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

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma

1

EXTRA CREDIT SEMINAR - FRIDAY

Research Talk: Rethinking Reliability in Large-Scale Distributed Systems

Dr. Li (Lilly) Wu, Postdoctoral Research Associate in the College of Information and Computer Sciences at UMass Amherst.

Friday Jan 23, 12:20pm - 1:20pm MDS 313 (room may change)

Abstract: Today's distributed systems have evolved into a vast continuum, from hyperscale cloud data centers spanning continents, to 1000s of edge servers operating closer to users and devices. These systems power transformative workloads, such as AI-driven and microservices-based applications, reshaping domains including scientific discovery, autonomous transportation, and real-time digital experiences. As systems continue to grow in scale and complexity, they are becoming increasingly fragile. The impact of poor reliability is no longer confined to service outages; it increasingly affects everyday life, critical infrastructure, and safety-sensitive applications. Ensuring large systems operate correctly in the presence of failures requires rethinking reliability as a 1st-class design principle.

L7.2

2

EXTRA CREDIT SEMINAR - FRIDAY

Research Talk: Rethinking Reliability in Large-Scale Distributed Systems

Dr. Li (Lilly) Wu, Postdoctoral Research Associate in the College of Information and Computer Sciences at UMass Amherst.

Friday Jan 23, 12:20pm - 1:20pm MDS 313 (room may change)

Abstract cont'd: This talk highlights two pressing challenges to achieving this goal. First, many latency-critical applications are now deployed at the edge, where computing resources are limited and failures are more frequent, making traditional fault-tolerance mechanisms impractical. Second, modern distributed applications consist of hundreds of interacting services spanning thousands of machines; as a result, a single fault can quickly propagate, generating tens or even hundreds of anomalies—cascading failures—that make root-cause localization extremely difficult. To address these challenges, I will present two systems: FailLite and MicroRCA. FailLite is a resilient edge AI system that rethinks fault tolerance for AI workloads under resource constraints. It intelligently deploys smaller backup models and strategically places them to maximize service availability with negligible accuracy loss. MicroRCA focuses on root-cause diagnosis for cascading failures in cloud microservices. It uses a graph-based approach to model anomaly propagation and accurately identify root causes at runtime. Together, these systems support a broader vision of making AI systems reliable by design, enabling AI-driven applications to be deployed and operated reliably at scale. I will conclude the talk by outlining my future research directions.

L7.3

3

TEXT BOOK COUPON

- 15% off textbook code: **AAC72SAVE15**
- 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

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.4

4

TCSS 422 – OFFICE HRS – WINTER 2026

- **Office Hours plan for Winter:**
- **Tuesday 2:30 - 3:30 pm Instructor Wes, Zoom**
- **Tue/Thur 6:00 - 7:00 pm Instructor Wes, CP 229/Zoom**
- **Tue 6:00 – 7:00 pm GTA Robert, Zoom/MDS 302**
- **Wed 1:00 – 2:00 pm GTA Robert, Zoom/MDS 302**
- Instructor is available after class at 6pm in CP 229 each day

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.5

5

BONUS SESSION – CPU SCHEDULING PROBLEMS

- To help prepare for quiz 1 and the midterm
- Wednesday January 28, 6 pm
- CP 108 and live-streamed on Zoom
- Recording is posted
- Sample problems solved
- Sample problem solutions are posted online:
- https://faculty.washington.edu/wlloyd/courses/tcss422/scheduler_examples_w2026-solutions.pdf

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.6

6

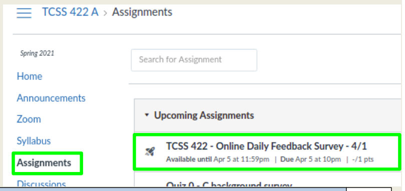
OBJECTIVES – 1/29

- Questions from 1/27
- Assignment 0 - Due Fri Jan 30 AOE
- C Tutorial - Pointers, Strings, Exec in C - Due Wed Feb 11 AOE
- Quiz 1 and Quiz 2
- Chapter 8: Multi-level Feedback Queue
  - 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

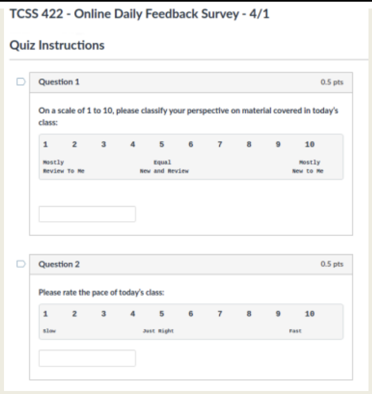
7

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 A › Assignments
Spring 2021
Search for Assignment
Home
Announcements
Zoom
Syllabus
Assignments
Discussions
Upcoming Assignments
TCSS 422 - Online Daily Feedback Survey - 4/1
Available until Apr 5 at 11:59pm | Due Apr 5 at 10pm | -/1 pts
Quiz 0 - C background survey

8


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

9

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (37 of 46 respondents – 80.4% - 7 online) :
- 1-mostly review, 5-equal new/review, 10-mostly new
- <u>**Average – 7.00 (↓ - previous 7.03)**</u>

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- <u>**Average – 5.59 (↑ - previous 5.08)**</u>

10

FEEDBACK FROM 1/27

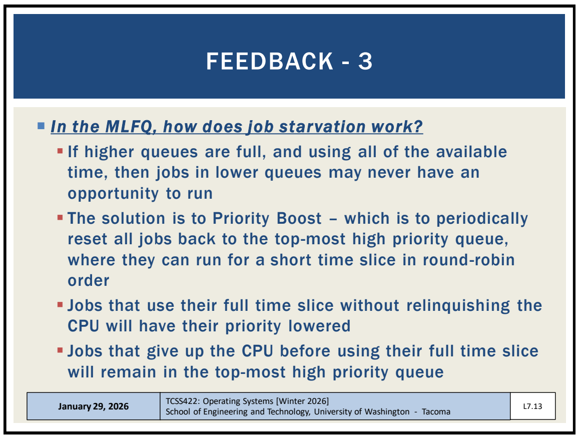
- <u>***How is a job's allotment time being tracked? Where in the CPU is it being tracked?***</u>
- The Linux kernel tracks '`vruntime`' for every process/thread in the system – this is in the `struct task_struct` process data structure described in chapter 4
- We talk about this more in chapter 9

11

FEEDBACK - 2

- <u>***For MLFQ, what is the difference between *time quantum* and *time slice*?***</u>
  - These terms are synonymous – they mean the same thing
- <u>***If an MLFQ has too many levels and frequently performs priority boosts (low S value), which should you change first?***</u>
  - If priority boosts are frequent, then jobs will never reach the lower queues. The lower queues can be removed as they will likely be under-utilized, if ever used at all.
- <u>***If there are multiple jobs spanning multiple quques, does each higher prorioty queue need to process its jobs before moving to jobs in a lower priority queue***</u>
  - YES

12

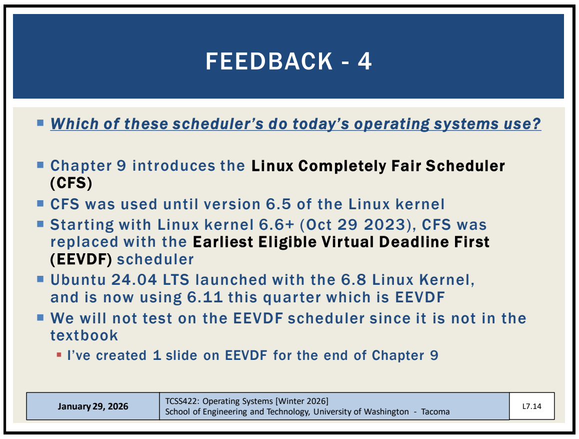


FEEDBACK - 3

- ***In the MLFQ, how does job starvation work?***
  - If higher queues are full, and using all of the available time, then jobs in lower queues may never have an opportunity to run
  - The solution is to Priority Boost – which is to periodically reset all jobs back to the top-most high priority queue, where they can run for a short time slice in round-robin order
  - Jobs that use their full time slice without relinquishing the CPU will have their priority lowered
  - Jobs that give up the CPU before using their full time slice will remain in the top-most high priority queue

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.13

13

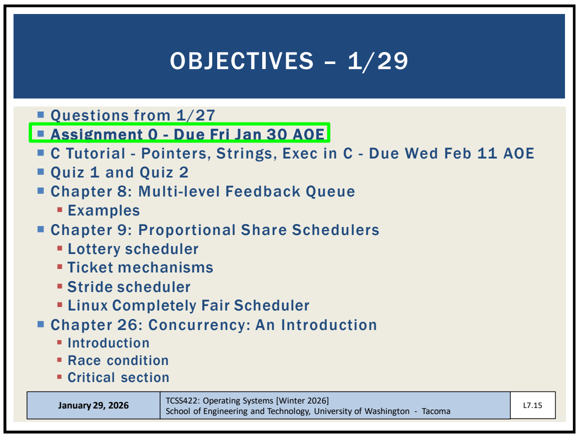


FEEDBACK - 4

- ***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've created 1 slide on EEVDF for the end of Chapter 9

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.14

14

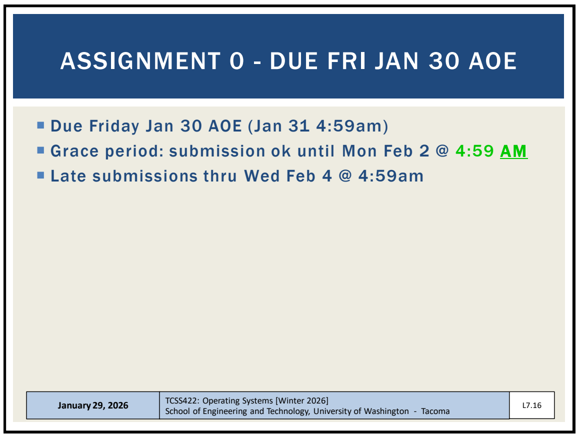


OBJECTIVES – 1/29

- Questions from 1/27
- **Assignment 0 - Due Fri Jan 30 AOE**
- C Tutorial - Pointers, Strings, Exec in C - Due Wed Feb 11 AOE
- Quiz 1 and Quiz 2
- Chapter 8: Multi-level Feedback Queue
  - 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

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.15

15

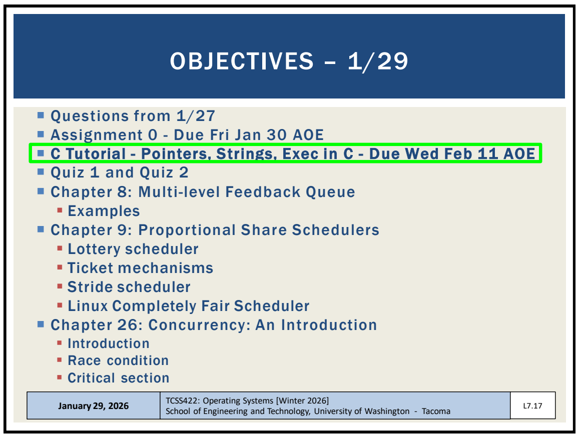


ASSIGNMENT 0 - DUE FRI JAN 30 AOE

- Due Friday Jan 30 AOE (Jan 31 4:59am)
- Grace period: submission ok until Mon Feb 2 @ 4:59 AM
- Late submissions thru Wed Feb 4 @ 4:59am

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.16

16

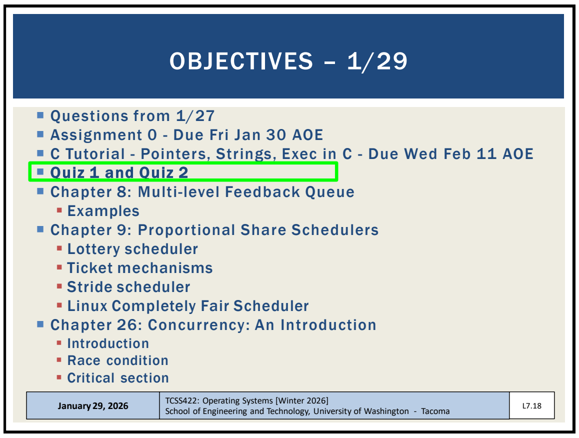


OBJECTIVES – 1/29

- Questions from 1/27
- Assignment 0 - Due Fri Jan 30 AOE
- **C Tutorial - Pointers, Strings, Exec in C - Due Wed Feb 11 AOE**
- Quiz 1 and Quiz 2
- Chapter 8: Multi-level Feedback Queue
  - 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

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.17

17

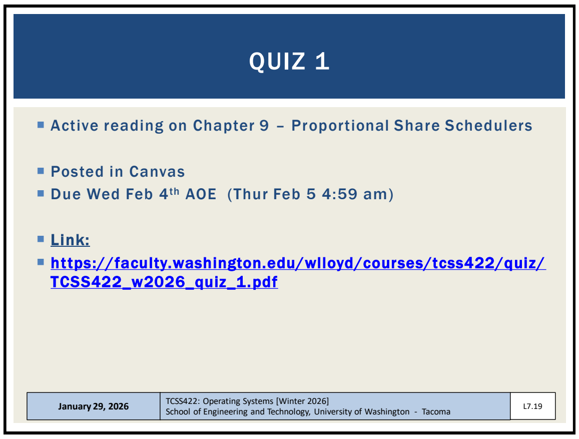


OBJECTIVES – 1/29

- Questions from 1/27
- Assignment 0 - Due Fri Jan 30 AOE
- C Tutorial - Pointers, Strings, Exec in C - Due Wed Feb 11 AOE
- **Quiz 1 and Quiz 2**
- Chapter 8: Multi-level Feedback Queue
  - 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

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.18

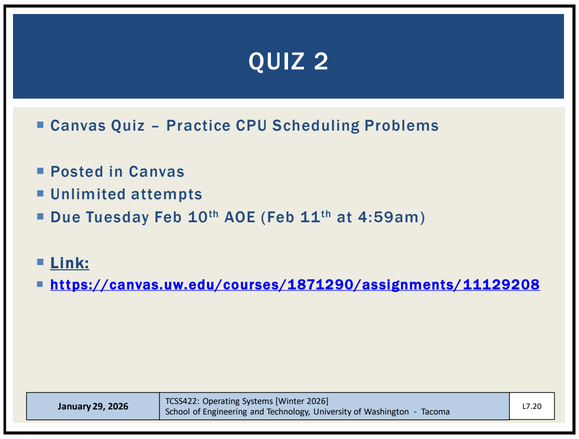
18



QUIZ 1

- Active reading on Chapter 9 – Proportional Share Schedulers
- Posted in Canvas
- Due Wed Feb 4th AOE (Thur Feb 5 4:59 am)
- **Link:**
- **https://faculty.washington.edu/wlloyd/courses/tcss422/quiz/TCSS422_w2026_quiz_1.pdf**

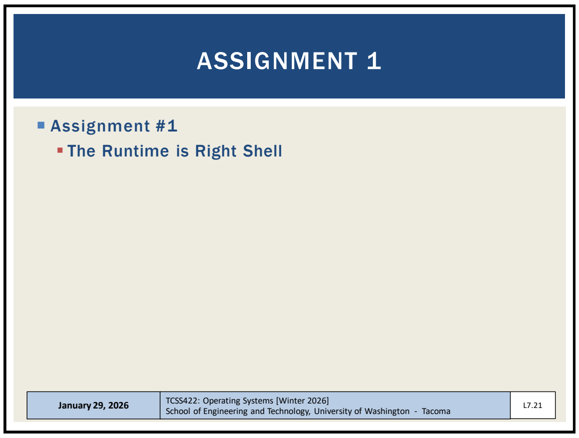
19



QUIZ 2

- Canvas Quiz – Practice CPU Scheduling Problems
- Posted in Canvas
- Unlimited attempts
- Due Tuesday Feb 10th AOE (Feb 11th at 4:59am)
- **Link:**
- **https://canvas.uw.edu/courses/1871290/assignments/11129208**

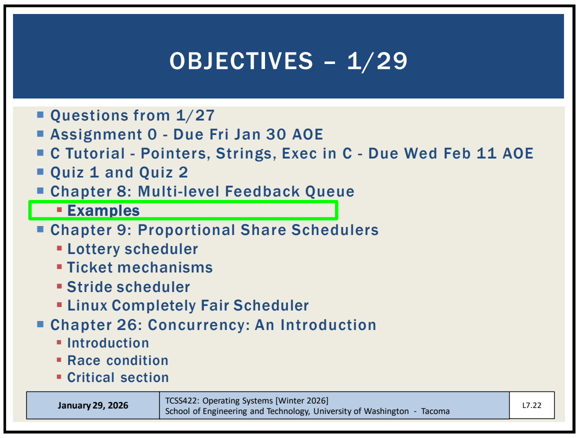
20



ASSIGNMENT 1

- Assignment #1
  - The Runtime is Right Shell

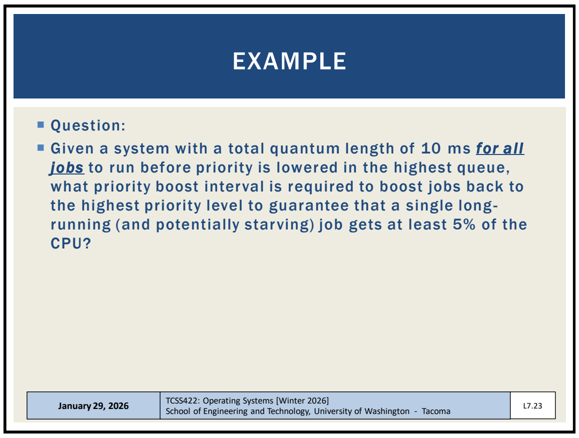
21



OBJECTIVES – 1/29

- Questions from 1/27
- Assignment 0 - Due Fri Jan 30 AOE
- C Tutorial - Pointers, Strings, Exec in C - Due Wed Feb 11 AOE
- Quiz 1 and Quiz 2
- Chapter 8: Multi-level Feedback Queue
  - 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

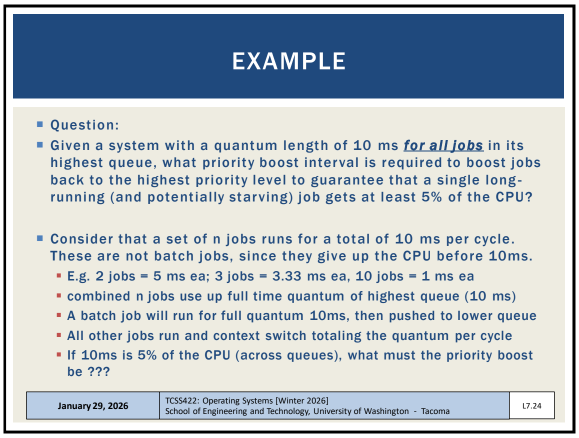
22



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, what priority boost interval is required 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?

23



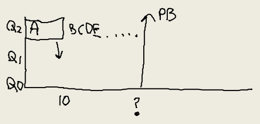
EXAMPLE

- Question:
- Given a system with a quantum length of 10 ms ***for all jobs*** in its highest queue, what priority boost interval is required 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?
- Consider that a set of n jobs runs for a total of 10 ms per cycle. These are not batch jobs, since they give up the CPU before 10ms.
  - E.g. 2 jobs = 5 ms ea; 3 jobs = 3.33 ms ea, 10 jobs = 1 ms ea
  - combined n jobs use up full time quantum of highest queue (10 ms)
  - A batch job will run for full quantum 10ms, then pushed to lower queue
  - All other jobs run and context switch totaling the quantum per cycle
  - If 10ms is 5% of the CPU (across queues), what must the priority boost be ???

24

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, what priority boost interval is required 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?



$.05\ PB = 10$

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

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.25

25

OBJECTIVES – 1/29

- Questions from 1/27
- Assignment 0 - Due Fri Jan 30 AOE
- C Tutorial - Pointers, Strings, Exec in C - Due Wed Feb 11 AOE
- Quiz 1 and Quiz 2
- Chapter 8: Multi-level Feedback Queue
  - 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

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.26

26

CHAPTER 9 - PROPORTIONAL SHARE SCHEDULER

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.27

27

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

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.28

28

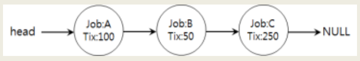
LOTTERY SCHEDULER

- 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

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.29

29

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...
```

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.30

30

OBJECTIVES – 1/29

- Questions from 1/27
- Assignment 0 - Due Fri Jan 30 AOE
- C Tutorial - Pointers, Strings, Exec in C - Due Wed Feb 11 AOE
- Quiz 1 and Quiz 2
- Chapter 8: Multi-level Feedback Queue
  - 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

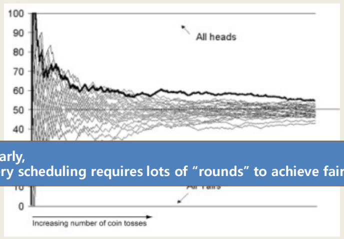
- Scheduler picks a **winning** ticket
  - Load the job with the winning ticket and run it
- Example:
  - Given 100 tickets in the pool
    - Job A has 75 tickets: 0 - 74
    - Job B has 25 tickets: 75 – 99

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

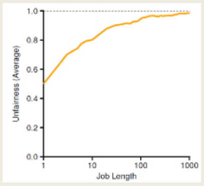
- Equality of distribution (fairness) requires a lot of flips!



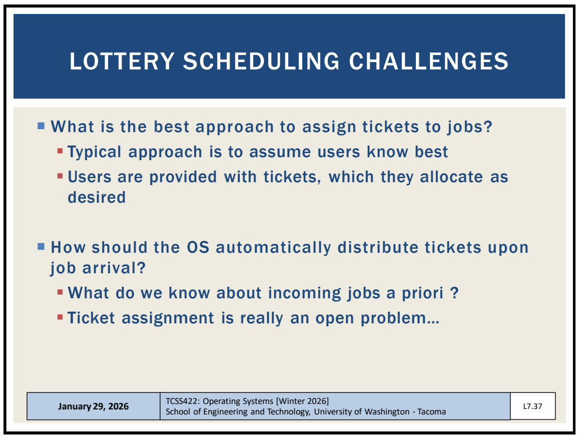
Similarly,
Lottery scheduling requires lots of "rounds" to achieve fairness.

LOTTERY FAIRNESS

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



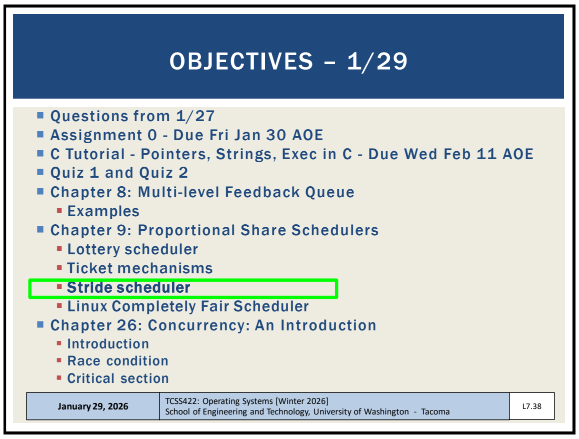
When the job length is not very long,
average unfairness can be quite severe.



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…

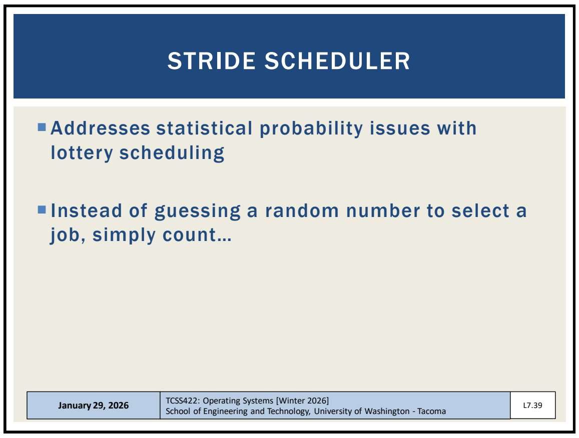
37



OBJECTIVES – 1/29

- Questions from 1/27
- Assignment 0 - Due Fri Jan 30 AOE
- C Tutorial - Pointers, Strings, Exec in C - Due Wed Feb 11 AOE
- Quiz 1 and Quiz 2
- Chapter 8: Multi-level Feedback Queue
  - 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

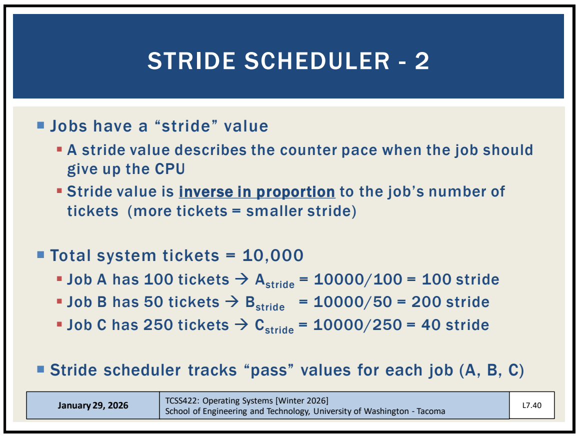
38



STRIDE SCHEDULER

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

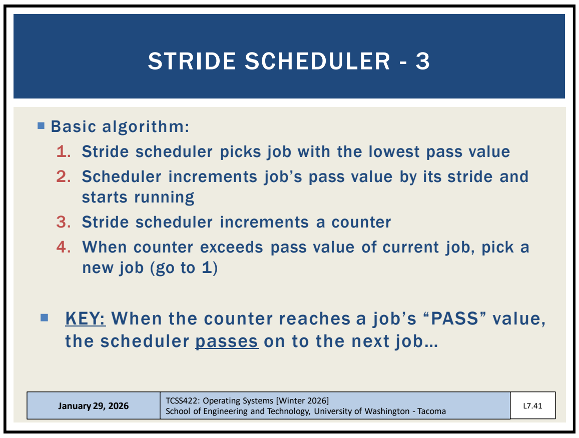
39



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)

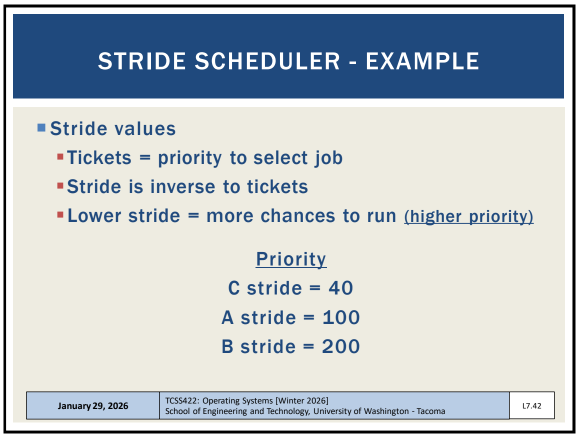
40



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…

41

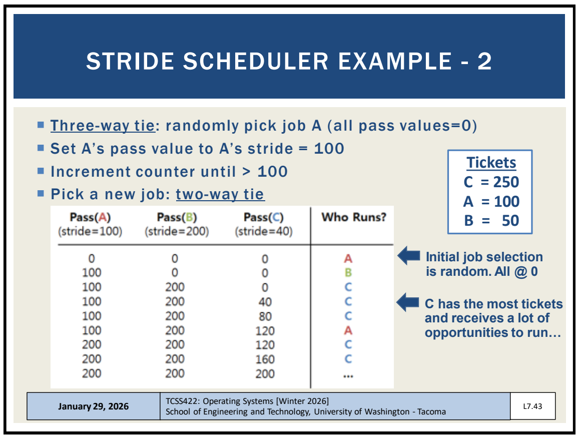


STRIDE SCHEDULER - EXAMPLE

- 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

42



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	…

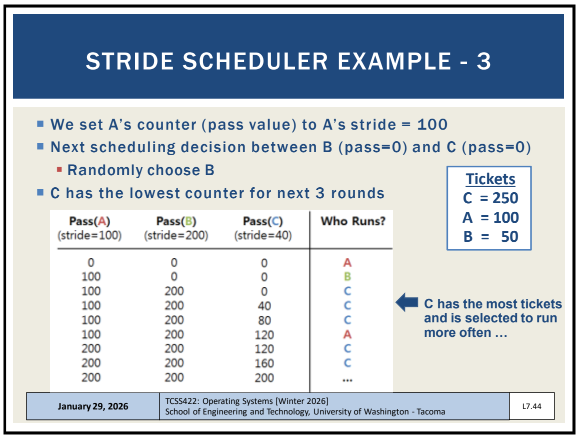
Tickets
C = 250
A = 100
B = 50

Initial job selection is random. All @ 0

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

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.43

43



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

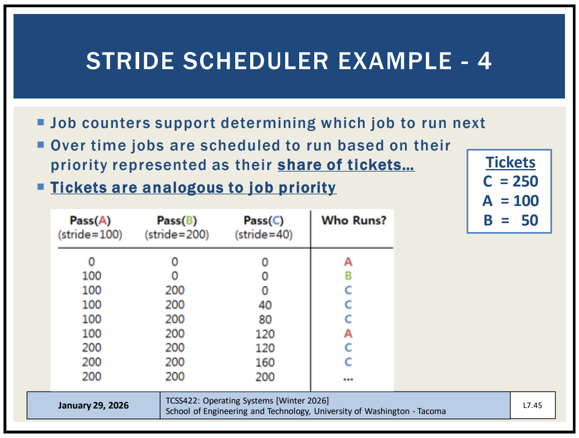
Pass(A) (stride=100)	Pass(B) (stride=200)	Pass(C) (stride=40)	Who Runs?
0	0	0	A
100	0	0	B
100	200	0	C
100	200	40	C
100	200	80	C
100	200	120	A
200	200	120	C
200	200	160	C
200	200	200	…

Tickets
C = 250
A = 100
B = 50

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

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.44

44



STRIDE SCHEDULER EXAMPLE - 4

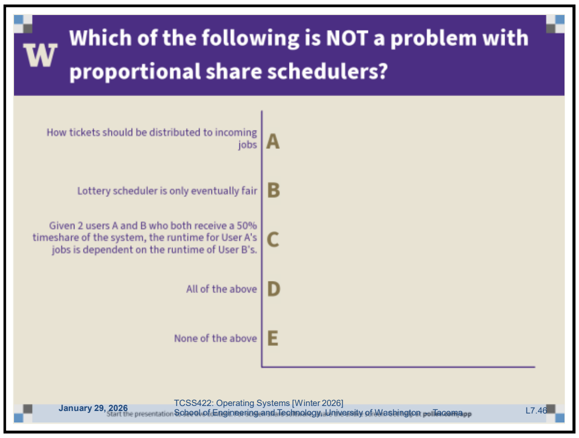
- Job counters support determining which job to run next
- Over time jobs are scheduled to run based on their priority represented as their **share of tickets…**
- **Tickets are analogous to job priority**

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

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.45

45



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

How tickets should be distributed to incoming jobs **A**

Lottery scheduler is only eventually fair **B**

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. **C**

All of the above **D**

None of the above **E**

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.46

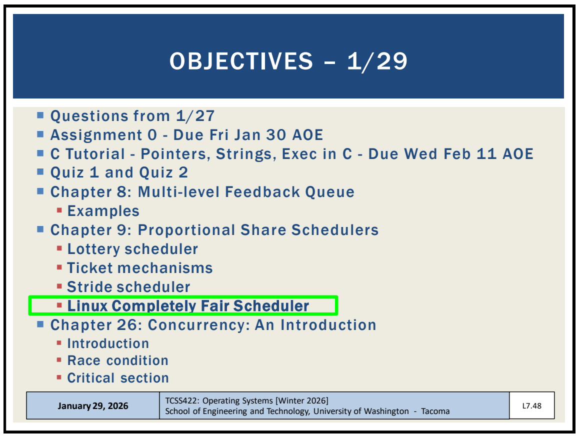
Start the presentation to see live content. For screen share software, share the entire screen. Get help at pollev.com/app

46

WE WILL RETURN AT 5:02PM

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.47

47

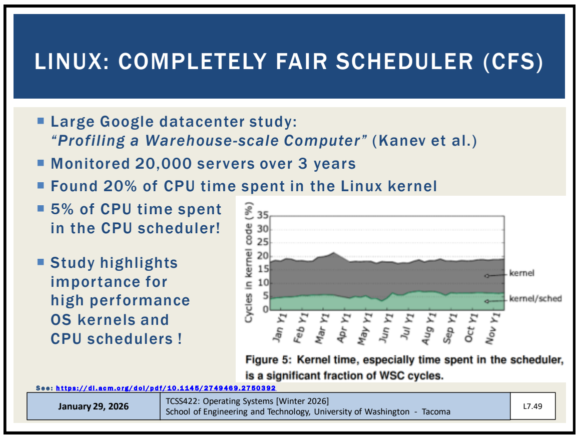


OBJECTIVES – 1/29

- Questions from 1/27
- Assignment 0 - Due Fri Jan 30 AOE
- C Tutorial - Pointers, Strings, Exec in C - Due Wed Feb 11 AOE
- Quiz 1 and Quiz 2
- Chapter 8: Multi-level Feedback Queue
  - 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

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.48

48



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 !

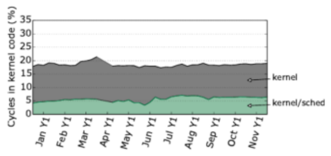
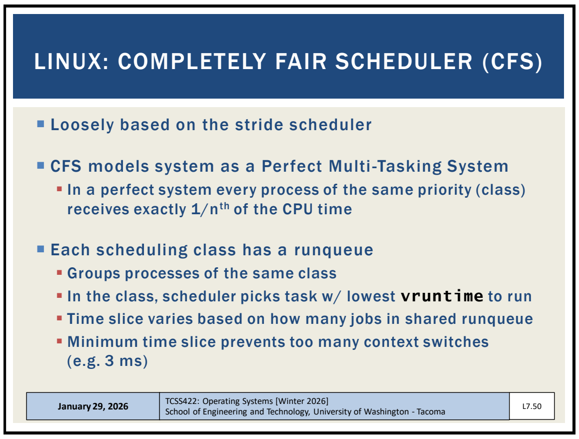


Figure 5: Kernel time, especially time spent in the scheduler, is a significant fraction of WSC cycles.

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

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.49

49

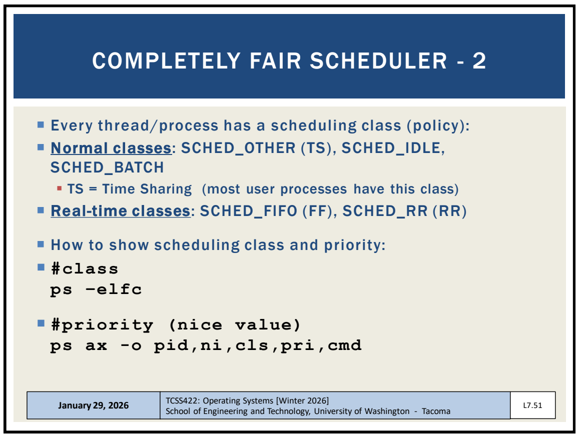


LINUX: COMPLETELY FAIR SCHEDULER (CFS)

- 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)

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.50

50



COMPLETELY FAIR SCHEDULER - 2

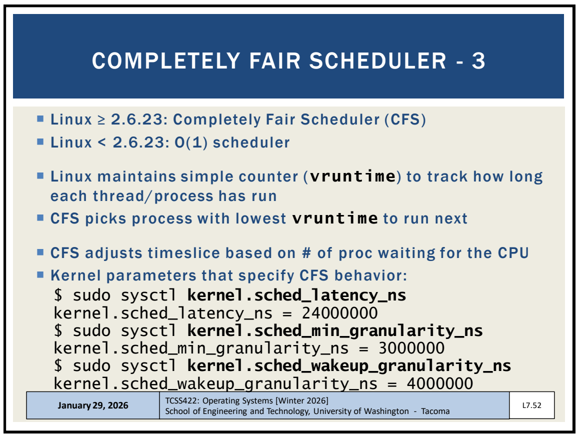
- Every thread/process has a scheduling class (policy):
- Normal classes: SCHED_OTHER (TS), SCHED_IDLE, SCHED_BATCH
  - TS = Time Sharing (most user processes have this class)
- Real-time classes: SCHED_FIFO (FF), SCHED_RR (RR)
- How to show scheduling class and priority:

-
  ```
  #class
  ps -elfc
  ```

-
  ```
  #priority (nice value)
  ps ax -o pid,ni,cls,pri,cmd
  ```

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.51

51



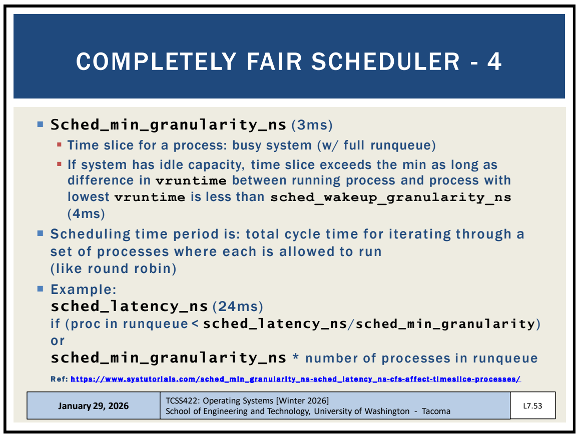
COMPLETELY FAIR SCHEDULER - 3

- 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
```

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.52

52



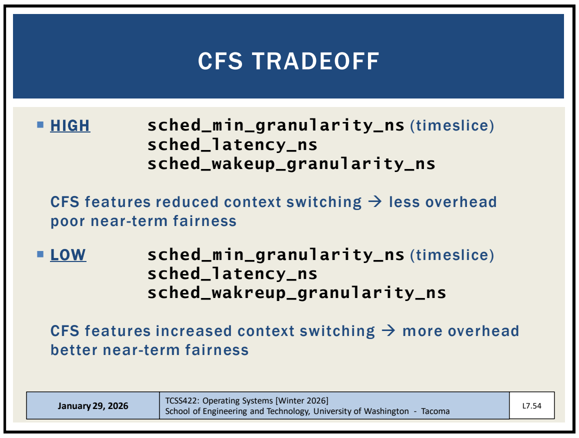
COMPLETELY FAIR SCHEDULER - 4

- **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/

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.53

53



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

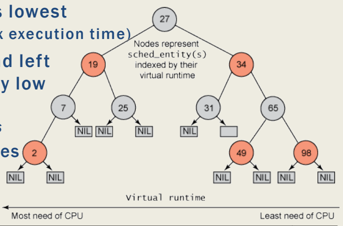
  CFS features increased context switching → more overhead
  better near-term fairness

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.54

54

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



January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.55

55

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

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.56

56

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!

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.57

57

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

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.58

58

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

January 29, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L7.59

59

QUESTIONS

77