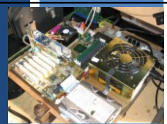
TCSS 422: OPERATING SYSTEMS



Proportional Share Schedulers, Linux Completely Fair Scheduler, Introduction to Concurrency

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

February 3, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma

1

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

February 3, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L8.2

2

TCSS 422 – OFFICE HRS – SPRING 2025

- **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

February 3, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L8.3

3

OBJECTIVES – 2/3

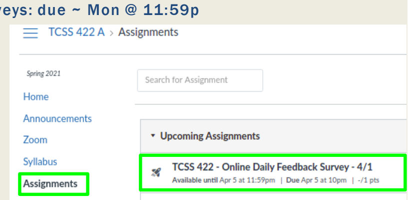
- **Questions from 1/29**
- C Tutorial - Pointers, Strings, Exec in C - Due Wed Feb 11 AOE
- Assignment 0 - Closes Tue Feb 3 AOE | Assignment 1 posted
- Quiz 1 (Due Wed Feb 4 AOE) – Quiz 2 (Due Tue Feb 10 AOE)
- Chapter 9: Proportional Share Schedulers
  - Linux Completely Fair Scheduler
- Chapter 26: Concurrency: An Introduction
  - Race condition
  - Critical section
- Chapter 27: Linux Thread API
  - pthread_create/_join
  - pthread_mutex_lock/_unlock/_trylock/_timelock
  - pthread_cond_wait/_signal/_broadcast

February 3, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L8.4

4

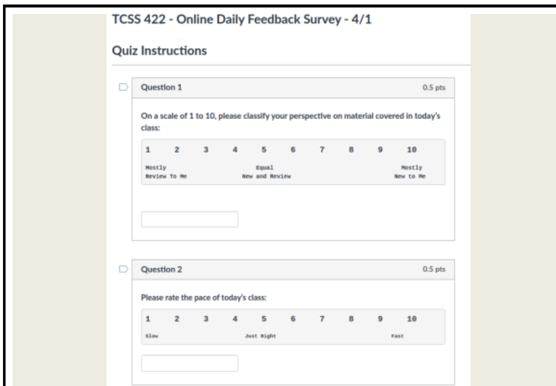
ONLINE DAILY FEEDBACK SURVEY

- 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



February 3, 2026 | TCSS422: Computer Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L8.5

5



February 3, 2026 | TCSS422: Computer Operating Systems [Spring 2025] School of Engineering and Technology, University of Washington - Tacoma | L8.6

6

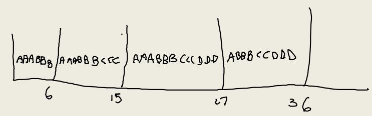
MATERIAL / PACE

- Please classify your perspective on material covered in today's class (41 of 46 respondents – 89.1%) :
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 5.75 (↓ - previous 7.00)**

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.03 (↓ - previous 5.59)**

7

Q2



9

$$\frac{n \cdot \sum (.8)^2 + (.2)^2}{n \cdot \sum (.64 + .04)}$$

$$2(.68)$$

$$1.36 \rightarrow \frac{1}{1.36} \rightarrow .735$$

10

FEEDBACK FROM 1/29

- ***Do argv I/O errors exist?***

- In: `int main(int argc, char * argv[])`
- `argv[]` is an array of pointers to C strings
- Reading elements of the `argv[]` array, reads memory
- No I/O is performed
- No reads, writes, syscalls
- There should be no I/O errors

11

FEEDBACK FROM 4/22

- ***What would happen in an MLFQ scheduler if there are so many jobs overall that the high priority queue never finishes giving each job a time slice to execute before doing a priority boost?***

- cycle time – total time shared among all jobs in a run queue
- time slice – time an individual job runs for

- From slide 6.50:
  - no rule explicitly describes how the cycle time is divided among jobs
  - No rule explicitly describes how time slice is determine
- Any MLFQ problem having this issue would require rules to describe how this scenario is handled to allow a scheduling graph to be drawn

12

REVIEW: 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.

13

ADDRESSING AN OVERLOADED RUNQUEUE

- One possible way:
  - Cycle time split evenly among jobs in runqueue with no min timeslice
  - For MLFQ, all jobs in runqueue use full timeslice and have priority reduced
  - Not realistic in practice - timeslice becomes too small to be useful
- Another way:
  - Specify min_time_slice (1 ms) per job, and total_cycle_time (10 ms)
  - Job's time_slice = total_cycle_time / jobs_in_runqueue
  - Beyond 10 jobs, other jobs receive no runtime this cycle
  - Jobs receiving no runtime are scheduled first in next cycle
  - Jobs could pile up and experience multi-cycle delays
  - More realistic

February 3, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L8.14

14

OBJECTIVES – 2/3

- Questions from 1/29
- **C Tutorial - Pointers, Strings, Exec in C - Due Wed Feb 11 AOE**
- Assignment 0 - Closes Tue Feb 3 AOE | Assignment 1 posted
- Quiz 1 (Due Wed Feb 4 AOE) – Quiz 2 (Due Tue Feb 10 AOE)
- Chapter 9: Proportional Share Schedulers
  - Linux Completely Fair Scheduler
- Chapter 26: Concurrency: An Introduction
  - Race condition
  - Critical section
- Chapter 27: Linux Thread API
  - pthread_create/_join
  - pthread_mutex_lock/_unlock/_trylock/_timelock
  - pthread_cond_wait/_signal/_broadcast

February 3, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L8.15

15

OBJECTIVES – 2/3

- Questions from 1/29
- C Tutorial - Pointers, Strings, Exec in C - Due Wed Feb 11 AOE
- **Assignment 0 - Closes Tue Feb 3 AOE | Assignment 1 posted**
- Quiz 1 (Due Wed Feb 4 AOE) – Quiz 2 (Due Tue Feb 10 AOE)
- Chapter 9: Proportional Share Schedulers
  - Linux Completely Fair Scheduler
- Chapter 26: Concurrency: An Introduction
  - Race condition
  - Critical section
- Chapter 27: Linux Thread API
  - pthread_create/_join
  - pthread_mutex_lock/_unlock/_trylock/_timelock
  - pthread_cond_wait/_signal/_broadcast

February 3, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L8.16

16

ASSIGNMENT 0 - CLOSES TUE FEB 03 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

February 3, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L8.17

17

OBJECTIVES – 2/3

- Questions from 1/29
- C Tutorial - Pointers, Strings, Exec in C - Due Wed Feb 11 AOE
- **Assignment 0 - Closes Tue Feb 3 AOE | Assignment 1 posted**
- Quiz 1 (Due Wed Feb 4 AOE) – Quiz 2 (Due Tue Feb 10 AOE)
- Chapter 9: Proportional Share Schedulers
  - Linux Completely Fair Scheduler
- Chapter 26: Concurrency: An Introduction
  - Race condition
  - Critical section
- Chapter 27: Linux Thread API
  - pthread_create/_join
  - pthread_mutex_lock/_unlock/_trylock/_timelock
  - pthread_cond_wait/_signal/_broadcast

February 3, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L8.18

18

OBJECTIVES – 2/3

- Questions from 1/29
- C Tutorial - Pointers, Strings, Exec in C - Due Wed Feb 11 AOE
- Assignment 0 - Closes Tue Feb 3 AOE | Assignment 1 posted
- **Quiz 1 (Due Wed Feb 4 AOE) – Quiz 2 (Due Tue Feb 10 AOE)**
- Chapter 9: Proportional Share Schedulers
  - Linux Completely Fair Scheduler
- Chapter 26: Concurrency: An Introduction
  - Race condition
  - Critical section
- Chapter 27: Linux Thread API
  - pthread_create/_join
  - pthread_mutex_lock/_unlock/_trylock/_timelock
  - pthread_cond_wait/_signal/_broadcast

February 3, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L8.19

19

QUIZ 1

- Active reading on Chapter 9 – Proportional Share Schedulers
- Posted in Canvas
- Due Wednesday Feb 4th AOE
- Link:
- https://faculty.washington.edu/wlloyd/courses/tcss422/quiz/TCSS422_w2026_quiz_1.pdf

February 3, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L8.20

20

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

February 3, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L8.21

21

OBJECTIVES – 2/3

- Questions from 1/29
- C Tutorial - Pointers, Strings, Exec in C - Due Wed Feb 11 AOE
- Assignment 0 - Closes Tue Feb 3 AOE | Assignment 1 posted
- Quiz 1 (Due Wed Feb 4 AOE) – Quiz 2 (Due Tue Feb 10 AOE)
- Chapter 9: Proportional Share Schedulers
  - Linux Completely Fair Scheduler
- Chapter 26: Concurrency: An Introduction
  - Race condition
  - Critical section
- Chapter 27: Linux Thread API
  - pthread_create/_join
  - pthread_mutex_lock/_unlock/_trylock/_timelock
  - pthread_cond_wait/_signal/_broadcast

February 3, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L8.22

22

CHAPTER 9 - PROPORTIONAL SHARE SCHEDULER

February 3, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L8.23

23

OBJECTIVES – 2/3

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

February 3, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L8.24

24

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

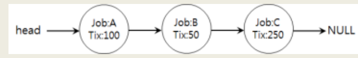
February 3, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L8.25

25

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




```c
// 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 – 2/3

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

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

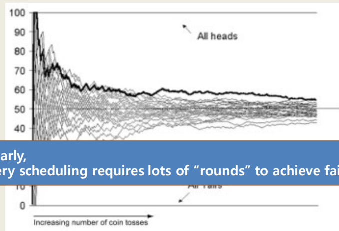
- Scheduler picks a **winning** ticket
  - Load the job with the winning ticket and run it
- Example:
  - Given 100 tickets in the pool
    - Job A has 75 tickets: 0 - 74
    - Job B has 25 tickets: 75 – 99

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!



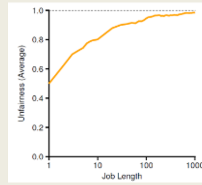
Similarly,
Lottery scheduling requires lots of “rounds” to achieve fairness.

February 3, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L8.32

32

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**.

February 3, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L8.33

33

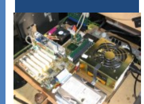
LOTTERY SCHEDULING CHALLENGES

- 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…

February 3, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L8.34

34

WE WILL RETURN AT 5:00PM



February 3, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L8.35

35

OBJECTIVES – 2/3

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

February 3, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L8.36

36

STRIDE SCHEDULER

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

February 3, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L8.37

37

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)

February 3, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L8.38

38

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...

February 3, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L8.39

39

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

February 3, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L8.40

40

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...

February 3, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L8.41

41

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 ...

February 3, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L8.42

42

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	...

February 3, 2026 | TCSS422: Operating Systems [Winter 2026] School of Engineering and Technology, University of Washington - Tacoma | L8.43

43

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**

44

OBJECTIVES – 2/3

- Questions from 1/29
- C Tutorial - Pointers, Strings, Exec in C - Due Wed Feb 11 AOE
- Assignment 0 - Closes Tue Feb 3 AOE | Assignment 1 posted
- Quiz 1 (Due Wed Feb 4 AOE) – Quiz 2 (Due Tue Feb 10 AOE)
- Chapter 9: Proportional Share Schedulers
  - Linux Completely Fair Scheduler
- Chapter 26: Concurrency: An Introduction
  - Race condition
  - Critical section
- Chapter 27: Linux Thread API
  - pthread_create/_join
  - pthread_mutex_lock/_unlock/_trylock/_timelock
  - pthread_cond_wait/_signal/_broadcast

45

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 !

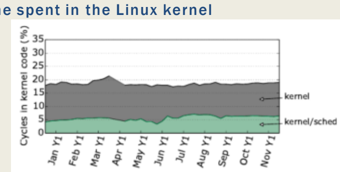


Figure 5: Kernel time, especially time spent in the scheduler, is a significant fraction of WSC cycles.

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

46

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 should receive exactly 1/nth of the CPU time
- Linux scheduling classes group jobs by priority
  - CFS only schedules user processes, those with SCHED_OTHER (SCHED_NORMAL) class (*class is also called scheduling "policy"*)
  - CFS 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)

47

COMPLETELY FAIR SCHEDULER - 2

- Every thread/process has a scheduling class (policy):
- CFS classes: SCHED_OTHER (TS), also 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
```


48

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 user 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
```


49

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/

50

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

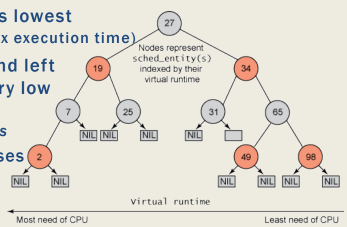
- **LOW** **sched_min_granularity_ns** (timeslice)
  sched_latency_ns
  sched_wakreup_granularity_ns

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

51

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



52

LINUX: JOB NICE VALUES

```
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,
};
```

- 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

53

COMPLETELY FAIR SCHEDULER - 6

- CFS tracks cumulative job run time with the **vruntime** variable
- The task 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
  - **CFS GOAL:** Be a perfect scheduler → achieve equal **vruntime** for all processes
- 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!

54

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

55

BEYOND CFS → EEVDF SCHEDULER

- **E**arliest **E**ligible **V**irtual **D**eadline **F**irst (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 calculates a virtual deadline (VD) for each task, by considering each task's "lag" value – the difference between the time a task should have received vs. what it actually received.
  - Only tasks with positive lag can run. They are owed CPU time
  - A task with negative lag has exceeded its timeshare – it does not run
- 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

56

QUESTIONS

97