TCSS 422: OPERATING SYSTEMS

Multi-level Feedback Queue (MLFQ) Scheduler – Proportional Share Schedulers

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

1

OFFICE HOURS – FALL 2021

- **Tuesdays:**
  - 4:00 to 4:30 pm - CP 229
  - 7:15 to 7:45+ pm – ONLINE via Zoom
- **Thursdays**
  - 4:15 to 4:45 pm – ONLINE via Zoom
  - 7:15 to 7:45+ pm – ONLINE via Zoom
- Or email for appointment
- Zoom link sent via Canvas Announcements

> Office Hours set based on Student Demographics survey feedback

2

TEXT BOOK COUPON

- 15% off textbook code: **SPOOKY15** (*through Friday Oct 22*)
- https://www.lulu.com/shop/remzi-arpaci-dusseau-and-andrea-arpaci-dusseau/operating-systems-three-easy-pieces-softcover-version-100/paperback/product-23779877.html?page=1&pageSize=4

3

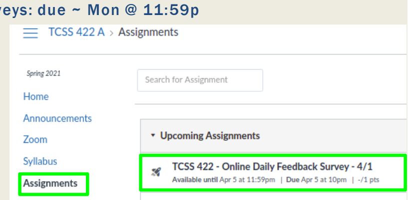
OBJECTIVES – 10/19

- Questions from 10/14
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Quiz 1 – Active Reading Chapter 9
- Chapter 8: Multi-level Feedback Queue
  - MLFQ Scheduler
  - Job Starvation
  - Gaming the Scheduler
  - Examples
- Chapter 9: Proportional Share Schedulers

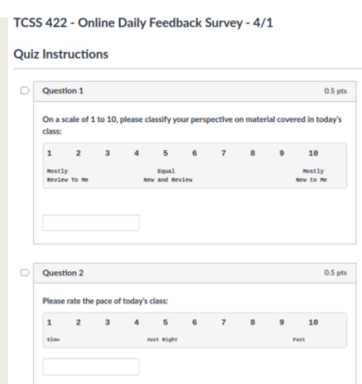
4

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



5



6

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (26 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- Average – 6.62 (↓ - previous 6.73)

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- Average – 5.54 (↓ - previous 5.59)

October 19, 2021 | TCSS422: Computer Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma | L6.7

7

FEEDBACK

- ***What happens when you manually decide for a process to have a higher priority than another? How does this effect the scheduler?***
  - In Linux, users cannot directly assign processes priority values
  - Linux offers the `nice` command which allows users to suggest a process priority to the kernel
  - By default, only superuser can increase the priority of a process. All other users can only decrease priority
  - User assignable nice values range from **-20** (most favorable to the process) to **19** (least favorable to the process), default is **0**

October 19, 2021 | TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma | L6.8

8

FEEDBACK - 2

- (cont'd) ***What happens when you manually decide for a process to have a higher priority than another? How does this effect the scheduler?***
  - If 2 identical CPU-bound processes run simultaneously on a single-CPU Linux system, each processes share of the CPU time will be proportional to (20 – p), where p is the process priority.
  - A process run with nice +15, will receive 25% of the original CPU time for a normal-priority process: (20 – 15)/(20 – 0) = 0.25 → 25%
  - ***For 2 identical processes, what is the lowest % timeshare possible when adjusting process priority with nice?***
  - (20 – 19) / (20 – 0)
  - (20 – 19) / (20 – 0) = 1 / 20 = .05 → 5%

October 19, 2021 | TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma | L6.9

9

FEEDBACK - 3

- (cont'd) ***What happens when you manually decide for a process to have a higher priority than another? How does this effect the scheduler?***
- Process priority, and the nice command are explained further when we discuss the Linux Completely Fair Scheduler at the end of Chapter 9

October 19, 2021 | TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma | L6.10

10

OBJECTIVES – 10/19

- Questions from 10/14
- **Assignment 0**
- C Tutorial - Pointers, Strings, Exec in C
- Quiz 1 – Active Reading Chapter 9

- Chapter 8: Multi-level Feedback Queue
  - MLFQ Scheduler
  - Job Starvation
  - Gaming the Scheduler
  - Examples
- Chapter 9: Proportional Share Schedulers

October 19, 2021 | TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma | L6.11

11

OBJECTIVES – 10/19

- Questions from 10/14
- Assignment 0
- **C Tutorial - Pointers, Strings, Exec in C**
- Quiz 1 – Active Reading Chapter 9

- Chapter 8: Multi-level Feedback Queue
  - MLFQ Scheduler
  - Job Starvation
  - Gaming the Scheduler
  - Examples
- Chapter 9: Proportional Share Schedulers

October 19, 2021 | TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma | L6.12

12

OBJECTIVES – 10/19

- Questions from 10/14
- Assignment 0
- **C Tutorial - Pointers, Strings, Exec in C**
- Quiz 1 – Active Reading Chapter 9
- Chapter 8: Multi-level Feedback Queue
  - MLFQ Scheduler
  - Job Starvation
  - Gaming the Scheduler
  - Examples
- Chapter 9: Proportional Share Schedulers

13

OBJECTIVES – 10/19

- Questions from 10/14
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Quiz 1 – Active Reading Chapter 9
- Chapter 8: Multi-level Feedback Queue
  - MLFQ Scheduler
  - Job Starvation
  - Gaming the Scheduler
  - Examples
- Chapter 9: Proportional Share Schedulers

14

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

15

MULTI-LEVEL FEEDBACK QUEUE

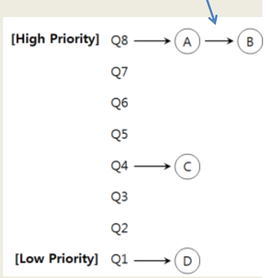
- Objectives:
  - Improve turnaround time:
    Run shorter jobs first
  - Minimize response time:
    Important for interactive jobs (UI)
- Achieve without a priori knowledge of job length

16

MLFQ - 2

Round-Robin within a Queue

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



17

MLFQ: DETERMINING JOB PRIORITY

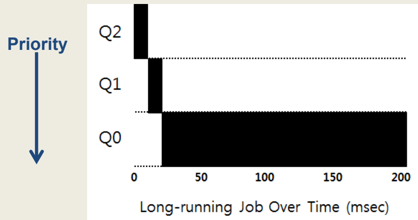
- New arriving jobs are placed into highest priority queue
- If a job uses its entire time slice, priority is reduced (↓)
  - Jobs appears CPU-bound ("batch" job), not interactive (GUI/UI)
- If a job relinquishes the CPU for I/O priority stays the same

MLFQ approximates SJF

18

MLFQ: LONG RUNNING JOB

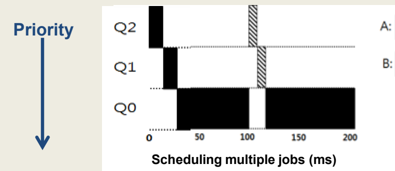
- Three-queue scheduler, time slice=10ms



19

MLFQ: BATCH AND INTERACTIVE JOBS

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

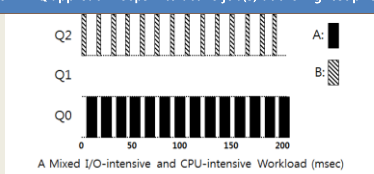


20

MLFQ: BATCH AND INTERACTIVE - 2

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

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



21

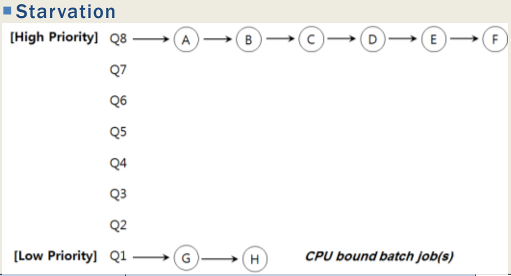
OBJECTIVES – 10/19

- Questions from 10/14
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Quiz 1 – Active Reading Chapter 9
- Chapter 8: Multi-level Feedback Queue
  - MLFQ Scheduler
  - **Job Starvation**
  - Gaming the Scheduler
  - Examples
- Chapter 9: Proportional Share Schedulers

22

MLFQ: ISSUES

- Starvation



23

OBJECTIVES – 10/19

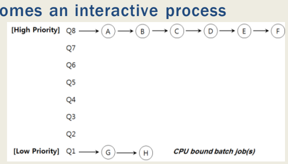
- Questions from 10/14
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Quiz 1 – Active Reading Chapter 9
- Chapter 8: Multi-level Feedback Queue
  - MLFQ Scheduler
  - Job Starvation
  - **Gaming the Scheduler**
  - Examples
- Chapter 9: Proportional Share Schedulers

24

MLFQ: ISSUES - 2

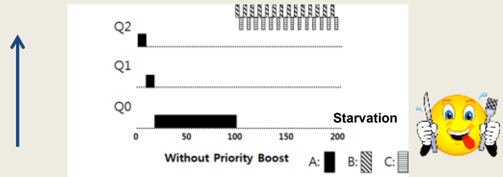
- Gaming the scheduler
  - Issue I/O operation at 99% completion of the time slice
  - Keeps job priority fixed – never lowered
- Job behavioral change
  - CPU/batch process becomes an interactive process

Priority becomes stuck



25

RESPONDING TO BEHAVIOR CHANGE

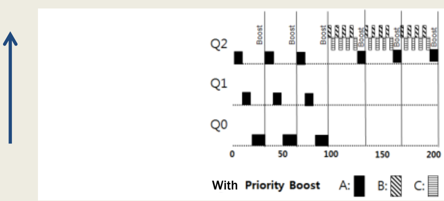


- Priority Boost
  - Reset all jobs to topmost queue after some time interval S

26

RESPONDING TO BEHAVIOR CHANGE - 2

- With priority boost
  - Prevents starvation



27

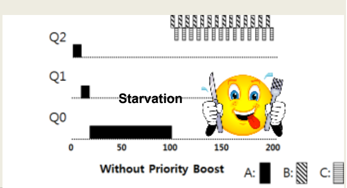
KEY TO UNDERSTANDING MLFQ – PB

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

28

STARVATION EXAMPLE

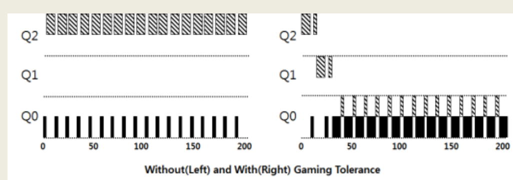
- <u>**Consider 3 queues:**</u>
- Q2 – HIGH PRIORITY – Time Quantum 10ms
- Q1 – MEDIUM PRIORITY – Time Quantum 20 ms
- Q0 – LOW PRIORITY – Time Quantum 40 ms
- Job A: 200ms no I/O
- Job B: 5ms then I/O
- Job C: 5ms then I/O
- Q2 fills up, starves Q1 & Q0
- A makes no progress



29

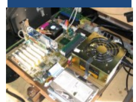
PREVENTING GAMING

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



30

WE WILL RETURN AT 2:45PM

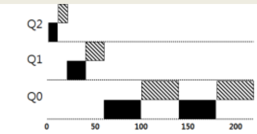


October 19, 2021 | TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma | L6.31

31

MLFQ: TUNING

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



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

October 19, 2021 | TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma | L6.32

32

PRACTICAL EXAMPLE

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

October 19, 2021 | TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma | L6.33

33

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.

October 19, 2021 | TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma | L6.34

34

OBJECTIVES – 10/19

- Questions from 10/14
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Quiz 1 – Active Reading Chapter 9
- Chapter 8: Multi-level Feedback Queue
  - MLFQ Scheduler
  - Job Starvation
  - Gaming the Scheduler
  - **Examples**
- Chapter 9: Proportional Share Schedulers

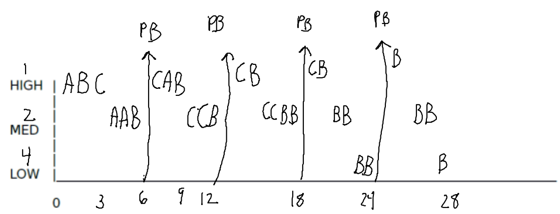
October 19, 2021 | TCSS422: Operating Systems [Fall 2021] School of Engineering and Technology, University of Washington - Tacoma | L6.35

35

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

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

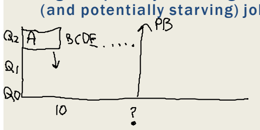
(11 points) Show a scheduling graph for the MLFQ scheduler for the jobs above. Draw vertical lines for key events and be sure to label the X-axis times as in the example. Please draw clearly. An unreadable graph will loose points.



36

EXAMPLE

- Question:
- Given a system with a quantum length of 10 ms 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?



$.05\ PB = 10$

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

37

EXAMPLE

- Question:
- Given a system with a quantum length of 10 ms 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 (10 ms)
  - Batch jobs starts, runs for full quantum of 10ms
  - 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***

38

OBJECTIVES – 10/19

- Questions from 10/14
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Quiz 1 – Active Reading Chapter 9
- Chapter 8: Multi-level Feedback Queue
  - MLFQ Scheduler
  - Job Starvation
  - Gaming the Scheduler
  - Examples
- Chapter 9: Proportional Share Schedulers

39

CHAPTER 9 - PROPORTIONAL SHARE SCHEDULER

40

OBJECTIVES – 10/19

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

41

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

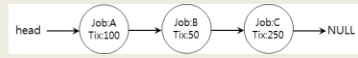
42

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

43

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...
```


44

OBJECTIVES – 10/19

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

45

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)

46

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.

47

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

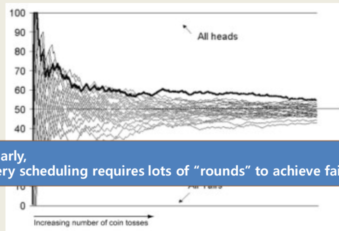
Scheduler's winning tickets:	63	85	70	39	76	17	29	41	36	39	10	99	68	83	63
Scheduled job:	A	B	A	A	B	A	A	A	A	A	A	B	A	B	A

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

48

COIN FLIPPING

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

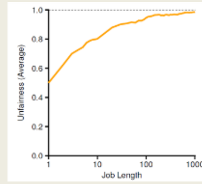


Similarly,
Lottery scheduling requires lots of "rounds" to achieve fairness.

49

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**.

50

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…

51

OBJECTIVES – 10/19

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

52

STRIDE SCHEDULER

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

53

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)

54

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...

55

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

56

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...

57

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 ...

58

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	...

59

OBJECTIVES – 10/19

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

60

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 !

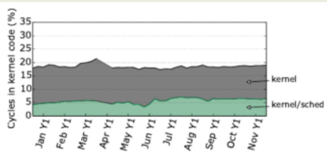


Figure 5: Kernel time, especially time spent in the scheduler, is a significant fraction of WSC cycles.

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

61

LINUX: COMPLETELY FAIR SCHEDULER (CFS)

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

62

COMPLETELY FAIR SCHEDULER - 2

- Every thread/process has a scheduling class (policy):
- **Normal classes**: SCHED_OTHER (TS), SCHED_IDLE, SCHED_BATCH
  - TS = Time Sharing
- **Real-time classes**: SCHED_FIFO (FF), SCHED_RR (RR)
- How to show scheduling class and priority:
- `#class`
  `ps -elfc`
- `#priority (nice value)`
  `ps ax -o pid,ni,cls,pri,cmd`

63

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
```


64

COMPLETELY FAIR SCHEDULER - 4

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

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

65

CFS TRADEOFF

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

  reduced context switching → less overhead
  poor near-term fairness

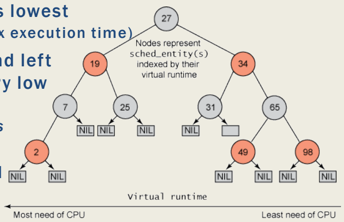
- **LOW** sched_min_granularity_ns (timeslice)
  sched_latency_ns
  sched_wakreup_granularity_ns

  increased context switching → more overhead
  better near-term fairness

66

COMPLETELY FAIR SCHEDULER - 5

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



67

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

68

COMPLETELY FAIR SCHEDULER - 6

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

69

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

70

QUESTIONS

71