TCSS 422: OPERATING SYSTEMS

Limited Direct Execution II, Intro to Schedulers

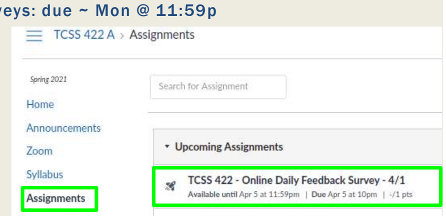
Wes J. Lloyd
School of Engineering and Technology
University of Washington - Tacoma

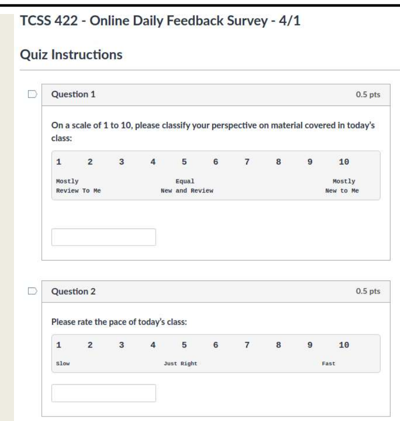
OBJECTIVES – 4/13

- Questions from 4/8
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Chapter 6: Limited Direct Execution
  - Cooperative multi-tasking
  - Context switching and preemptive multi-tasking
- Chapter 7: Scheduling Introduction
  - Scheduling metrics
    - Turnaround time, Jain's Fairness Index, Response time
  - FIFO, SJF, STCF, RR schedulers
- Chapter 8: Multi-level Feedback Queue

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





MATERIAL / PACE

- Please classify your perspective on material covered in today's class (56 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 6.91** (no change - **previous 6.91**)
- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.65 (↓ - previous 5.67)**

FEEDBACK

- ***Could you fork another child process from a child process to have a grandparent/grandchild process?***
  - YES
- ***Can the exec() function be thought of as opening another thread to a different program?***
  - NO, exec runs another program in the existing process
  - Control is transferred to another executable and is not returned
- ***Does exec transfer control back to the main program that called external program?***
  - No, the PID is transferred to the new executable and does not come back. If you want to preserve the original program the idea is to fork and run exec with a child process

FEEDBACK - 2

- ***What are the advantages of using lower-level APIs such as open() compared to the specialized versions with additional features like fopen()? Is this similar to the control tradeoff? Introducing unnecessary overhead and the like?***
  - fopen() and other functions like it are provided largely out of convenience for developers
  - Specialized wrappers such as fopen() abstract additional functionality to make it more easily accessible for programmers
- ***With the use of standard out and standard error when EXEC with file redirection, I'm still not sure about the steps from L4.30***

EXEC WITH FILE REDIRECTION (OUTPUT)

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <fcntl.h>
#include <sys/wait.h>

int
main(int argc, char *argv[]){
    int rc = fork();
    if (rc < 0) {          // fork failed; exit
        fprintf(stderr, "fork failed\n");
        exit(1);
    } else if (rc == 0) { // child: redirect standard output to a file
        close(STDOUT_FILENO);
        open("./p4.output", O_CREAT|O_WRONLY|O_TRUNC, S_IRWXU);
        …
```


FILE MODE BITS

```
S_IRWXU
read, write, execute/search by owner
S_IRUSR
read permission, owner
S_IWUSR
write permission, owner
S_IXUSR
execute/search permission, owner
S_IRWXG
read, write, execute/search by group
S_IRGRP
read permission, group
S_IWGRP
write permission, group
S_IXGRP
execute/search permission, group
S_IRWXO
read, write, execute/search by others
S_IROTH
read permission, others
S_IWOTH
write permission, others
```


EXEC W/ FILE REDIRECTION (OUTPUT) - 2

```
        …
        // now exec "wc"...
        char *myargs[3];
        myargs[0] = strdup("wc");          // program: "wc" (word count)
        myargs[1] = strdup("p4.c");        // argument: file to count
        myargs[2] = NULL;                  // marks end of array
        execvp(myargs[0], myargs);         // runs word count
    } else {                           // parent goes down this path (main)
        int wc = wait(NULL);
    }
    return 0;
}
```

```
prompt> ./p4
prompt> cat p4.output
32 109 846 p4.c
prompt>
```


FEEDBACK - 3

- ***Do system calls and traps execute with limited direct execution?***
  - These are privileged operations that are executed in the kernel, with direct execution.
- ***In layman operating systems like Windows and iOS, what processes are already trusted? How difficult is it to make other processes trusted for the purposes of LDE?"***
  - User processes by default are not trusted
  - They run with Limited Direct Execution
  - Only operating system kernel code is trusted
    - In Linux this can be the kernel itself of kernel modules

FEEDBACK - 4

- ***What is kernel? Is it the same thing as OS?***
  - Yes, the kernel in the binary executable that runs with the computer boots
  - In Linux this is the "/boot/vmlinuz" executable
  - Identify file properties of your kernel with the command:

    `sudo file /boot/vmlinuz-$(uname –r)`
- ***Can you please explain the differences between the kernel, hardware, and the program?***
- The Linux kernel is the binary file /boot/vmlinuz… that is largely written in C & Assembly that implements the Linux OS
- The hardware is the laptop, desktop, or virtual machine
- The program is a user program that you write such as the examples we've reviewed in class (i.e. fork.c)

FEEDBACK - 5

- ***What are trap tables and trap handlers?"***
  - **TRAP TABLE:**
    The x86 processor uses a table known as the interrupt descriptor table (IDT) to determine how to transfer control when a trap occurs. The x86 allows up to 256 different interrupt or exception entry points into the kernel, each with a different interrupt vector.
  - **TRAP HANDLERS:**
    Trap handlers are OS kernel functions that are pointed to by the trap table. These are "event handlers" that respond to various traps.
- ***What is the difference between system APIs and system calls? Are Fork(), wait(), exec() both system APIs and system calls?***
  - A system call is the action of actually calling the system API
  - A system API is the function as defined in the OS

OBJECTIVES – 4/13

- Questions from 4/8
- **Assignment 0**
- C Tutorial - Pointers, Strings, Exec in C
- Chapter 6: Limited Direct Execution
  - Cooperative multi-tasking
  - Context switching and preemptive multi-tasking
- Chapter 7: Scheduling Introduction
  - Scheduling metrics
    - Turnaround time, Jain's Fairness Index, Response time
  - FIFO, SJF, STCF, RR schedulers
- Chapter 8: Multi-level Feedback Queue

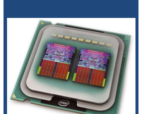
OBJECTIVES – 4/13

- Questions from 4/8
- Assignment 0
- **C Tutorial - Pointers, Strings, Exec in C**
- Chapter 6: Limited Direct Execution
  - Cooperative multi-tasking
  - Context switching and preemptive multi-tasking
- Chapter 7: Scheduling Introduction
  - Scheduling metrics
    - Turnaround time, Jain's Fairness Index, Response time
  - FIFO, SJF, STCF, RR schedulers
- Chapter 8: Multi-level Feedback Queue

OBJECTIVES – 4/13

- Questions from 4/8
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Chapter 6: Limited Direct Execution
  - **Cooperative multi-tasking**
  - Context switching and preemptive multi-tasking
- Chapter 7: Scheduling Introduction
  - Scheduling metrics
    - Turnaround time, Jain's Fairness Index, Response time
  - FIFO, SJF, STCF, RR schedulers
- Chapter 8: Multi-level Feedback Queue

CH. 6: LIMITED DIRECT EXECUTION



MULTITASKING

- How/when should the OS regain control of the CPU to switch between processes?
- Cooperative multitasking (mostly pre 32-bit)
  - < Windows 95, Mac OSX
  - Opportunistic: running programs must give up control
    - User programs must call a special **yield** system call
    - When performing I/O
    - Illegal operations
  - (POLLEV)
    What problems could you for see with this approach?

MULTITASKING

- How/when should the OS regain control of the CPU to switch between processes?
- Cooperative multitasking (mostly pre 32-bit)
  - <
  - Op
    - When performing I/O
    - Illegal operations
  - (POLLEV)
    What problems could you for see with this approach?

A process gets stuck in an infinite loop.
→ **Reboot the machine**

What problems exist for regaining the control of the CPU with cooperative multitasking OSes?

QUESTION: MULTITASKING

- What problems exist for regaining the control of the CPU with cooperative multitasking OSes?

MULTITASKING - 2

- Preemptive multitasking (32 & 64 bit OSes)
- >= Mac OSX, Windows 95+
- Timer interrupt
  - Raised at some regular interval (in ms)
  - Interrupt handling
    1. Current program is halted
    2. Program states are saved
    3. OS Interrupt handler is run (kernel mode)
- (PollEV) What is a good interval for the timer interrupt?

MULTITASKING - 2

- Preemptive multitasking (32 & 64 bit OSes)
- >= Mac OSX, Windows 95+
- Timer
  - Rais
  - Inte
    1. Current program is halted
    2. Program states are saved
    3. OS Interrupt handler is run (kernel mode)
- (PollEV) What is a good interval for the timer interrupt?

A **timer interrupt** gives OS the ability to run again on a CPU.

For an OS that uses a system timer to force arbitrary context switches to share the CPU, what is a good value (in seconds) for the timer interrupt?

QUESTION: TIME SLICE

- For an OS that uses a system timer to force arbitrary context switches to share the CPU, what is a good value (in seconds) for the timer interrupt?

OBJECTIVES – 4/13

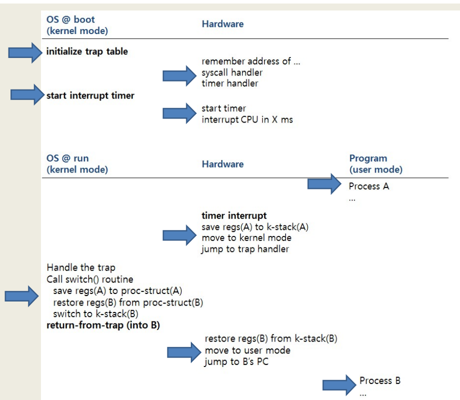
- Questions from 4/8
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Chapter 6: Limited Direct Execution
  - Cooperative multi-tasking
  - **Context switching and preemptive multi-tasking**
- Chapter 7: Scheduling Introduction
  - Scheduling metrics
    - Turnaround time, Jain's Fairness Index, Response time
  - FIFO, SJF, STCF, RR schedulers
- Chapter 8: Multi-level Feedback Queue

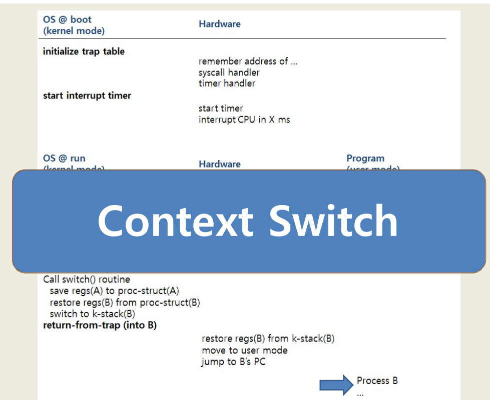
CONTEXT SWITCH

- Preemptive multitasking initiates "trap" into the OS code to determine:
  - Whether to continue running the **current process**, or switch to a **different one**.
  - If the decision is made to switch, the OS performs a <u>context switch</u> swapping out the current process for a new one.

CONTEXT SWITCH - 2

1. Save register values of the current process to its kernel stack
   - General purpose registers
   - PC: program counter (instruction pointer)
   - kernel stack pointer
2. Restore soon-to-be-executing process from its kernel stack
3. Switch to the kernel stack for the soon-to-be-executing process





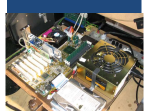
INTERRUPTED INTERRUPTS

- What happens if during an interrupt (trap to kernel mode), another interrupt occurs?
- Linux
  - < 2.6 kernel: non-preemptive kernel
  - >= 2.6 kernel: preemptive kernel

PREEMPTIVE KERNEL

- Use "locks" as markers of regions of non-preemptibility (non-maskable interrupt)
- Preemption counter (`preempt_count`)
  - begins at zero
  - increments for each lock acquired (not safe to preempt)
  - decrements when locks are released
- Interrupt can be interrupted when `preempt_count=0`
  - It is safe to preempt (maskable interrupt)
  - the interrupt is more important

WE WILL RETURN AT 5:10PM



OBJECTIVES – 4/13

- Questions from 4/8
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Chapter 6: Limited Direct Execution
  - Cooperative multi-tasking
  - Context switching and preemptive multi-tasking
- **Chapter 7: Scheduling Introduction**
  - **Scheduling metrics**
    - **Turnaround time**, **Jain's Fairness Index**, Response time
  - FIFO, SJF, STCF, RR schedulers
- Chapter 8: Multi-level Feedback Queue

CHAPTER 7- SCHEDULING: INTRODUCTION

SCHEDULING METRICS

- **Metrics**: A standard measure to quantify to what degree a system possesses some property. Metrics provide *repeatable* techniques to quantify and compare systems.
- **Measurements** are the numbers derived from the application of metrics
- Scheduling Metric #1: **Turnaround time**
- The time at which the job completes minus the time at which the job arrived in the system

$$T_{turnaround} = T_{completion} - T_{arrival}$$

- How is turnaround time different than execution time?

SCHEDULING METRICS - 2

- Scheduling Metric #2: **Fairness**
  - Jain's fairness index
  - Quantifies if jobs receive a fair share of system resources

$$\mathcal{J}(x_1, x_2, \ldots, x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \cdot \sum_{i=1}^{n} {x_i}^2}$$

- n processes
- x_i is time share of each process
- worst case = 1/n
- best case = 1

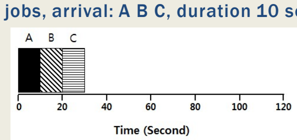
- Consider n=3, worst case = .333, best case=1
- With n=3 and x_1=.2, x_2=.7, x_3=.1, fairness=.62
- With n=3 and x_1=.33, x_2=.33, x_3=.33, fairness=1

OBJECTIVES – 4/13

- Questions from 4/8
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Chapter 6: Limited Direct Execution
  - Cooperative multi-tasking
  - Context switching and preemptive multi-tasking
- Chapter 7: Scheduling Introduction
  - Scheduling metrics
    - Turnaround time, Jain's Fairness Index, Response time
  - FIFO SJF, STCF, RR schedulers
- Chapter 8: Multi-level Feedback Queue

SCHEDULERS

- FIFO: first in, first out
  - Very simple, easy to implement
- Consider
  - 3 x 10sec jobs, arrival: A B C, duration 10 sec each



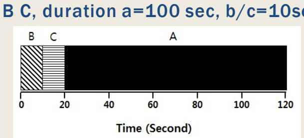
$$Average\ turnaround\ time = \frac{10+20+30}{3} = 20\ sec$$

OBJECTIVES – 4/13

- Questions from 4/8
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Chapter 6: Limited Direct Execution
  - Cooperative multi-tasking
  - Context switching and preemptive multi-tasking
- Chapter 7: Scheduling Introduction
  - Scheduling metrics
    - Turnaround time, Jain's Fairness Index, Response time
  - FIFO, SJF STCF, RR schedulers
- Chapter 8: Multi-level Feedback Queue

SJF: SHORTEST JOB FIRST

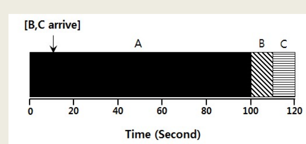
- Given that we know execution times in advance:
  - Run in order of duration, shortest to longest
  - Non preemptive scheduler
  - This is not realistic
  - Arrival: A B C, duration a=100 sec, b/c=10sec



$$Average\ turnaround\ time = \frac{10+20+120}{3} = 50\ sec$$

SJF: WITH RANDOM ARRIVAL

- If jobs arrive at any time: duration a=100s, b/c=10s
- A @ t=0sec, B @ t=10sec, C @ t=10sec



$$Average\ turnaround\ time = \frac{100+(110-10)+(120-10)}{3} = 103.33\ sec$$

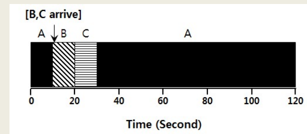
OBJECTIVES – 4/13

- Questions from 4/8
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Chapter 6: Limited Direct Execution
  - Cooperative multi-tasking
  - Context switching and preemptive multi-tasking
- Chapter 7: Scheduling Introduction
  - Scheduling metrics
    - Turnaround time, Jain's Fairness Index, Response time
  - FIFO, SJF, STCF, RR schedulers
- Chapter 8: Multi-level Feedback Queue

April 13, 2021 | TCSS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma | L5.43

SCTF: SHORTEST TIME TO COMPLETION FIRST

- Consider: duration a=100sec, b/c=10sec
  - A_{len}=100 $A_{arrival}$=0
  - B_{len}=10, $B_{arrival}$=10, C_{len}=10, $C_{arrival}$=10



$$Average\ turnaround\ time = \frac{(120-0)+(20-10)+(30-10)}{3} = 50\ sec$$

April 13, 2021 | TCSS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma | L5.44

OBJECTIVES – 4/13

- Questions from 4/8
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Chapter 6: Limited Direct Execution
  - Cooperative multi-tasking
  - Context switching and preemptive multi-tasking
- **Chapter 7: Scheduling Introduction**
  - **Scheduling metrics**
    - Turnaround time, Jain's Fairness Index, **Response time**
  - FIFO, SJF, STCF, RR schedulers
- Chapter 8: Multi-level Feedback Queue

April 13, 2021 | TCSS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma | L5.45

SCHEDULING METRICS - 3

- Scheduling Metric #3: **Response Time**
- Time from when job arrives until it starts execution

$$T_{response} = T_{firstrun} - T_{arrival}$$

- STCF, SJF, FIFO
  - can perform poorly with respect to response time

What scheduling algorithm(s) can help minimize response time?

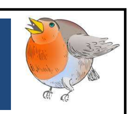
April 13, 2021 | TCSS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma | L5.46

OBJECTIVES – 4/13

- Questions from 4/8
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Chapter 6: Limited Direct Execution
  - Cooperative multi-tasking
  - Context switching and preemptive multi-tasking
- Chapter 7: Scheduling Introduction
  - Scheduling metrics
    - Turnaround time, Jain's Fairness Index, Response time
  - FIFO, SJF, STCF, RR schedulers
- Chapter 8: Multi-level Feedback Queue

April 13, 2021 | TCSS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma | L5.47

RR: ROUND ROBIN



- Run each job awhile, then switch to another distributing the CPU evenly (fairly)
- Scheduling Quantum is called a time slice
- Time ... a mu... time ... period.

Process	Burst Time
P1	12

RR is fair, but performs poorly on metrics such as turnaround time

Round Robin scheduling algorithm Gantt chart

Scheduling Quantum = 5 seconds

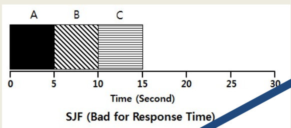
P1	P2	P3	P4	P5	P1	P2	P4	P1

0 5 10 14 19 24 29 32 37 39

April 13, 2021 | TCSS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma | L5.48

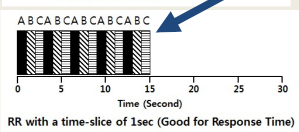
RR EXAMPLE

- ABC arrive at time=0, each run for 5 seconds



OVERHEAD not considered

$$T_{average\ response} = \frac{0+5+10}{3} = 5sec$$



$$T_{average\ response} = \frac{0+1+2}{3} = 1sec$$

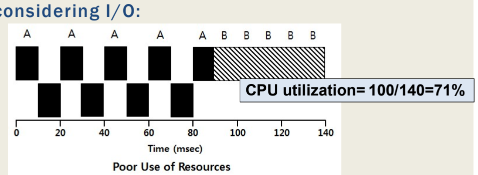
ROUND ROBIN: TRADEOFFS

Short Time Slice	Long Time Slice
Fast Response Time	Slow Response Time
High overhead from context switching	Low overhead from context switching

- Time slice impact:
  - Turnaround time (for earlier example): ts(1,2,3,4,5)=14,14,13,14,10
  - Fairness: round robin is always fair, J=1

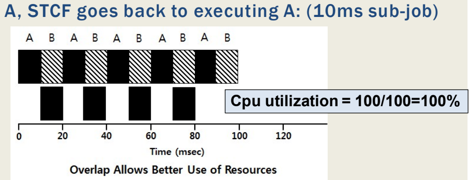
SCHEDULING WITH I/O

- STCF scheduler
  - A: CPU=50ms, I/O=40ms, 10ms intervals
  - B: CPU=50ms, I/O=0ms
  - Consider A as 10ms subjobs (CPU, then I/O)
- Without considering I/O:



SCHEDULING WITH I/O - 2

- When a job initiates an I/O request
  - A is blocked, waits for I/O to compute, frees CPU
  - STCF scheduler assigns B to CPU
- When I/O completes → raise interrupt
  - Unblock A, STCF goes back to executing A: (10ms sub-job)



Which scheduler, thus far, best address fairness and average response time of jobs?

Respond at **PollEv.com/wesleylloyd641**
Text **WESLEYLLOYD641** to **22333** once to join, then **1, 2, 3, 4, 5...**

- First In - First Out (FIFO) **1**
- Shortest Job First (SJF) **2**
- Shortest Time to Completion First (STCF) **3**
- Round Robin **4**
- None of the Above **5**
- All of the Above **6**

Start the presentation to see live content. Still no live content? Install the app or get help at **PollEv.com/app**

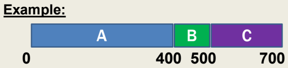
QUESTION: SCHEDULING FAIRNESS

- Which scheduler, this far, best addresses fairness and average response time of jobs?
- First In – First Out (FIFO)
- Shortest Job First (SJF)
- Shortest Time to Completion First (STCF)
- Round Robin (RR)
- None of the Above
- All of the Above

SCHEDULING METRICS

- Consider Three jobs (A, B, C) that require: $time_A$=400ms, $time_B$=100ms, and $time_C$=200ms
- All jobs arrive at time=0 in the sequence of A B C.
- Draw a scheduling graph to help compute the average response time (ART) and average turnaround time (ATT) scheduling metrics for the FIFO scheduler.

Example:



April 13, 2021 | TCSS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma | L5.55

When poll is active, respond at **PollEv.com/wesleylloyd641**
Text **WESLEYLLOYD641** to **22333** once to join

What is the Average Response Time of the FIFO scheduler?

Start the presentation to see live content. Still no live content? Install the app or get help at **PollEv.com/app**

When poll is active, respond at **PollEv.com/wesleylloyd641**
Text **WESLEYLLOYD641** to **22333** once to join

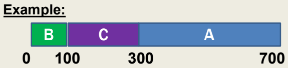
What is the Average Turnaround Time of the FIFO scheduler?

Start the presentation to see live content. Still no live content? Install the app or get help at **PollEv.com/app**

SCHEDULING METRICS

- Consider Three jobs (A, B, C) that require: $time_A$=400ms, $time_B$=100ms, and $time_C$=200ms
- All jobs arrive at time=0 in the sequence of A B C.
- Draw a scheduling graph to help compute the average response time (ART) and average turnaround time (ATT) scheduling metrics for the SJF scheduler.

Example:



April 13, 2021 | TCSS422: Operating Systems [Spring 2021] School of Engineering and Technology, University of Washington - Tacoma | L5.58

When poll is active, respond at **PollEv.com/wesleylloyd641**
Text **WESLEYLLOYD641** to **22333** once to join

What is the Average Response Time of the Shortest Job First Scheduler?

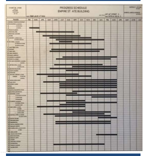
Start the presentation to see live content. Still no live content? Install the app or get help at **PollEv.com/app**

When poll is active, respond at **PollEv.com/wesleylloyd641**
Text **WESLEYLLOYD641** to **22333** once to join

What is the Average Turnaround Time of the Shortest Job First Scheduler?

Start the presentation to see live content. Still no live content? Install the app or get help at **PollEv.com/app**

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



OBJECTIVES

- Chapter 8: Multi-level Feedback Queue
  - MLFQ Scheduler
  - Job Starvation
  - Gaming the Scheduler
  - Examples

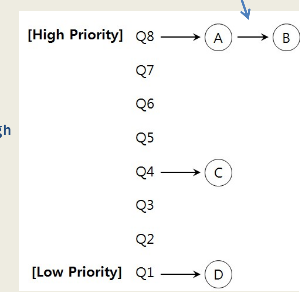
MULTI-LEVEL FEEDBACK QUEUE

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

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



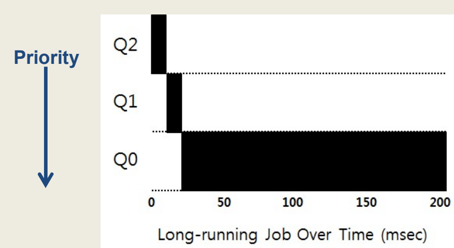
MLFQ: DETERMINING JOB PRIORITY

- 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

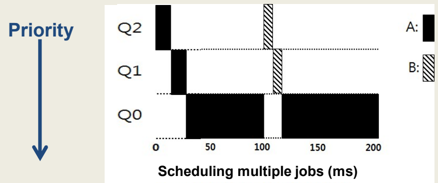
MLFQ: LONG RUNNING JOB

- Three-queue scheduler, time slice=10ms



MLFQ: BATCH AND INTERACTIVE JOBS

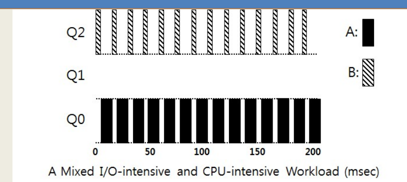
- $A_{arrival\_time}$=0ms, A_{run_time}=200ms,
- B_{run_time} =20ms, $B_{arrival\_time}$ =100ms



MLFQ: BATCH AND INTERACTIVE - 2

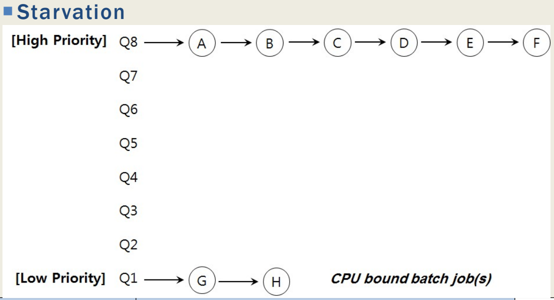
- Continuous interactive job (B) with long running batch job (A)
  - Low response time is good for B
  - A continues to make progress

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



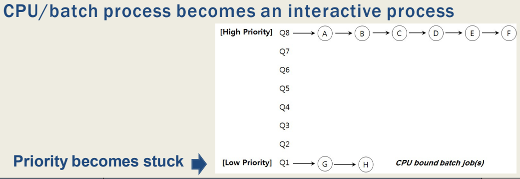
MLFQ: ISSUES

- Starvation

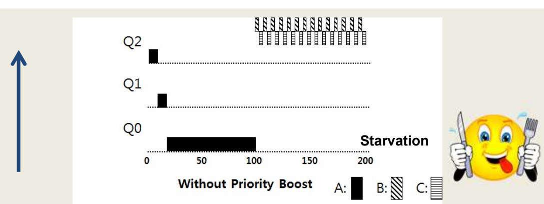


MLFQ: ISSUES - 2

- 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



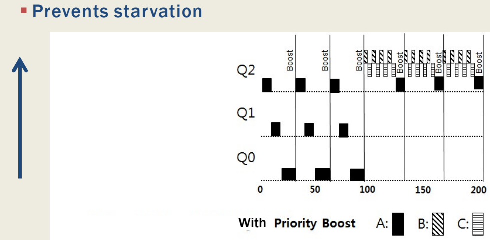
RESPONDING TO BEHAVIOR CHANGE



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

RESPONDING TO BEHAVIOR CHANGE - 2

- With priority boost
  - Prevents starvation

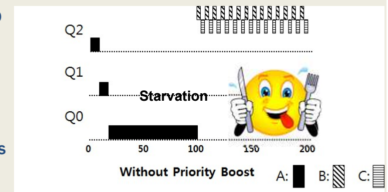


KEY TO UNDERSTANDING MLFQ – PB

- 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.
- **KEY**: If time quantum of a higher queue is filled, then we don't run any jobs in lower priority queues!!!

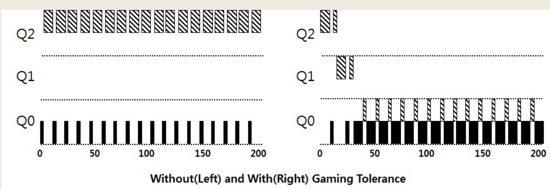
STARVATION EXAMPLE

- **Consider 3 queues:**
- 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



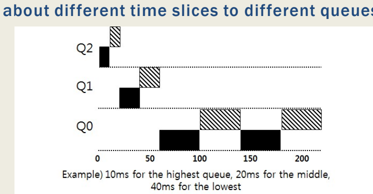
PREVENTING GAMING

- 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



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?



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

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.

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.

HIGH |
MED |
LOW |______________________________
0

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***

QUESTIONS