TCSS 422: OPERATING SYSTEMS

Multi-level Feedback Queue II, Proportional Share Schedulers, Linux Completely Fair Scheduler

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

1

TEXT BOOK COUPON

- 15% off textbook code: **HAPPYPLANET15** (*through Friday Apr 25*)
- 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 $33.79 + 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
  - THIS WEEK: 5:30 to 6:30pm CP 229 & Zoom Instructor Wes
- Friday 12:00pm - 1:00p Instructor Wes or GTA Xinghan

- Office hours this Friday April 18th
  - Wes

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

3

BONUS SESSION – CPU SCHEDULING PROBLEMS

- To help prepare for quiz 1 and the midterm

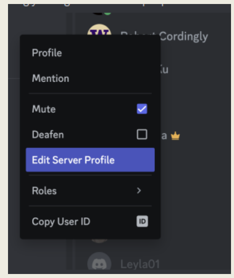
- Wednesday April 23, 4 to 5pm
- MLG 311 and live-streamed on Zoom
- Recording will be posted

- Sample problems will be solved
- Sample problems are posted online:
- https://faculty.washington.edu/wlloyd/courses/tcss422/scheduler_examples_s2025.pdf

4

TCSS 422 DISCORD SERVER

- Please join the TCSS 422 A – Spring 2025 Discord Server
- https://discord.gg/H7PPZ5ArFW
- Under Edit Server Profile:
  Please update your 'Server Nickname' to your real name or UW NET ID
  THANK YOU



5

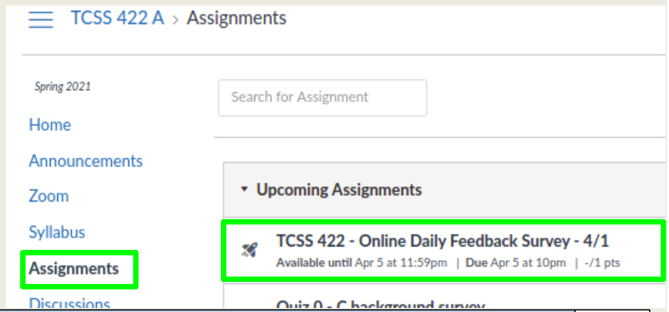
OBJECTIVES – 4/22

- Questions from 4/17
- Assignment 0 - Due Fri Apr 26
- C Tutorial - Pointers, Strings, Exec in C - Due Fri Apr 30
- Quiz 1 and Quiz 2
- Chapter 8: Multi-level Feedback Queue
  - Gaming the Scheduler
  - Examples
- Chapter 9: Proportional Share Schedulers
  - Lottery scheduler
  - Ticket mechanisms
  - Stride scheduler
  - Linux Completely Fair Scheduler
- Chapter 26: Concurrency: An Introduction
  - Introduction
  - Race condition
  - Critical section

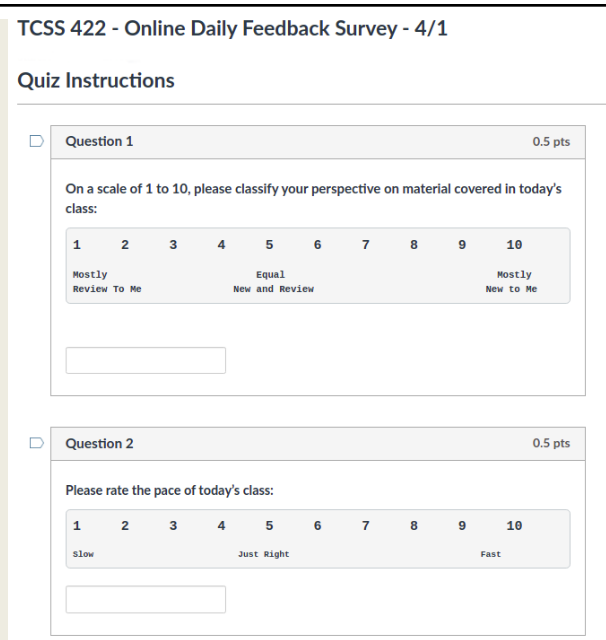
6

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



7



8

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (44 of 63 respondents – 69.8%) :
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 6.32 (↓ - previous 6.88)**

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 4.98 (↓ - previous 5.05)**

FEEDBACK FROM 4/17

- ***Round-robin schedulers are excellent for providing low job response time - but they sacrifice job turnaround time.***

- ***For process scheduling, why is it generally better for schedulers to satisfy both response time and turnaround time?***
  - This is how the Multi-level Feedback Queue (MLFQ) improves on the round-robin scheduler

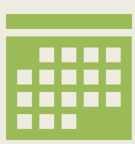
- ***IDEA: Could schedulers distribute tasks to dedicated processors with specialized roles for handling interactive jobs and batch (long-running & high-CPU-load) jobs?***

FEEDBACK - 2

- ***How is the MLFQ itself scheduled within the OS ?***
  Is the scheduler assigned a dedicated time slice to prevent a higher priority job from keeping it from running ?
- ***How can we be sure that priority boosts and job priority adjustments can continue to occur ?***

- Remember that preemptive multitasking operating systems feature a timer interrupt. This interrupt is setup at boot time. The timer fires every ~2 to 10ms to perform a non-voluntary context switch. Also when the user performs I/O, this creates voluntary context switches.
- These context switches allow the system to take over, and the OS kernel can run the MLFQ scheduler to manage jobs running on the system
  - MLFQ will priority boost, adjust job priority among queues, etc.

REVIEW



In the context of process scheduling, what is a "batch job" ?

What is an "interactive job" ?

FEEDBACK - 3

- ***Which of these scheduler's do today's operating systems use?***
- Chapter 9 introduces the **Linux Completely Fair Scheduler (CFS)**
- CFS was used until version 6.5 of the Linux kernel
- Starting with Linux kernel 6.6+ (Oct 29 2023), CFS was replaced with the **Earliest Eligible Virtual Deadline First (EEVDF)** scheduler
- Ubuntu 24.04 LTS launched with the 6.8 Linux Kernel, and is now using 6.11 this quarter which is EEVDF
- We will not test on the EEVDF scheduler since it is not in the textbook
  - I created 1 slide on EEVDF at the end of Chapter 9

April 22, 2025	TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma	L7.13

13

FEEDBACK - 4

- ***What is going to be on the quiz ?***
- ***What is going to be on the quiz ??***
- ***What is going to be on the quiz ???***
- ***What is going to be on the quiz ????***
- ***What is going to be on the quiz ?????***
- ***What is going to be on the quiz ??????***
- ***What is going to be on the quiz ???????***
- ***What is going to be on the quiz ????????***
- ***What is going to be on the quiz ?????????***
- ***What is going to be on the quiz ??????????***
- ***What is going to be on the quiz ???????????***

April 22, 2025	TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma	L7.14

14

OBJECTIVES – 4/22

- Questions from 4/17
- **Assignment 0 - Due Fri Apr 26**
- C Tutorial - Pointers, Strings, Exec in C - Due Fri Apr 30
- Quiz 1 and Quiz 2
- Chapter 8: Multi-level Feedback Queue
  - Gaming the Scheduler
  - Examples
- Chapter 9: Proportional Share Schedulers
  - Lottery scheduler
  - Ticket mechanisms
  - Stride scheduler
  - Linux Completely Fair Scheduler
- Chapter 26: Concurrency: An Introduction
  - Introduction
  - Race condition
  - Critical section

ASSIGNMENT 0 - DUE FRI APR 26

- Due Friday April 26 AOE (Apr 27 4:59am)
- Grace period: submission ok until Tue Apr 29 @ 4:59 **AM**
- Late submissions thru Tuesday Thur May 1 @ 4:59am

OBJECTIVES – 4/22

- Questions from 4/17
- Assignment 0 - Due Fri Apr 26
- C Tutorial - Pointers, Strings, Exec in C - Due Fri Apr 30
- Quiz 1 and Quiz 2
- Chapter 8: Multi-level Feedback Queue
  - Gaming the Scheduler
  - Examples
- Chapter 9: Proportional Share Schedulers
  - Lottery scheduler
  - Ticket mechanisms
  - Stride scheduler
  - Linux Completely Fair Scheduler
- Chapter 26: Concurrency: An Introduction
  - Introduction
  - Race condition
  - Critical section

April 22, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L7.17

17

OBJECTIVES – 4/22

- Questions from 4/17
- Assignment 0 - Due Fri Apr 26
- C Tutorial - Pointers, Strings, Exec in C - Due Fri Apr 30
- Quiz 1 and Quiz 2
- Chapter 8: Multi-level Feedback Queue
  - Gaming the Scheduler
  - Examples
- Chapter 9: Proportional Share Schedulers
  - Lottery scheduler
  - Ticket mechanisms
  - Stride scheduler
  - Linux Completely Fair Scheduler
- Chapter 26: Concurrency: An Introduction
  - Introduction
  - Race condition
  - Critical section

April 22, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L7.18

18

QUIZ 1

- Active reading on Chapter 9 – Proportional Share Schedulers
- Posted in Canvas
- Due Thursday May 1st AOE (Fri May 2 4:59 am)
- **Link:**
- https://faculty.washington.edu/wlloyd/courses/tcss422/quiz/TCSS422_s2025_quiz_1.pdf

19

QUIZ 2

- Canvas Quiz – Practice CPU Scheduling Problems
- Posted in Canvas
- Unlimited attempts
- Due Tuesday May 6th AOE (May 7th at 4:59am)
- **Link:**
- https://canvas.uw.edu/courses/1809484/assignments/10329061

20

COMING SOON...

- Assignment #1
  - To be posted soon

- Quiz 1
  - Thursday April 24th
  - In Class – BHS 106
  - 4:40 – 5:40 pm
  - Open notes, open books, no digital devices

- Midterm Exam
  - Thursday May 8th
  - In Class – BHS 106
  - 3:40 – 5:40 pm
  - 3 pages of notes double-sides, no digital devices

CATCH UP FROM LECTURE 6

- Switch to Lecture 6 Slides
- Slides L6.43 to L6.50 (MLFQ priority boost, MLFQ summary)

OBJECTIVES – 4/22

- Questions from 4/17
- Assignment 0 - Due Fri Apr 26
- C Tutorial - Pointers, Strings, Exec in C - Due Fri Apr 30
- Quiz 1 and Quiz 2
- Chapter 8: Multi-level Feedback Queue
  - Gaming the Scheduler
  - **Examples**
- Chapter 9: Proportional Share Schedulers
  - Lottery scheduler
  - Ticket mechanisms
  - Stride scheduler
  - Linux Completely Fair Scheduler
- Chapter 26: Concurrency: An Introduction
  - Introduction
  - Race condition
  - Critical section

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.

***SANITY CHECK*: Consider the timing graph x-axis should not exceed the combined job length of all jobs.**

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

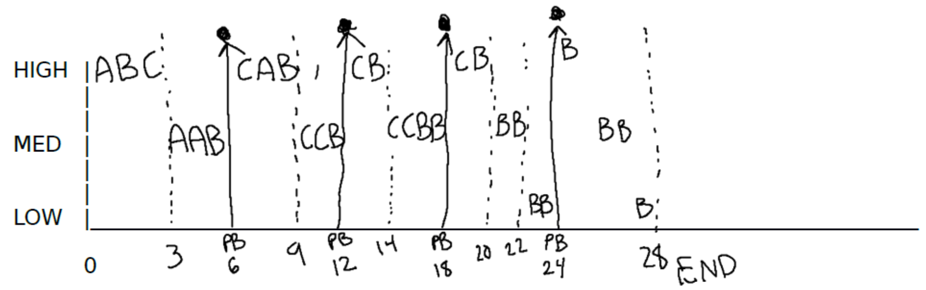
(11 points) Show a scheduling graph for the MLFQ scheduler for the jobs above.
Draw vertical lines for key events and be sure to label the X-axis times as in the example.
Please draw clearly. An unreadable graph will loose points.

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.

***SANITY CHECK*: Consider the timing graph x-axis should not exceed the combined job length of all jobs.**

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.



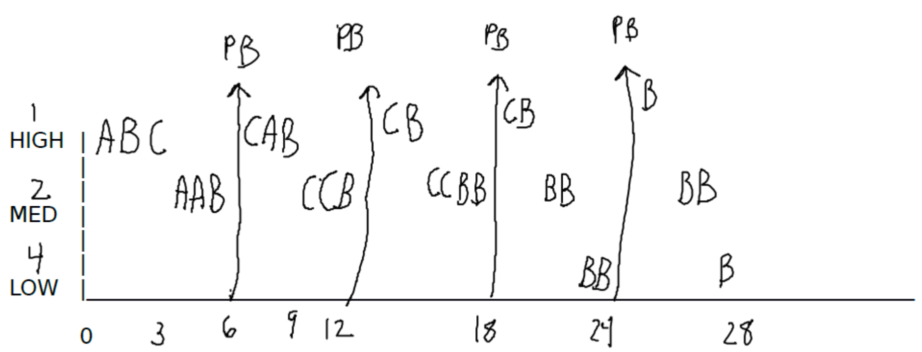
25

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	

time slice is JOB time Before c/s

(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.



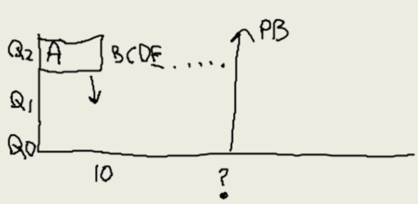
26

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 job A (the first job to arrive and run) back to the highest priority level to guarantee that job A, a long-running (and potentially starving) job gets at least 5% of the CPU assuming that on priority boost job execution resets to the front of the queue?

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?



$$.05\ PB = 10$$

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

EXAMPLE

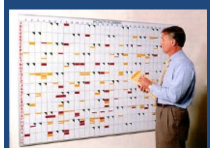
- Question:
- Given a system with a quantum length of 10 ms ***for all jobs*** in its highest queue, how often would you have to boost jobs back to the highest priority level to guarantee that a single long-running (and potentially starving) job gets at least 5% of the CPU?

- Some combination of n short 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 – 4/22

- Questions from 4/17
- Assignment 0 - Due Fri Apr 26
- C Tutorial - Pointers, Strings, Exec in C - Due Fri Apr 30
- Quiz 1 and Quiz 2
- Chapter 8: Multi-level Feedback Queue
  - Gaming the Scheduler
  - Examples
- Chapter 9: Proportional Share Schedulers
  - **Lottery scheduler**
  - Ticket mechanisms
  - Stride scheduler
  - Linux Completely Fair Scheduler
- Chapter 26: Concurrency: An Introduction
  - Introduction
  - Race condition
  - Critical section

CHAPTER 9 - PROPORTIONAL SHARE SCHEDULER



April 22, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L7.31

31

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

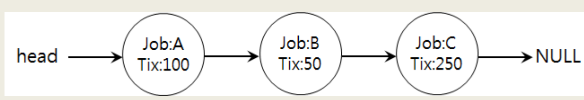
April 22, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L7.32

32

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...
```

OBJECTIVES – 4/22

- Questions from 4/17
- Assignment 0 - Due Fri Apr 26
- C Tutorial - Pointers, Strings, Exec in C - Due Fri Apr 30
- Quiz 1 and Quiz 2
- Chapter 8: Multi-level Feedback Queue
  - Gaming the Scheduler
  - Examples
- Chapter 9: Proportional Share Schedulers
  - Lottery scheduler
  - **Ticket mechanisms**
  - Stride scheduler
  - Linux Completely Fair Scheduler
- Chapter 26: Concurrency: An Introduction
  - Introduction
  - Race condition
  - Critical section

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

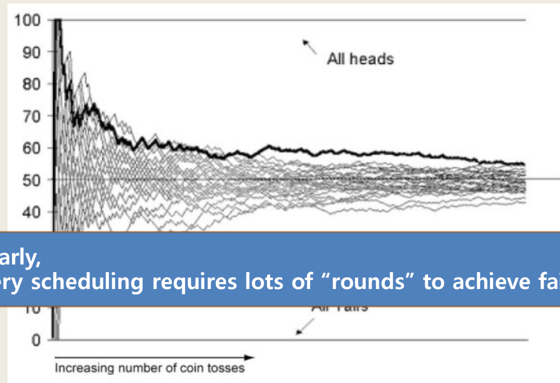
- Scheduler picks a **winning** ticket
  - Load the job with the winning ticket and run it
- Example:
  - Given 100 tickets in the pool
    - Job A has 75 tickets: 0 - 74
    - Job B has 25 tickets: 75 – 99

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!



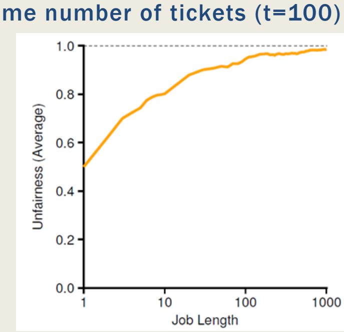
Similarly,
Lottery scheduling requires lots of "rounds" to achieve fairness.

April 22, 2025	TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma	L7.39

39

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**.

April 22, 2025	TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma	L7.40

40

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 – 4/22

- Questions from 4/17
- Assignment 0 - Due Fri Apr 26
- C Tutorial - Pointers, Strings, Exec in C - Due Fri Apr 30
- Quiz 1 and Quiz 2
- Chapter 8: Multi-level Feedback Queue
  - Gaming the Scheduler
  - Examples
- Chapter 9: Proportional Share Schedulers
  - Lottery scheduler
  - Ticket mechanisms
  - **Stride scheduler**
  - Linux Completely Fair Scheduler
- Chapter 26: Concurrency: An Introduction
  - Introduction
  - Race condition
  - Critical section

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)

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…

45

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

46

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

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	…

Initial job selection is random. All @ 0

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

47

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 …

48

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	…

49

Which of the following is NOT a problem with proportional share schedulers?

A. How tickets should be distributed to incoming jobs

B. Lottery scheduler is only eventually fair

C. Given 2 users A and B who both receive a 50% timeshare of the system, the runtime for User A's jobs is dependent on the runtime of User B's.

D. All of the above

E. None of the above

50

WE WILL RETURN AT 5:02PM

April 22, 2025 TCSS422: Operating Systems [Spring 2025]
School of Engineering and Technology, University of Washington - coma L7.51

51

OBJECTIVES – 4/22

- Questions from 4/17
- Assignment 0 - Due Fri Apr 26
- C Tutorial - Pointers, Strings, Exec in C - Due Fri Apr 30
- Quiz 1 and Quiz 2
- Chapter 8: Multi-level Feedback Queue
  - Gaming the Scheduler
  - Examples
- Chapter 9: Proportional Share Schedulers
  - Lottery scheduler
  - Ticket mechanisms
  - Stride scheduler
  - **Linux Completely Fair Scheduler**
- Chapter 26: Concurrency: An Introduction
  - Introduction
  - Race condition
  - Critical section

April 22, 2025	TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma	L7.52

52

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 !

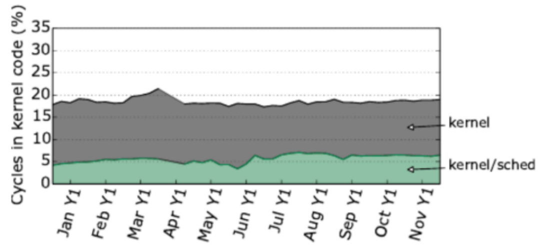


Figure 5: Kernel time, especially time spent in the scheduler, is a significant fraction of WSC cycles.

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

LINUX: COMPLETELY FAIR SCHEDULER (CFS)

- Loosely based on the stride scheduler
- CFS models system as a Perfect Multi-Tasking System
  - In a perfect system every process of the same priority (class) receives exactly $1/n^{th}$ of the CPU time
- Each scheduling class has a runqueue
  - Groups processes of the same class
  - In the 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)

COMPLETELY FAIR SCHEDULER - 2

- Every thread/process has a scheduling class (policy):
- **Normal classes**: SCHED_OTHER (TS), SCHED_IDLE, SCHED_BATCH
  - TS = Time Sharing (most user processes have this class)
- **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
  ```


55

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
```


56

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 exceeds 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_ns * number of processes in runqueue

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

57

CFS TRADEOFF

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

  CFS features reduced context switching → less overhead
  poor near-term fairness

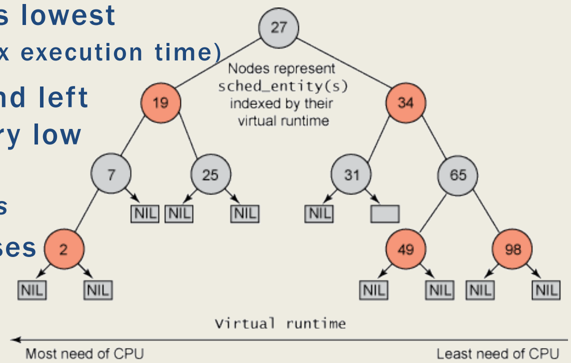
- **LOW** sched_min_granularity_ns (timeslice)
  sched_latency_ns
  sched_wakreup_granularity_ns

  CFS features increased context switching → more overhead
  better near-term fairness

58

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 are removed



59

CFS: JOB PRIORITY

- 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`

```
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,
};
```

- 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

60

COMPLETELY FAIR SCHEDULER - 6

- CFS tracks cumulative job run time with the **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
  - **GOAL:** Perfect scheduler → achieve equal **vruntime** for all processes of same priority
- Sleeping jobs: upon return a temporary **vruntime** can be used to increase temporarily the priority of the task
- When tasks wait for I/O they should receive a comparable share of the CPU as if they were performing compute ops when run again
- Key takeaway: ***identifying the next job to schedule is really fast!***

April 22, 2025	TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma	L7.61

61

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

April 22, 2025	TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma	L7.62

62

BEYOND CFS → EEVDF SCHEDULER

- Earliest Eligible Virtual Deadline First (EEVDF) Scheduler
  - **Linux kernel version 6.6**, October 29, 2023
  - First described in a research article in 1995
- Like CFS, EEVDF aims to distribute CPU time equally among all runnable tasks with the same priority.
- EEVDF assigns a virtual runtime to each task, creating a "lag" value that is used to determine whether a task has received its fair share of CPU time
  - A task with a positive lag is owed CPU time
  - A task with negative lag has exceeded its timeshare
- EEVDF calculates a virtual deadline (VD) for each task with lag greater or equal to zero
- Task with the earliest virtual deadline is selected to run next
- Virtual deadlines enable latency-sensitive tasks with shorter-time slices to be prioritized more than CFS which helps improve responsiveness
- More info: https://docs.kernel.org/scheduler/sched-eevdf.html

63

OBJECTIVES – 4/22

- Questions from 4/17
- Assignment 0 - Due Fri Apr 26
- C Tutorial - Pointers, Strings, Exec in C - Due Fri Apr 30
- Quiz 1 and Quiz 2
- Chapter 8: Multi-level Feedback Queue
  - Gaming the Scheduler
  - Examples
- Chapter 9: Proportional Share Schedulers
  - Lottery scheduler
  - Ticket mechanisms
  - Stride scheduler
  - Linux Completely Fair Scheduler
- Chapter 26: Concurrency: An Introduction
  - **Introduction**
  - Race condition
  - Critical section

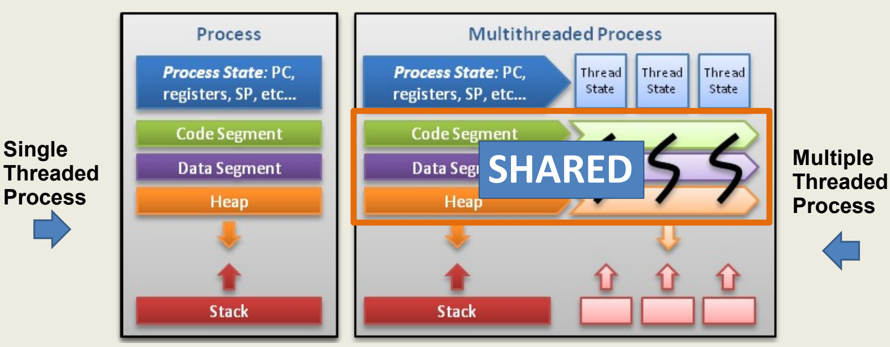
64

CHAPTER 26 - CONCURRENCY: AN INTRODUCTION

April 22, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - T oma | L7.65

65

THREADS



©Alfred Park, http://randu.org/tutorials/threads

April 22, 2025	TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma	L7.66

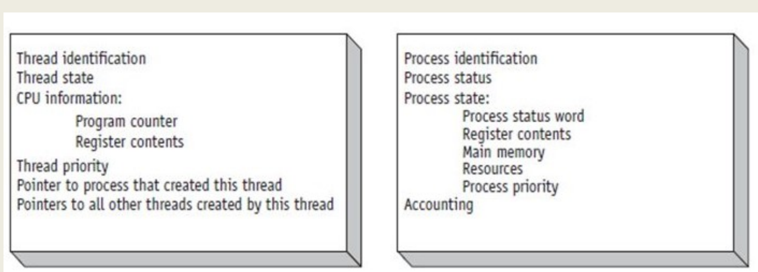
66

THREADS - 2

- Enables a single process (program) to have multiple "workers"
  - This is parallel programming...
- Supports independent path(s) of execution within a program *with shared memory* ...
- Each thread has its own Thread Control Block (TCB)
  - PC, registers, SP, and stack
- Threads share code segment, memory, and heap are shared
- ***What is an embarrassingly parallel program?***

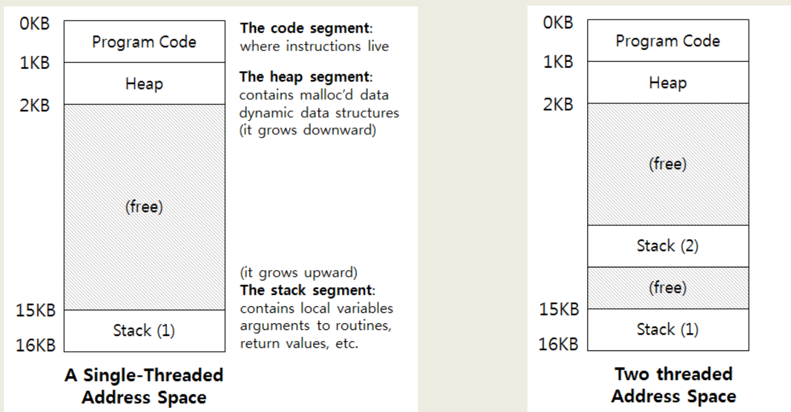
PROCESS AND THREAD METADATA

- Thread Control Block vs. Process Control Block



SHARED ADDRESS SPACE

- Every thread has it's own stack / PC



THREAD CREATION EXAMPLE

```
#include <stdio.h>
#include <assert.h>
#include <pthread.h>

void *mythread(void *arg) {
    printf("%s\n", (char *) arg);
    return NULL;
}

int
main(int argc, char *argv[]) {
    pthread_t p1, p2;
    int rc;
    printf("main: begin\n");
    rc = pthread_create(&p1, NULL, mythread, "A"); assert(rc == 0);
    rc = pthread_create(&p2, NULL, mythread, "B"); assert(rc == 0);
    // join waits for the threads to finish
    rc = pthread_join(p1, NULL); assert(rc == 0);
    rc = pthread_join(p2, NULL); assert(rc == 0);
    printf("main: end\n");
    return 0;
}
```

POSSIBLE ORDERINGS OF EVENTS

int main()	Thread 1	Thread 2
Starts running		
Prints 'main: begin'		
Creates Thread 1		
Creates Thread 2		
Waits for T1		
	Runs	
	Prints 'A'	
	Returns	
Waits for T2		
		Runs
		Prints 'B'
		Returns
Prints 'main: end'		

April 22, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L7.71

71

POSSIBLE ORDERINGS OF EVENTS - 2

int main()	Thread 1	Thread 2
Starts running		
Prints 'main: begin'		
Creates Thread 1		
	Runs	
	Prints 'A'	
	Returns	
Creates Thread 2		
		Runs
		Prints 'B'
		Returns
Waits for T1	*Returns immediately*	
Waits for T2		*Returns immediately*
Prints 'main: end'		

April 22, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L7.72

72

POSSIBLE ORDERINGS OF EVENTS - 3

int main()	Thread 1	Thread 2
Starts running		
Prints 'main: begin'		
Creates Thread 1		
Creates Thread 2		
Waits for T		
	Runs	
	Prints 'A'	
	Returns	
Waits for T2		*Immediately returns*
Prints 'main: end'		

What if execution order of events in the program matters?

73

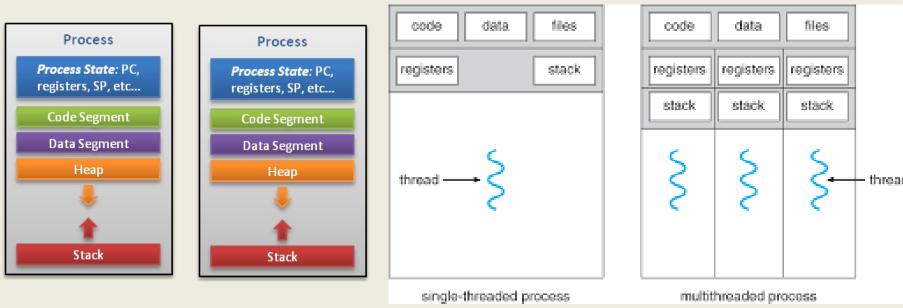
COUNTER EXAMPLE

- Counter example
- A + B : ordering
- Counter: incrementing global variable by two threads
- ***Is the counter example embarrassingly parallel?***
- ***What does the parallel counter program require?***

74

PROCESSES VS. THREADS

- What's the difference between forks and threads?
  - Forks: duplicate a process
  - Think of ***CLONING*** **-** There will be two identical processes at the end
  - Threads: no duplication of code/heap, lightweight execution threads



April 22, 2025	TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma	L7.75

75

OBJECTIVES – 4/22

- Questions from 4/17
- Assignment 0 - Due Fri Apr 26
- C Tutorial - Pointers, Strings, Exec in C - Due Fri Apr 30
- Quiz 1 and Quiz 2
- Chapter 8: Multi-level Feedback Queue
  - Gaming the Scheduler
  - Examples
- Chapter 9: Proportional Share Schedulers
  - Lottery scheduler
  - Ticket mechanisms
  - Stride scheduler
  - Linux Completely Fair Scheduler
- Chapter 26: Concurrency: An Introduction
  - Introduction
  - **Race condition**
  - Critical section

April 22, 2025	TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma	L7.76

76

RACE CONDITION

- What is happening with our counter?
  - When counter=50, consider code: counter = counter + 1
  - If synchronized, counter will = 52

OS	Thread1	Thread2	PC (after instruction)	%eax (after instruction)	counter (after instruction)
	before critical section		100	0	50
	mov 0x8049a1c, %eax		105	50	50
	add $0x1, %eax		108	51	50
interrupt					
save T1's state					
restore T2's state			100	0	50
		mov 0x8049a1c, %eax	105	50	50
		add $0x1, %eax	108	51	50
		mov %eax, 0x8049a1c	113	51	51
interrupt					
save T2's state					
restore T1's state			108	51	50
	mov %eax, 0x8049a1c		113	51	**51**

77

OBJECTIVES – 4/22

- Questions from 4/17
- Assignment 0 - Due Fri Apr 26
- C Tutorial - Pointers, Strings, Exec in C - Due Fri Apr 30
- Quiz 1 and Quiz 2
- Chapter 8: Multi-level Feedback Queue
  - Gaming the Scheduler
  - Examples
- Chapter 9: Proportional Share Schedulers
  - Lottery scheduler
  - Ticket mechanisms
  - Stride scheduler
  - Linux Completely Fair Scheduler
- Chapter 26: Concurrency: An Introduction
  - Introduction
  - Race condition
  - **Critical section**

78

CRITICAL SECTION

- Code that accesses a shared variable must not be ***concurrently*** executed by more than one thread
- Multiple *active* threads inside a ***critical section*** produce a ***race condition***.
- ***Atomic execution*** (*all code executed as a unit*) must be ensured in ***critical*** sections
  - These sections must be ***mutually exclusive***

April 22, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L7.79

79

LOCKS

- To demonstrate how critical section(s) can be executed "atomically-*as a unit*" Chapter 27 & beyond introduce locks

```
lock_t mutex;
. . .
lock(&mutex);
balance = balance + 1;
unlock(&mutex);
```

Critical section

- Counter example revisited

April 22, 2025 | TCSS422: Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L7.80

80

QUESTIONS

81