 = 40 stride
- Stride scheduler tracks "pass" values for each job (A, B, C)

| April 17, 2025 | TCSS422: Operating Systems [Spring 2025]<br>School of Engineering and Technology, University of Washington - Tacoma | L6.73 |
|---|---|---|

73

# STRIDE SCHEDULER - 3

- Basic algorithm:
  1. Stride scheduler picks job with the lowest pass value
  2. Scheduler increments job's pass value by its stride and starts running
  3. Stride scheduler increments a counter
  4. When counter exceeds pass value of current job, pick a new job (go to 1)
- KEY: When the counter reaches a job's "PASS" value, the scheduler passes on to the next job…

| April 17, 2025 | TCSS422: Operating Systems [Spring 2025]<br>School of Engineering and Technology, University of Washington - Tacoma | L6.74 |
|---|---|---|

74

# STRIDE SCHEDULER - EXAMPLE

- Stride values
  - Tickets = priority to select job
  - Stride is inverse to tickets
  - Lower stride = more chances to run (higher priority)

Priority
C stride = 40
A stride = 100
B stride = 200



# STRIDE SCHEDULER EXAMPLE - 2

- Three-way tie: randomly pick job A (all pass values=0)
- Set A's pass value to A's stride = 100
- Increment counter until > 100
- Pick a new job: two-way tie

| Pass(A) (stride=100) | Pass(B) (stride=200) | Pass(C) (stride=40) | Who Runs? |
|---|---|---|---|
| 0 | 0 | 0 | A |
| 100 | 0 | 0 | B |
| 100 | 200 | 0 | C |
| 100 | 200 | 40 | C |
| 100 | 200 | 80 | C |
| 100 | 200 | 120 | A |
| 200 | 200 | 120 | C |
| 200 | 200 | 160 | C |
| 200 | 200 | 200 | … |

Tickets
C = 250
A = 100
B = 50

Initial job selection is random. All @ 0

C has the most tickets and receives a lot of opportunities to run…

# STRIDE SCHEDULER EXAMPLE - 3

- We set A's counter (pass value) to A's stride = 100
- Next scheduling decision between B (pass=0) and C (pass=0)
  - Randomly choose B
- C has the lowest counter for next 3 rounds

**Tickets**
C = 250
A = 100
B = 50

| Pass(A) (stride=100) | Pass(B) (stride=200) | Pass(C) (stride=40) | Who Runs? |
|---|---|---|---|
| 0 | 0 | 0 | A |
| 100 | 0 | 0 | B |
| 100 | 200 | 0 | C |
| 100 | 200 | 40 | C |
| 100 | 200 | 80 | C |
| 100 | 200 | 120 | A |
| 200 | 200 | 120 | C |
| 200 | 200 | 160 | C |
| 200 | 200 | 200 | … |

C has the most tickets and is selected to run more often …



77

# STRIDE SCHEDULER EXAMPLE - 4

- Job counters support determining which job to run next
- Over time jobs are scheduled to run based on their priority represented as their **share of tickets…**
- **Tickets are analogous to job priority**

**Tickets**
C = 250
A = 100
B = 50

| Pass(A) (stride=100) | Pass(B) (stride=200) | Pass(C) (stride=40) | Who Runs? |
|---|---|---|---|
| 0 | 0 | 0 | A |
| 100 | 0 | 0 | B |
| 100 | 200 | 0 | C |
| 100 | 200 | 40 | C |
| 100 | 200 | 80 | C |
| 100 | 200 | 120 | A |
| 200 | 200 | 120 | C |
| 200 | 200 | 160 | C |
| 200 | 200 | 200 | … |



78

# OBJECTIVES – 4/17

- Chapter 9: Proportional Share Schedulers
  - Lottery scheduler
  - Ticket mechanisms
  - Stride scheduler
  - **Linux Completely Fair Scheduler**

| April 17, 2025 | TCSS422: Operating Systems [Spring 2025]<br>School of Engineering and Technology, University of Washington - Tacoma | L6.79 |
|---|---|---|

79

# LINUX: COMPLETELY FAIR SCHEDULER (CFS)

- Large Google datacenter study: *"Profiling a Warehouse-scale Computer"* (Kanev et al.)
- Monitored 20,000 servers over 3 years
- Found 20% of CPU time spent in the Linux kernel
- 5% of CPU time spent in the CPU scheduler!
- Study highlights importance for high performance OS kernels and CPU schedulers !

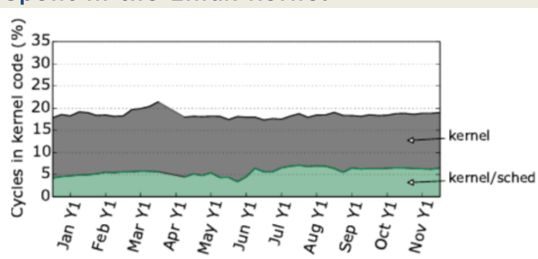



**Figure 5: Kernel time, especially time spent in the scheduler, is a significant fraction of WSC cycles.**

See: https://dl.acm.org/doi/pdf/10.1145/2749469.2750392

| April 17, 2025 | TCSS422: Operating Systems [Spring 2025]<br>School of Engineering and Technology, University of Washington - Tacoma | L6.80 |
|---|---|---|

80

## LINUX: COMPLETELY FAIR SCHEDULER (CFS)

- Loosely based on the stride scheduler
- CFS models system as a Perfect Multi-Tasking System
  - In perfect system every process of the same priority (class) receive exactly $1/n^{th}$ of the CPU time
- Each scheduling class has a runqueue
  - Groups process of same class
  - In class, scheduler picks task w/ lowest `vruntime` to run
  - Time slice varies based on how many jobs in shared runqueue
  - Minimum time slice prevents too many context switches (e.g. 3 ms)

| April 17, 2025 | TCSS422: Operating Systems [Spring 2025]<br>School of Engineering and Technology, University of Washington - Tacoma | L6.81 |
|---|---|---|

81

## COMPLETELY FAIR SCHEDULER - 2

- Every thread/process has a scheduling class (policy):
- **Normal classes**: SCHED_OTHER (TS), SCHED_IDLE, SCHED_BATCH
  - TS = Time Sharing
- **Real-time classes**: SCHED_FIFO (FF), SCHED_RR (RR)
- How to show scheduling class and priority:
- #class

```
ps -elfc
```

- #priority (nice value)

```
ps ax -o pid,ni,cls,pri,cmd
```

| April 17, 2025 | TCSS422: Operating Systems [Spring 2025]<br>School of Engineering and Technology, University of Washington - Tacoma | L6.82 |
|---|---|---|

82

## COMPLETELY FAIR SCHEDULER - 3

- Linux ≥ 2.6.23: Completely Fair Scheduler (CFS)
- Linux < 2.6.23: O(1) scheduler

- Linux maintains simple counter (vruntime) to track how long each thread/process has run
- CFS picks process with lowest vruntime to run next

- CFS adjusts timeslice based on # of proc waiting for the CPU
- Kernel parameters that specify CFS behavior:

```
$ sudo sysctl kernel.sched_latency_ns
kernel.sched_latency_ns = 24000000
$ sudo sysctl kernel.sched_min_granularity_ns
kernel.sched_min_granularity_ns = 3000000
$ sudo sysctl kernel.sched_wakeup_granularity_ns
kernel.sched_wakeup_granularity_ns = 4000000
```



## COMPLETELY FAIR SCHEDULER - 4

- `Sched_min_granularity_ns` (3ms)
  - Time slice for a process: busy system (w/ full runqueue)
  - If system has idle capacity, time slice exceed the min as long as difference in `vruntime` between running process and process with lowest `vruntime` is less than `sched_wakeup_granularity_ns` (4ms)
- Scheduling time period is: total cycle time for iterating through a set of processes where each is allowed to run (like round robin)
- Example:
  `sched_latency_ns` (24ms)
  if (proc in runqueue < `sched_latency_ns/sched_min_granularity`)
  or
  `sched_min_granularity` * number of processes in runqueue

Ref: https://www.systutorials.com/sched_min_granularity_ns-sched_latency_ns-cfs-affect-timeslice-processes/

# CFS TRADEOFF

- **HIGH** sched_min_granularity_ns (timeslice)
  sched_latency_ns
  sched_wakeup_granularity_ns

  reduced context switching → less overhead
  poor near-term fairness

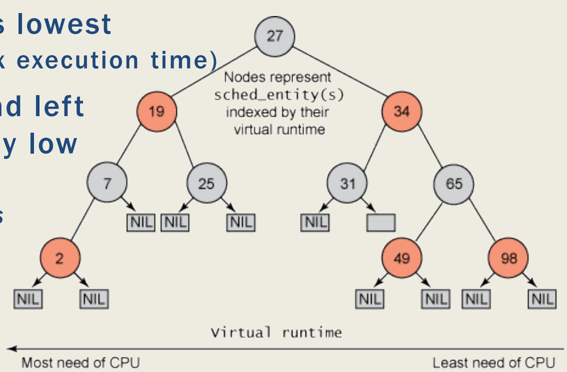
- **LOW** sched_min_granularity_ns (timeslice)
  sched_latency_ns
  sched_wakreup_granularity_ns

  increased context switching → more overhead
  better near-term fairness



# COMPLETELY FAIR SCHEDULER - 5

- Runqueues are stored using a linux red-black tree
  - Self balancing binary tree - nodes indexed by `vruntime`
- Leftmost node has lowest `vruntime` (approx execution time)
- Walking tree to find left most node has very low big O complexity: *~O(log N) for N nodes*
- Completed processes removed



## CFS: JOB PRIORITY

```
static const int prio_to_weight[40] = {
    /* -20 */ 88761, 71755, 56483, 46273, 36291,
    /* -15 */ 29154, 23254, 18705, 14949, 11916,
    /* -10 */  9548,  7620,  6100,  4904,  3906,
    /*  -5 */  3121,  2501,  1991,  1586,  1277,
    /*   0 */  1024,   820,   655,   526,   423,
    /*   5 */   335,   272,   215,   172,   137,
    /*  10 */   110,    87,    70,    56,    45,
    /*  15 */    36,    29,    23,    18,    15,
};
```

- Time slice: Linux ***"Nice value"***
  - Nice predates the CFS scheduler
  - Top shows nice values
  - Process command (nice & priority):
    `ps ax -o pid,ni,cmd,%cpu, pri`
- Nice Values: from -20 to 19
  - Lower is ***higher*** priority, default is 0
  - Vruntime is a weighted time measurement
  - Priority weights the calculation of vruntime within a runqueue to give high priority jobs a boost.
    - Influences job's position in rb-tree

| April 17, 2025 | TCSS422: Operating Systems [Spring 2025]<br>School of Engineering and Technology, University of Washington - Tacoma | L6.87 |
|---|---|---|

87

## COMPLETELY FAIR SCHEDULER - 6

- CFS tracks cumulative job run time in `vruntime` variable
- The task on a given runqueue with the lowest `vruntime` is scheduled next
- `struct sched_entity` contains `vruntime` parameter
  - Describes process execution time in nanoseconds
  - Value is not pure runtime, is weighted based on job priority
  - Perfect scheduler →
    achieve equal `vruntime` for all processes of same priority
- Sleeping jobs: upon return reset vruntime to lowest value in system
  - Jobs with frequent short sleep ***SUFFER !!***
- Key takeaway:
  ***identifying the next job to schedule is really fast!***

| April 17, 2025 | TCSS422: Operating Systems [Spring 2025]<br>School of Engineering and Technology, University of Washington - Tacoma | L6.88 |
|---|---|---|

88

## COMPLETELY FAIR SCHEDULER - 7

- More information:
- Man page: "man sched" : Describes Linux scheduling API
- http://manpages.ubuntu.com/manpages/bionic/man7/sched.7.html
- https://www.kernel.org/doc/Documentation/scheduler/sched-design-CFS.txt
- https://en.wikipedia.org/wiki/Completely_Fair_Scheduler
- See paper: The Linux Scheduler – a Decade of Wasted Cores
- http://www.ece.ubc.ca/~sasha/papers/eurosys16-final29.pdf



89

## QUESTIONS

90