

TCCS 422: OPERATING SYSTEMS

Limited Direct Execution, Introduction to Scheduling, Multilevel Feedback Queue



Wes J. Lloyd
Institute of Technology
University of Washington - Tacoma

January 22, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

OBJECTIVES

- Homework 0 Questions
- Tutorial 1: Pointers, Strings, and Exec in C
- Introducing Homework 1
- Feedback from 1/17
- Ch. 8
 - Multi-level feedback queue (MLFQ)
- Ch. 9
 - Proportional Share Scheduler, Ticket Schedulers
- Ch. 26
 - Introduction to concurrency, threads

January 22, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.2

SELECTED FEEDBACK FROM 1/17

- **What is the difference between redirecting and piping output?**
- Redirecting: STDOUT/STDERR to a file
STDIN from a file
- Piping: Send STDOUT of one program to STDIN of next
NO FILES ARE INVOLVED
- **Piping Example:** `cat mylog.txt | grep the
grep the mylog.txt`
- **Redirecting Examples:** `grep 42 log.txt > 42_occurrences.txt
grep 42 < log.txt`

January 22, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.3

FEEDBACK - 2

- Are fork(), exec(), wait() the "bread and butter" of thread management in C? Isn't fork() too slow since it copies the entire process?
- fork(), exec(), wait() support creating and merging processes in C

January 22, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.4

FEEDBACK - 3

- Interrupting interrupts (maskable)
- And not being about to interrupt interrupts (non-maskable)
- What again is a preemptive kernel?

January 22, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.5

FEEDBACK - 4

- **Jain's fairness Index**
 - Quantifies if jobs receive a fair share of system resources
- $$\mathcal{J}(x_1, x_2, \dots, x_n) = \frac{(\sum_{i=1}^n x_i)^2}{n \cdot \sum_{i=1}^n x_i^2}$$
- n processes
 - x_i is time share of each process
 - worst case = $1/n$, best case = 1
 - **Compute Example**
 - With $n=3$ and $x_1=.2, x_2=.7, x_3=.1$
 - fairness=.62
 - With $n=3$ and $x_1=.33, x_2=.33, x_3=.33$
 - fairness=1

January 22, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.6

FEEDBACK - 5

- What is the difference between **Shortest Job First** and **Shortest Job First with Random Arrival**?
 - Same scheduler
 - Random arrival is a problem parameter – when do jobs arrive?
 - All at the start?
 - OR gradually over time (*more realistic*)
- Round-Robin**
 - While this scheduler is still quite simple, there are many practical applications of RR used in practice in real systems
 - Traffic/resource load balancers commonly use “round robin” to determine where to send jobs (e.g. which server)

January 22, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.7

FEEDBACK - 6

- What is the scheduling algorithm used in a real OS? MLFQ?
 - Linux/Ubuntu uses the Completely Fair Scheduler
 - Not covered in detail in **Three Easy Places**
- Can the OS implement multiple job schedulers at the same time?
 - Absolutely, but only for scheduling different resources
 - You can have multiple levels of scheduling like MLFQ
 - At queue level, different scheduler / scheduler configs can be used
 - Example:** XEN hypervisor, used by Amazon Cloud to provide VMs:
 - BVT: Borrowed Virtual Time
 - SEDF: Simple Earliest Deadline First
 - Credit: a fair share proportional scheduler

January 22, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.8

FEEDBACK - 7

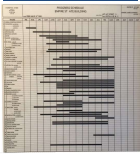
- Still confused about MLFQ and how the jobs work with queues
- When deciding what queue a process should be in, how does the OS decide?
- Is the CPU between context switches moving process in their respective queues?
 - Queues are likely pointers to struct task_struct data structures
 - Easy to move pointers between different queues

January 22, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.9

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



January 11, 2017

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.10

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

January 22, 2018

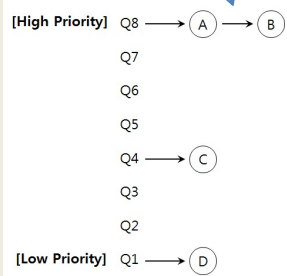
TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.11

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



January 22, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.12

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

January 22, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.13

MLFQ: LONG RUNNING JOB

- Three-queue scheduler, time slice=10ms

Priority

Q2

Q1

Q0

0 50 100 150 200

Long-running Job Over Time (msec)

January 22, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.14

MLFQ: BATCH AND INTERACTIVE JOBS

- $A_{arrival_time} = 0ms, A_{run_time} = 200ms,$
- $B_{run_time} = 20ms, B_{arrival_time} = 100ms$

Priority

Q2

Q1

Q0

0 50 100 150 200

Scheduling multiple jobs (ms)

A:

B:

January 22, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.15

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

Q2

Q1

Q0

0 50 100 150 200

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

A:

B:

January 22, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.16

MLFQ: ISSUES

- Starvation

[High Priority] Q8 → A → B → C → D → E → F

Q7

Q6

Q5

Q4

Q3

Q2

[Low Priority] Q1 → G → H CPU bound batch job(s)

January 22, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.17

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

[High Priority] Q8 → A → B → C → D → E → F

Q7

Q6

Q5

Q4

Q3

Q2

[Low Priority] Q1 → G → H CPU bound batch job(s)

January 22, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.18

RESPONDING TO BEHAVIOR CHANGE

↑

Q2

Q1

Q0

0

50

100

150

200

Starvation

A: B: C:

Without Priority Boost

■ Priority Boost

■ Reset all jobs to topmost queue after some time interval S

January 22, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.19

RESPONDING TO BEHAVIOR CHANGE - 2

■ With priority boost

■ Prevents starvation

↑

Q2

Q1

Q0

0

50

100

150

200

Without(Left) and With(Right) Priority Boost

A: B: C:

Without(Left) and With(Right) Priority Boost

January 22, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.20

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

Q2

Q1

Q0

0

50

100

150

200

Without(Left) and With(Right) Gaming Tolerance

Without(Left) and With(Right) Gaming Tolerance

January 22, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.21

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?

Q2

Q1

Q0

0

50

100

150

200

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

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

January 22, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.22

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

January 22, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.23

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.

January 22, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.24

CHAPTER 9 - PROPORTIONAL SHARE SCHEDULER

January 22, 2018

TCSS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.25

PROPORTIONAL SHARE SCHEDULER

- Also called fair-share scheduler or lottery scheduler
 - Guarantee 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 22, 2018

TCSS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.26

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 22, 2018

TCSS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.27

LOTTERY SCHEDULER IMPLEMENTATION



```

1 // counter: used to track if we've found the winner yet
2 int counter = 0;
3
4 // winner: use some call to a random number generator to
5 // get a value, between 0 and the total # of tickets
6 int winner = getrandom(0, totaltickets);
7
8 // current: use this to walk through the list of jobs
9 node_t *current = head;
10
11 // loop until the sum of ticket values is > the winner
12 while (current) {
13     counter = counter + current->tickets;
14     if (counter > winner)
15         break; // found the winner
16     current = current->next;
17 }
18 // 'current' is the winner: schedule it...
    
```

January 22, 2018

TCSS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.28

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)

January 22, 2018

TCSS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.29

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.

January 22, 2018

TCSS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.30

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?

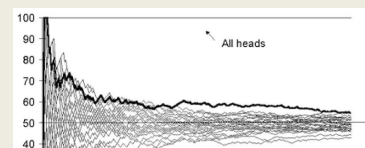
January 22, 2018

TCS5422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.31

COIN FLIPPING

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



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

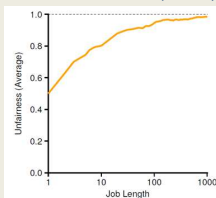
January 22, 2018

TCS5422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.32

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.

January 22, 2018

TCS5422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.33

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...

January 22, 2018

TCS5422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.34

STRIDE SCHEDULER

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

January 22, 2018

TCS5422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.35

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 $\rightarrow A_{stride} = 10000/100 = 100$
 - Job B has 50 tickets $\rightarrow B_{stride} = 10000/50 = 200$
 - Job C has 250 tickets $\rightarrow C_{stride} = 10000/250 = 40$
- Stride scheduler tracks "pass" values for each job (A, B, C)

January 22, 2018

TCS5422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.36

STRIDE SCHEDULER - 3

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

January 22, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.37

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

January 22, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.38

STRIDE SCHEDULER EXAMPLE - 2

- 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

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 22, 2018

TCCS422: Operating Systems [Winter 2018]
Institute of Technology, University of Washington - Tacoma

L5.39

QUESTIONS