TCSS 422: OPERATING SYSTEMS

Introduction to OS Schedulers

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

April 11, 2023 | TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

1

TEXT BOOK COUPON

- 10% off textbook code: **LIBRARY10** (*through Friday Apr 14*)
- https://www.lulu.com/shop/andrea-arpaci-dusseau-and-remzi-arpaci-dusseau/operating-systems-three-easy-pieces-softcover-version-100/paperback/product-14mjrrgk.html
- With coupon textbook is only $19.80 + tax & shipping

April 11, 2023 | TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma | L5.2

2

OFFICE HOURS – SPRING 2023

- **Tuesdays:**
  - 2:30 to 3:30 pm - CP 229 / Zoom
- **Fridays**
  - *1:30 to 2:30 pm – Zoom / (CP 229-on some days)
- Also available after class
- Or email for appointment

> Office Hours set based on Student Demographics survey feedback
** time may be occasionally rescheduled due to faculty meeting conflicts*

3

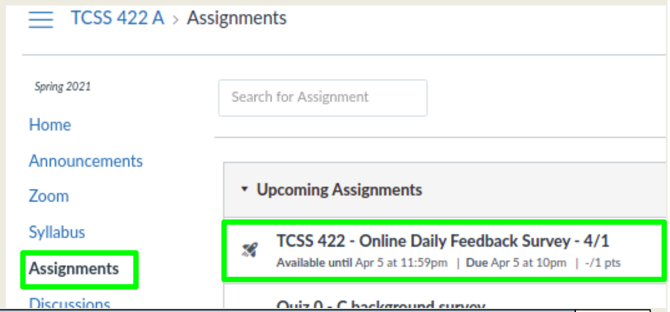
OBJECTIVES – 4/11

- Questions from 4/6
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Chapter 6: Limited Direct Execution
- 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
  - MLFQ Scheduler
  - Job Starvation
  - Gaming the Scheduler
  - Examples

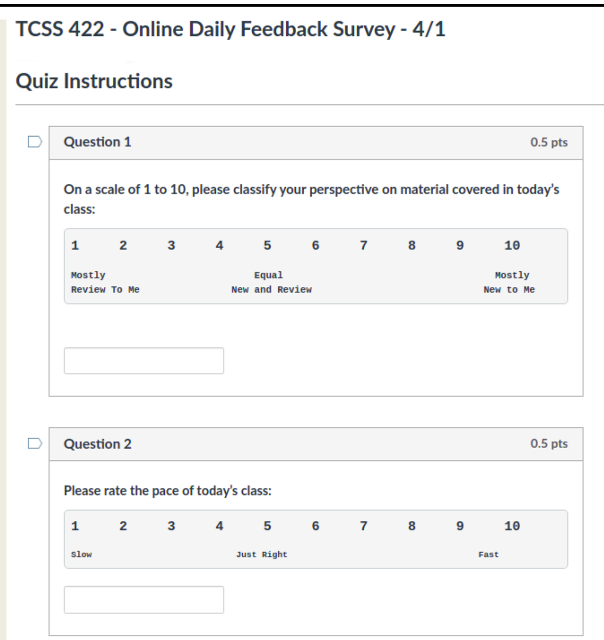
4

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 (52 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 7.58 (↑ - previous 7.20)**

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 6.08 (↑ - previous 5.42)**

FEEDBACK FROM 4/6

- ***Is it possible to have multiple standard outputs working simultaneously in the same process? For instance, one writing to the terminal and another one writing to a file.***
- No – standard output will go to the console (/dev/console) or (/dev/tty0) or to a file
- It is possible to have "standard output" go to the console, and "standard err" go to a file

FEEDBACK - 2

- ***Are the Operating System and the C language 2 separate entities?***
- Yes, the operating system is the kernel program that is primarily written in C (with some assembly portions)

- ***From the class code we are still importing c libraries. Does that mean there is no need for low level assembly to build the operating System?***
  - Most of Linux is written in C
  - There is some inline assembly in the kernel code
  - See example use of 'asm' and .S files (assembly):
    https://github.com/torvalds/linux/blob/master/arch/x86/boot/cpuflags.c
    https://github.com/torvalds/linux/blob/master/arch/x86/boot/

April 11, 2023	TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma	L5.9

9

FEEDBACK - 3

- ***I would like to see how control goes back to the parent if wait() and exec() are both used (exec in the child).***
- wait() and waitpid() are called by the parent to wait for the child process to exit
- When exec() is called the "control" of the child process is transferred to another executable.
- That means the original executable code is no longer run by the child.
- When the external program exits, the parent will trap this with wait() or waitpid().

April 11, 2023	TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma	L5.10

10

FEEDBACK - 4

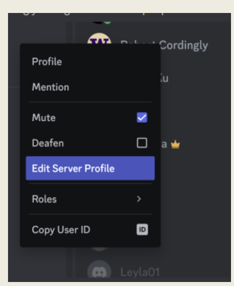
- ***How much work/time is involved in forking a process?***
- Forking a process, clones the process into two processes, the parent, and a child
- The cloning of memory is typically "lazy"
- Essentially memory is only cloned (copied) on write Copy-On-Write (COW) to save time
- ***Is forking the current process and then calling exec the most efficient way to start a new program without losing the current program?***
- The system() API call runs another program and captures the output in the same process
- No new process is created
- This is probably more efficient than cloning a process
- It would be possible to write test code to compare runtime of system() vs. fork() + exec() to run another program

April 11, 2023	TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma	L5.11

11

TCSS 422 DISCORD SERVER

- Please join the TCSS 422 A – Spring 2023 Discord Server
- https://discord.gg/hqNanxEQ
- Under Edit Server Profile:
  Please update your 'Server Nickname' to your real name or UW NET ID
  THANK YOU



April 11, 2023	TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma	L5.12

12

OBJECTIVES – 4/11

- Questions from 4/6
- **Assignment 0**
- C Tutorial - Pointers, Strings, Exec in C
- Chapter 6: Limited Direct Execution
- 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
  - MLFQ Scheduler
  - Job Starvation
  - Gaming the Scheduler
  - Examples

April 11, 2023	TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma	L5.13

13

FEEDBACK ON ASSIGNMENT 0

- ***In the homework, it specifies to use "non-interactive" commands. What does this mean exactly?***
- An non-interactive command does not require any input from the user (i.e. from the keyboard)
- Non-interactive commands and scripts can run entirely on their own without intervention
- These commands are considered "headless" in that they don't feature a USER INTERFACE, either a GUI, or TUI
- **What is a TUI?**
  - ****Text-based User Interface***
    - **TUI is also a bird** →

April 11, 2023	TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma	L5.14

14

FEEDBACK - 2

- ***My laptop is Apple M1 and the version of Ubuntu is 22.04.2 LTS. I was trying to look up the CPU model name from the VM, and it does not show up in my output. I'm wondering if it is due to M1, and is there any possible way for me to address the problem?***
- The ARM version of Ubuntu does not have the ability to identify the Model Name of M1/M2 Mac processors.
- You can likely find the CPU model from "About this Mac" from the MacOS.
- Additionally, you may be able to learn about the processor from the wikipedia pages:
- https://en.wikipedia.org/wiki/Apple_M1
- https://en.wikipedia.org/wiki/Apple_M2

TCSS 422 – SET VMS

- School of Engineering and Technology hosted Ubuntu 22.04 VMs for TCSS 422 – Spring 2023 are created
- Connection information on how to access SET VMs has been emailed to students who requested BMs

OBJECTIVES – 4/11

- Questions from 4/6
- Assignment 0
- **C Tutorial - Pointers, Strings, Exec in C**
- Chapter 6: Limited Direct Execution
- 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
  - MLFQ Scheduler
  - Job Starvation
  - Gaming the Scheduler
  - Examples

April 11, 2023	TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma	L5.17

17

OBJECTIVES – 4/11

- Questions from 4/6
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- **Chapter 6: Limited Direct Execution**
- 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
  - MLFQ Scheduler
  - Job Starvation
  - Gaming the Scheduler
  - Examples

April 11, 2023	TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma	L5.18

18

CH. 6: LIMITED DIRECT EXECUTION

April 11, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L5.19

19

CHAPTER 6

- Chapter 6: Limited Direct Execution
  - Direct execution
  - **Limited direct execution**
  - CPU modes
  - System calls and traps
  - Cooperative multi-tasking
  - Context switching and preemptive multi-tasking

April 11, 2023

TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma

L5.20

20

LIMITED DIRECT EXECUTION

- OS implements LDE to support time/resource sharing
- Limited direct execution means "only limited" processes can execute DIRECTLY on the CPU in ***trusted*** mode
- TRUSTED means the process is trusted, and it can do anything… (e.g. it is a system / kernel level process)
- Enabled by ***protected (safe) control transfer***
- CPU supported context switch
- Provides data isolation

CHAPTER 6

- Chapter 6: Limited Direct Execution
  - Direct execution
  - Limited direct execution
  - **CPU modes**
  - System calls and traps
  - Cooperative multi-tasking
  - Context switching and preemptive multi-tasking

CPU MODES

- Utilize CPU Privilege Rings (Intel x86)
  - rings 0 (kernel), 1 (VM kernel), 2 (unused), 3 (user)

access ⟵ no access

- **User mode**:
  Application is running, but w/o direct I/O access
- **Kernel mode**:
  OS kernel is running performing restricted operations

23

CPU MODES

- **User mode: ring 3 - untrusted**
  - Some instructions and registers are disabled by the CPU
  - Exception registers
  - HALT instruction
  - MMU instructions
  - OS memory access
  - I/O device access
- **Kernel mode: ring 0 – trusted**
  - All instructions and registers enabled

24

CHAPTER 6

- Chapter 6: Limited Direct Execution
  - Direct execution
  - Limited direct execution
  - CPU modes
  - **System calls and traps**
  - Cooperative multi-tasking
  - Context switching and preemptive multi-tasking

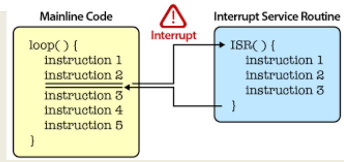
25

SYSTEM CALLS

- Implement restricted “OS” operations
- Kernel exposes key functions through an API:
  - Device I/O (e.g. file I/O)
  - Task swapping: context switching between processes
  - Memory management/allocation: malloc()
  - Creating/destroying processes

26

TRAPS: SYSTEM CALLS, EXCEPTIONS, INTERRUPTS



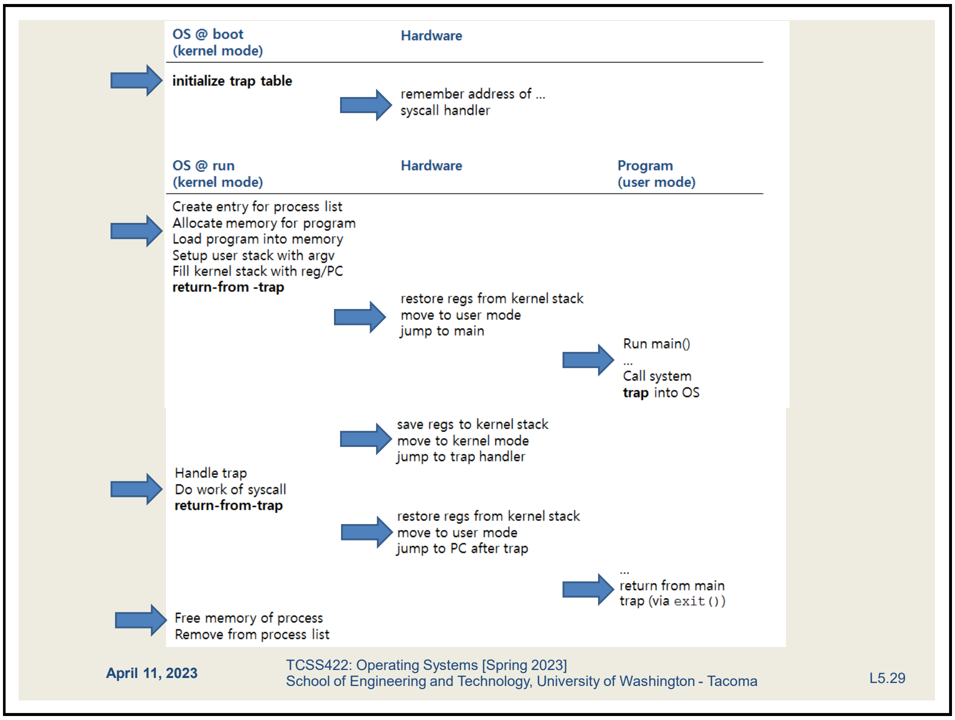
- Trap: any transfer to kernel mode
- Three kinds of traps
  - **System call:** (planned) user → kernel
    - SYSCALL for I/O, etc.
  - **Exception:** (error) user → kernel
    - Div by zero, page fault, page protection error
  - **Interrupt:** (event) user → kernel
    - Non-maskable vs. maskable
    - Keyboard event, network packet arrival, timer ticks
    - Memory parity error (ECC), hard drive failure

27

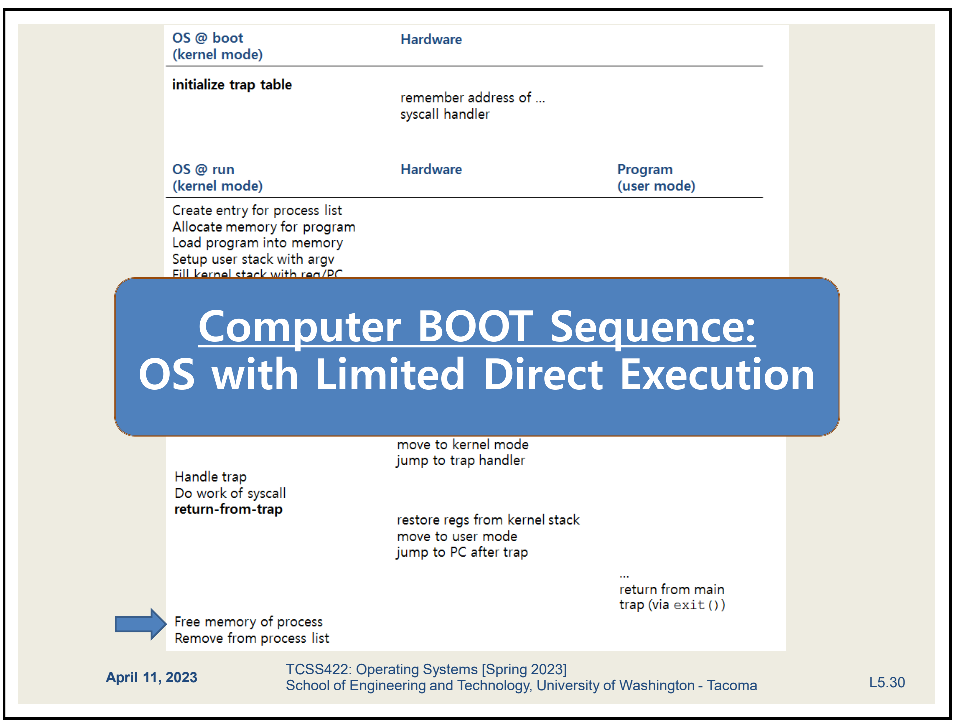
EXCEPTION TYPES

Exception type	Synchronous vs. asynchronous	User request vs. coerced	User maskable vs. nonmaskable	Within vs. between instructions	Resume vs. terminate
I/O device request	Asynchronous	Coerced	Nonmaskable	Between	Resume
Invoke operating system	Synchronous	User request	Nonmaskable	Between	Resume
Tracing instruction execution	Synchronous	User request	User maskable	Between	Resume
Breakpoint	Synchronous	User request	User maskable	Between	Resume
Integer arithmetic overflow	Synchronous	Coerced	User maskable	Within	Resume
Floating-point arithmetic overflow or underflow	Synchronous	Coerced	User maskable	Within	Resume
Page fault	Synchronous	Coerced	Nonmaskable	Within	Resume
Misaligned memory accesses	Synchronous	Coerced	User maskable	Within	Resume
Memory protection violation	Synchronous	Coerced	Nonmaskable	Within	Resume
Using undefined instruction	Synchronous	Coerced	Nonmaskable	Within	Terminate
Hardware malfunction	Asynchronous	Coerced	Nonmaskable	Within	Terminate
Power failure	Asynchronous	Coerced	Nonmaskable	Within	Terminate

28


OS @ boot
(kernel mode)
Hardware
initialize trap table
remember address of …
syscall handler
OS @ run
(kernel mode)
Hardware
Program
(user mode)
Create entry for process list
Allocate memory for program
Load program into memory
Setup user stack with argv
Fill kernel stack with reg/PC
return-from -trap
restore regs from kernel stack
move to user mode
jump to main
Run main()
…
Call system
trap into OS
save regs to kernel stack
move to kernel mode
jump to trap handler
Handle trap
Do work of syscall
return-from-trap
restore regs from kernel stack
move to user mode
jump to PC after trap
…
return from main
trap (via exit())
Free memory of process
Remove from process list
April 11, 2023
TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma
L5.29

29


OS @ boot
(kernel mode)
Hardware
initialize trap table
remember address of …
syscall handler
OS @ run
(kernel mode)
Hardware
Program
(user mode)
Create entry for process list
Allocate memory for program
Load program into memory
Setup user stack with argv
Computer BOOT Sequence:
OS with Limited Direct Execution
move to kernel mode
jump to trap handler
Handle trap
Do work of syscall
return-from-trap
restore regs from kernel stack
move to user mode
jump to PC after trap
…
return from main
trap (via exit())
Free memory of process
Remove from process list
April 11, 2023
TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma
L5.30

30

CHAPTER 6

- Chapter 6: Limited Direct Execution
  - Direct execution
  - Limited direct execution
  - CPU modes
  - System calls and traps
  - **Cooperative multi-tasking**
  - Context switching and preemptive multi-tasking

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?

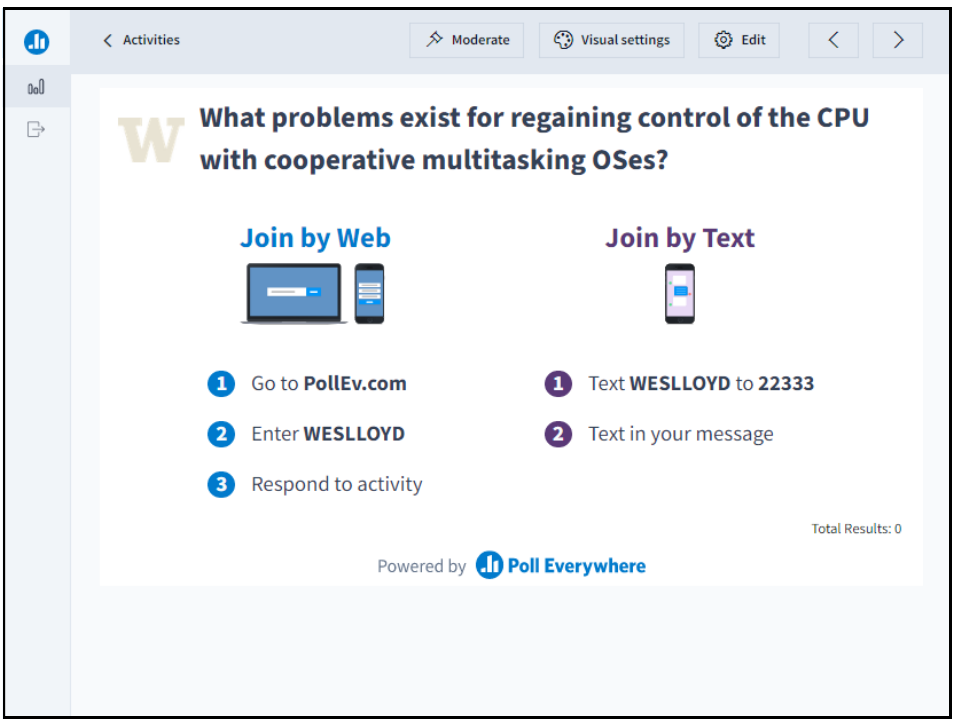
MULTITASKING

- How/when should the OS regain control of the CPU to switch between processes?
- Coop
  - <
  - 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**

April 11, 2023	TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma	L5.33

33



34

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
  - Inter
    1. Current program is halted
    2. Program states are saved
    3. OS Interrupt handler is run (kernel mode)

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

- (PollEV) What is a good interval for the timer interrupt?

37

Activities | Moderate | Visual settings | Edit

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?

Join by Web

1. Go to **PollEv.com**
2. Enter **WESLLOYD**
3. Respond to activity

Join by Text

1. Text **WESLLOYD** to **22333**
2. Text in your message

Total Results: 0

Powered by **Poll Everywhere**

38

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?

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?
  - Typical time slice for process execution is **10 to 100 milliseconds**
  - Typical context switch overhead is (*switch between processes*) **0.01 milliseconds**
    - 0.1% of the time slice (1/1000th)

CHAPTER 6

- Chapter 6: Limited Direct Execution
  - Direct execution
  - Limited direct execution
  - CPU modes
  - System calls and traps
  - Cooperative multi-tasking
  - **Context switching and preemptive multi-tasking**

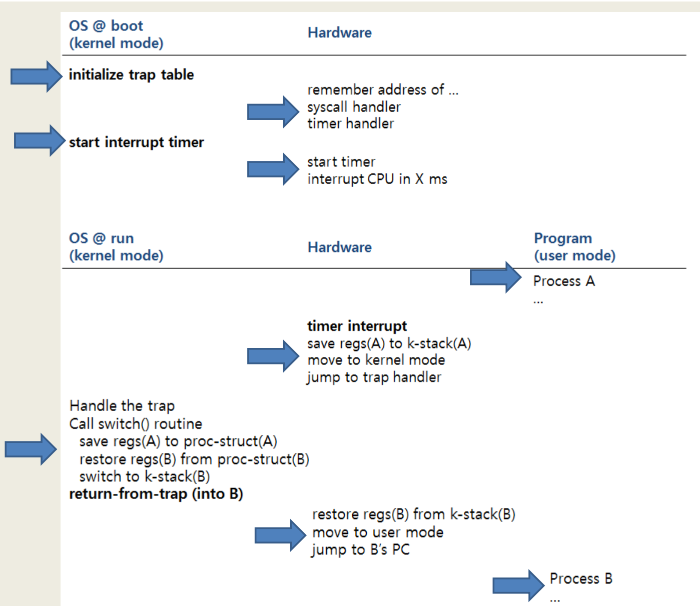
CONTEXT SWITCH

- 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 context switch 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



OS @ boot (kernel mode)	Hardware	
initialize trap table		
	remember address of … syscall handler timer handler	
start interrupt timer		
	start timer interrupt CPU in X ms	

OS @ run (kernel mode)	Hardware	Program (user mode)

Context Switch

Call switch() routine
save regs(A) to proc-struct(A)
restore regs(B) from proc-struct(B)
switch to k-stack(B)
return-from-trap (into B)

restore regs(B) from k-stack(B)
move to user mode
jump to B's PC

→ Process B
...

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

OBJECTIVES – 4/11

- Questions from 4/6
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Chapter 6: Limited Direct Execution
- **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
  - MLFQ Scheduler
  - Job Starvation
  - Gaming the Scheduler
  - Examples

CHAPTER 7- SCHEDULING: INTRODUCTION

49

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?

50

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

With n=3 and x_1=.2, x_2=.7, x_3=.1

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

With n=3 and x_1=.33, x_2=.33, x_3=.33

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

April 11, 2023 TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma L5.53

53

OBJECTIVES – 4/11

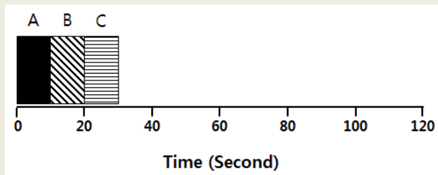
- Questions from 4/6
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Chapter 6: Limited Direct Execution
- Chapter 7: Scheduling Introduction
  - Scheduling metrics
    - Turnaround time, Jain's Fairness Index, Response time
  - FIFO, SJF, STCF, RR schedulers
- Chapter 8: Multi-level Feedback Queue
  - MLFQ Scheduler
  - Job Starvation
  - Gaming the Scheduler
  - Examples

April 11, 2023 TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma L5.54

54

SCHEDULERS

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



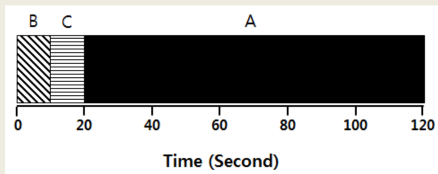
$$Average\ turnaround\ time = \frac{10+20+30}{3} = 20\ sec$$

OBJECTIVES – 4/11

- Questions from 4/6
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Chapter 6: Limited Direct Execution
- 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
  - MLFQ Scheduler
  - Job Starvation
  - Gaming the Scheduler
  - Examples

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

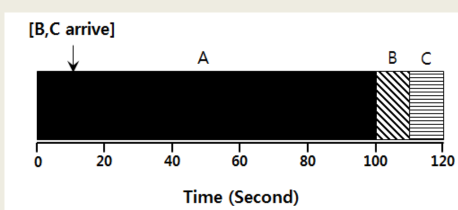


$$Average\ turnaround\ time = \frac{10 + 20 + 120}{3} = 50\ sec$$

57

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



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

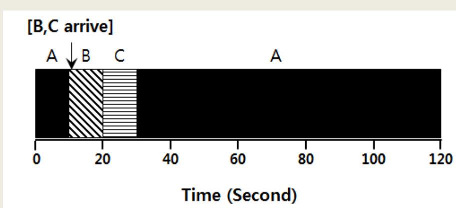
58

OBJECTIVES – 4/11

- Questions from 4/6
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Chapter 6: Limited Direct Execution
- 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
  - MLFQ Scheduler
  - Job Starvation
  - Gaming the Scheduler
  - Examples

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



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

WE WILL RETURN AT ~2:43PM

61

OBJECTIVES – 4/11

- Questions from 4/6
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Chapter 6: Limited Direct Execution
- **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
  - MLFQ Scheduler
  - Job Starvation
  - Gaming the Scheduler
  - Examples

62

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

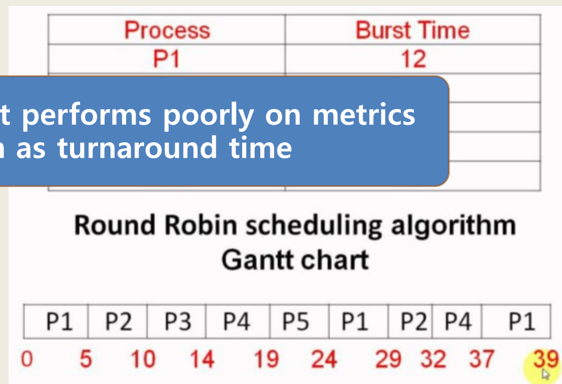
What scheduling algorithm(s) can help minimize response time?

OBJECTIVES – 4/11

- Questions from 4/6
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Chapter 6: Limited Direct Execution
- 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
  - MLFQ Scheduler
  - Job Starvation
  - Gaming the Scheduler
  - Examples

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.



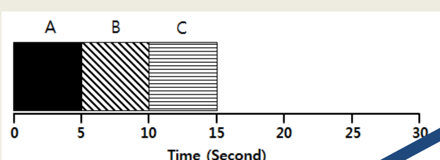
RR is fair, but performs poorly on metrics such as turnaround time

Scheduling Quantum = 5 seconds

65

RR EXAMPLE

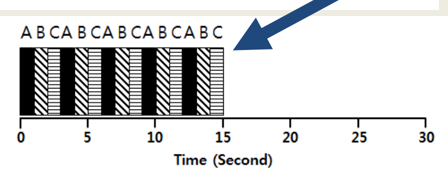
- ABC arrive at time=0, each run for 5 seconds



SJF (Bad for Response Time)

OVERHEAD not considered

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



RR with a time-slice of 1sec (Good for Response Time)

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

66

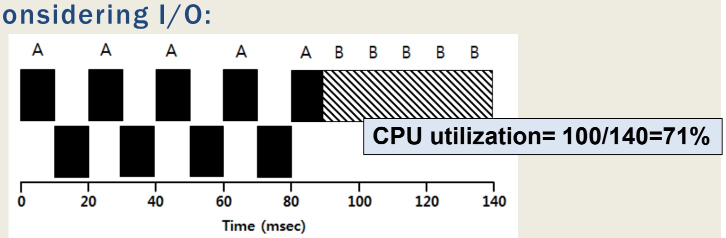
ROUND ROBIN: TRADEOFFS

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

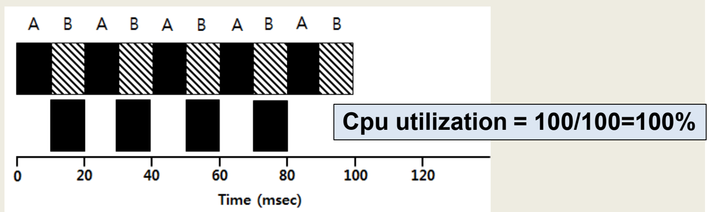
- STCF scheduler
  - A: CPU=50ms, I/O=40ms, 10ms intervals
  - B: CPU=50ms, I/O=0ms
  - Consider A as 10ms subjobs (CPU, then I/O)
- Without considering I/O:



Poor Use of Resources

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)



Overlap Allows Better Use of Resources

April 11, 2023	TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma	L5.69

69

Activities | Visual settings | Edit

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

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

- First In - First Out (FIFO)
- Shortest Job First (SJF)
- Shortest Time to Completion First (STCF)
- Round Robin
- None of the Above
- All of the Above

Total Results: 0

Powered by Poll Everywhere

70

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

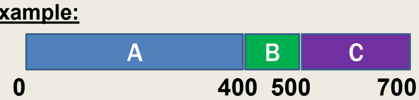
SCHEDULING METRICS

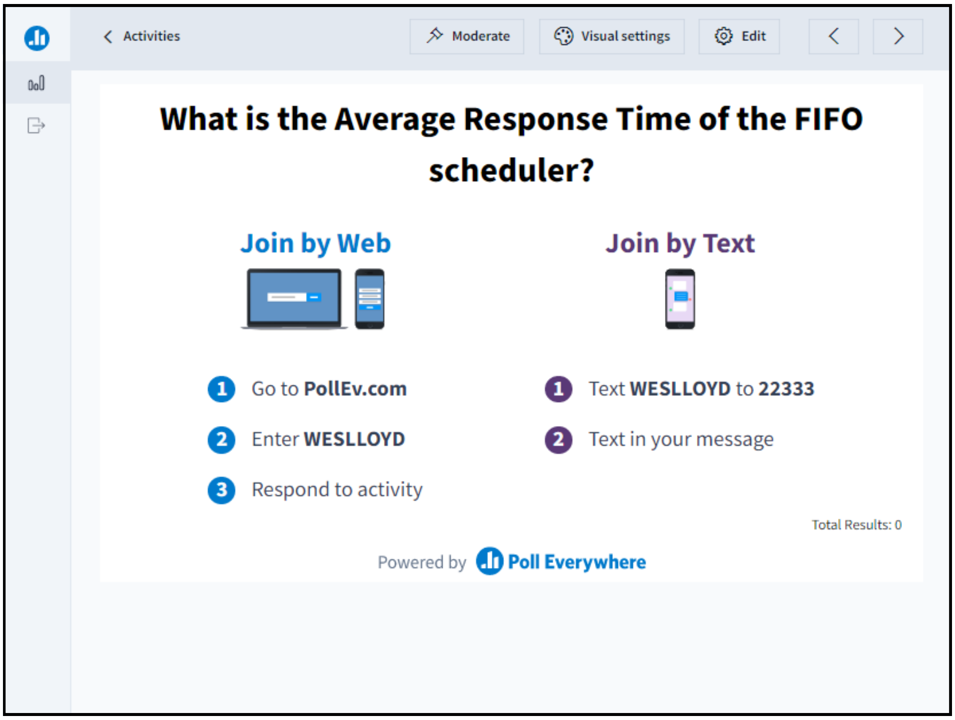
- Consider Three jobs (A, B, C) that require: $time_A$=400ms, $time_B$=100ms, and $time_C$=200ms

- All jobs arrive at time=0 in the sequence of A B C.

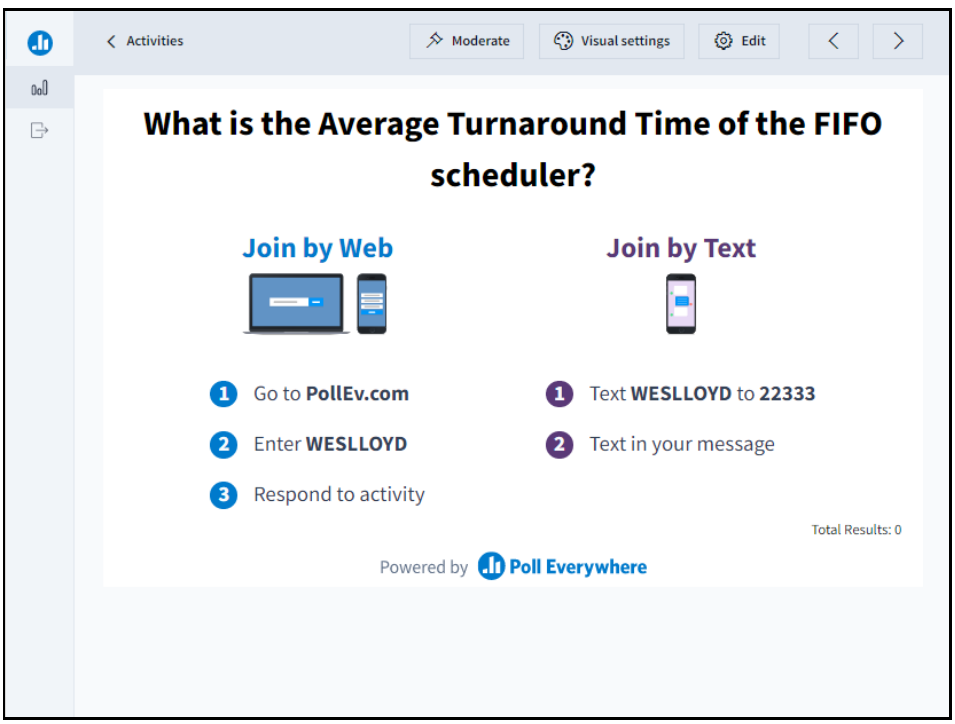
- Draw a scheduling graph to help compute the <u>**average response time (ART)**</u> and <u>**average turnaround time (ATT)**</u> scheduling metrics for the FIFO scheduler.

<u>Example:</u>




Activities
Moderate
Visual settings
Edit
What is the Average Response Time of the FIFO scheduler?
Join by Web
Join by Text
1 Go to PollEv.com
2 Enter WESLLOYD
3 Respond to activity
1 Text WESLLOYD to 22333
2 Text in your message
Total Results: 0
Powered by Poll Everywhere

73

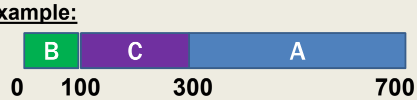

Activities
Moderate
Visual settings
Edit
What is the Average Turnaround Time of the FIFO scheduler?
Join by Web
Join by Text
1 Go to PollEv.com
2 Enter WESLLOYD
3 Respond to activity
1 Text WESLLOYD to 22333
2 Text in your message
Total Results: 0
Powered by Poll Everywhere

74

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:



75

Activities | Moderate | Visual settings | Edit

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

Join by Web

1. Go to **PollEv.com**
2. Enter **WESLLOYD**
3. Respond to activity

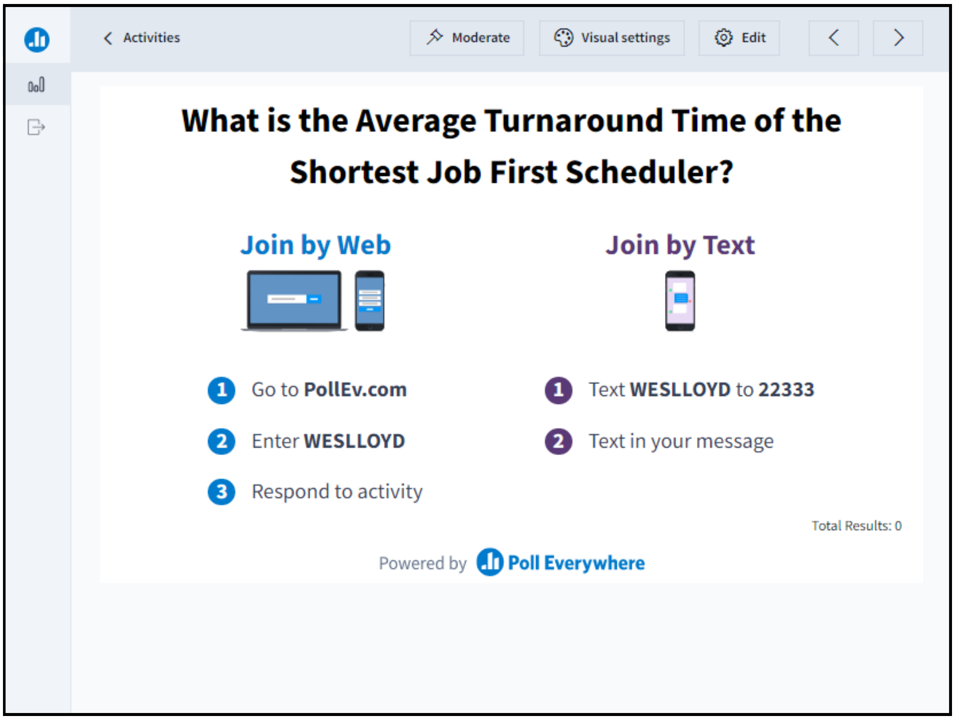
Join by Text

1. Text **WESLLOYD** to **22333**
2. Text in your message

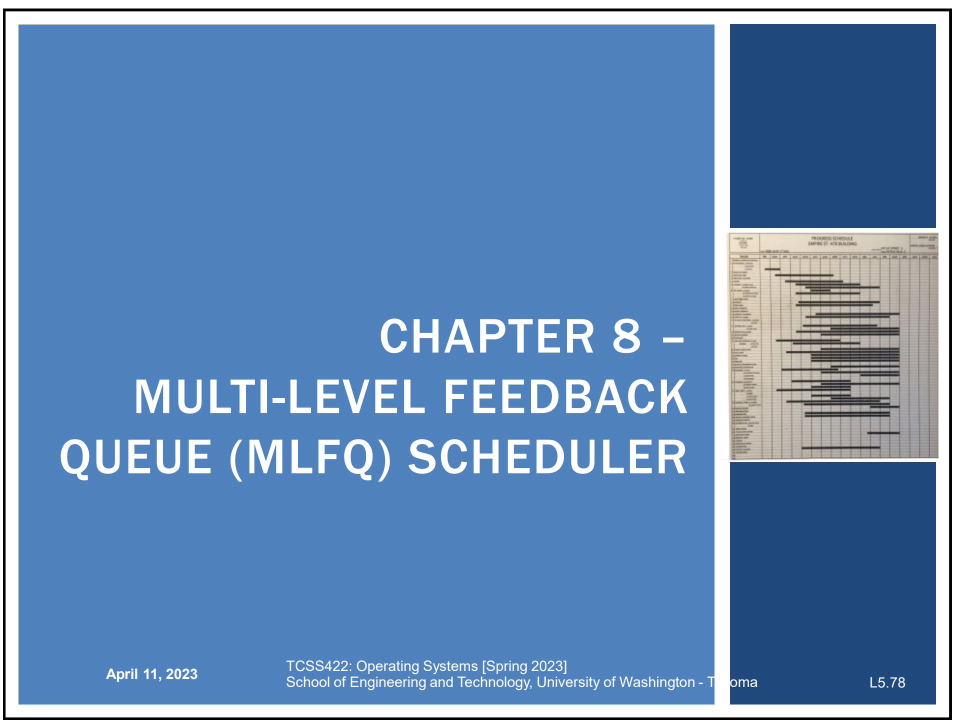
Total Results: 0

Powered by Poll Everywhere

76


Activities
Moderate
Visual settings
Edit
What is the Average Turnaround Time of the Shortest Job First Scheduler?
Join by Web
Join by Text
1 Go to PollEv.com
2 Enter WESLLOYD
3 Respond to activity
1 Text WESLLOYD to 22333
2 Text in your message
Total Results: 0
Powered by Poll Everywhere

77


CHAPTER 8 –
MULTI-LEVEL FEEDBACK
QUEUE (MLFQ) SCHEDULER
April 11, 2023
TCSS422: Operating Systems [Spring 2023]
School of Engineering and Technology, University of Washington - Tacoma
L5.78

78

OBJECTIVES – 4/11

- Questions from 4/6
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Chapter 6: Limited Direct Execution
- 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
  - **MLFQ Scheduler**
  - Job Starvation
  - Gaming the Scheduler
  - Examples

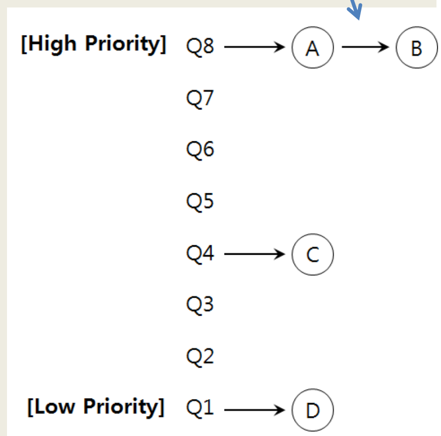
MULTI-LEVEL FEEDBACK QUEUE

- 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



81

MLFQ: DETERMINING JOB PRIORITY

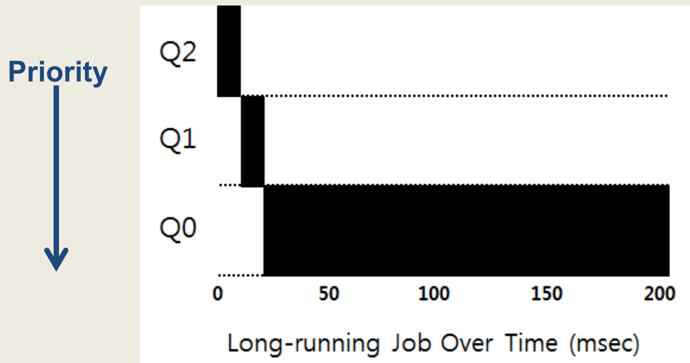
- New arriving jobs are placed into highest priority queue
- If a job uses its entire time slice, priority is reduced (↓)
  - Jobs appears CPU-bound (“batch” job), not interactive (GUI/UI)
- If a job relinquishes the CPU for I/O priority stays the same

MLFQ approximates SJF

82

MLFQ: LONG RUNNING JOB

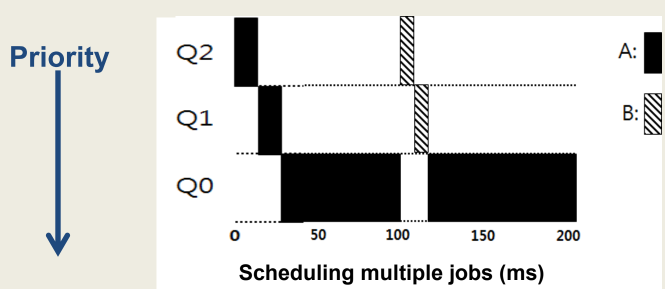
- Three-queue scheduler, time slice=10ms



83

MLFQ: BATCH AND INTERACTIVE JOBS

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

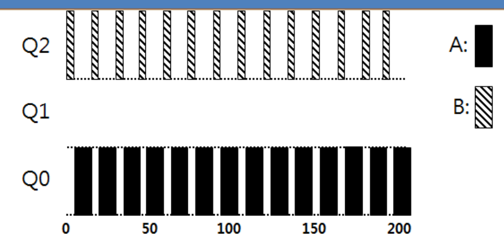


84

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



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

April 11, 2023	TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma	L5.85

85

OBJECTIVES – 4/11

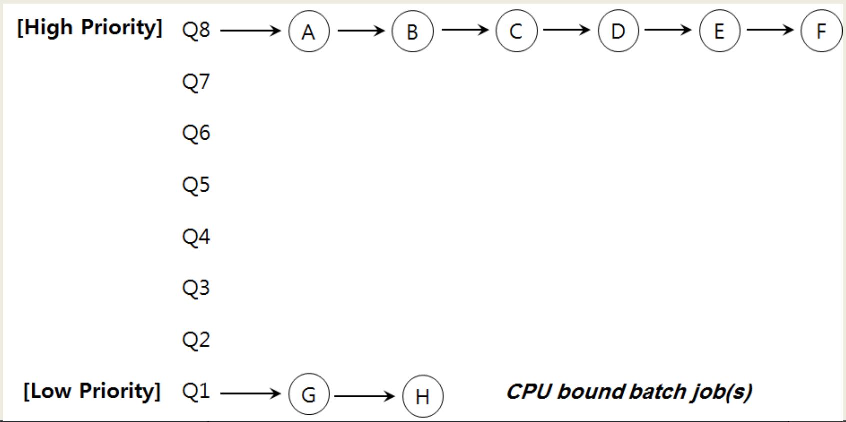
- Questions from 4/6
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Chapter 6: Limited Direct Execution
- Chapter 7: Scheduling Introduction
  - Scheduling metrics
    - Turnaround time, Jain's Fairness Index, Response time
  - FIFO, SJF, STCF, RR schedulers
- Chapter 8: Multi-level Feedback Queue
  - MLFQ Scheduler
  - **Job Starvation**
  - Gaming the Scheduler
  - Examples

April 11, 2023	TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma	L5.86

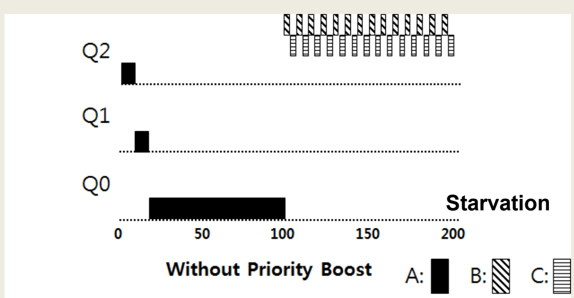
86

MLFQ: ISSUES

- Starvation



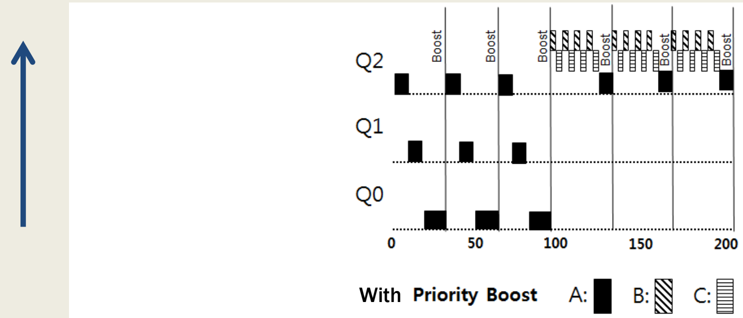
RESPONDING TO BEHAVIOR CHANGE



- 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.

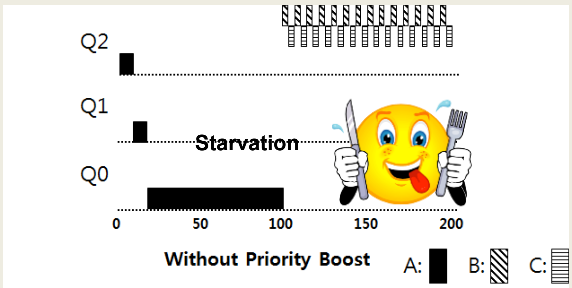
- **KEY**: If time quantum of a higher queue is filled, then we don't run any jobs in lower priority queues!!!

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

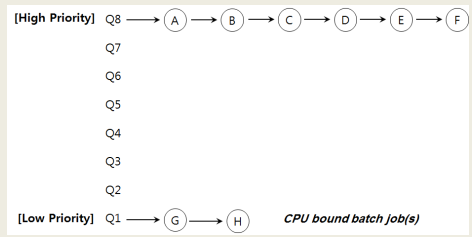


OBJECTIVES – 4/11

- Questions from 4/6
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Chapter 6: Limited Direct Execution
- 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
  - MLFQ Scheduler
  - Job Starvation
  - **Gaming the Scheduler**
  - Examples

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

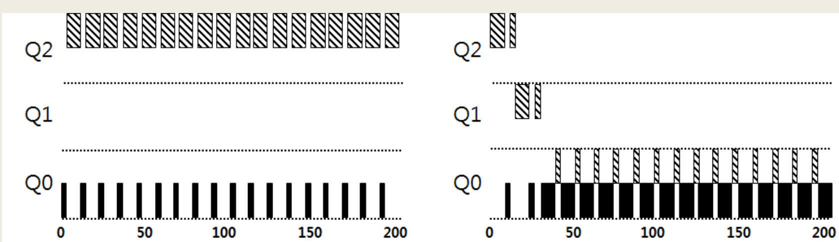


Priority becomes stuck

93

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

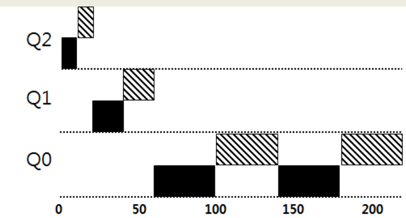


Without(Left) and With(Right) Gaming Tolerance

94

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

April 11, 2023 | TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma | L5.95

95

OBJECTIVES – 4/11

- Questions from 4/6
- Assignment 0
- C Tutorial - Pointers, Strings, Exec in C
- Chapter 6: Limited Direct Execution
- 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
  - MLFQ Scheduler
  - Job Starvation
  - Gaming the Scheduler
  - **Examples**

April 11, 2023 | TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma | L5.96

96

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

99

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***

April 11, 2023	TCSS422: Operating Systems [Spring 2023] School of Engineering and Technology, University of Washington - Tacoma	L5.100

100

QUESTIONS

101