TCSS 422: OPERATING SYSTEMS

INTRODUCTION TO OPERATING SYSTEMS, PROCESSES

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

April 9, 2020 — TCSS422: Operating Systems [Spring 2020]
School of Engineering and Technology, University of Washington - Tacoma

1

OBJECTIVES – 4/9

- Questions from 4/7
- Assignment 0
- 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
- Chapter 7: Scheduling Introduction
  - Scheduling metrics
  - SJF, STCF, RR schedulers

April 9, 2020 — TCSS422: Operating Systems [Spring 2020]
School of Engineering and Technology, University of Washington - Tacoma — L4.2

2

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (53 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- Average – 7.51 (↑ *from* 7.03)

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- Average – 5.82 (↑ from 5.76)

3

FEEDBACK FROM 4/7

- ***REVIEW: What were the key takeaways from Tuesday's lecture?***

- Ch. 4: Kernel data structures: What are they? Where are they? How do you find them?

- Ch. 5: Process APIs: fork(), wait(), and exec()

- Ch. 6:
  Direct Execution, Operating system control trade-off

4

FEEDBACK - 2

- ***Can you quickly summarize what is Exec() is, and when it should be used?***
  - Exec() is used to direct the currently running process to execute another program on the file-system
- Think of this like a "hand-off"
- When the other program concludes, the process does not return, but exits
- The original process passes its three file systems to the new executable:

C constants:

- `stdin` (input stream, reads from the keyboard)
- `stdout` (output stream, writes to the screen)
- `stderr` (output stream, typically writes to the screen)

April 9, 2020	TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma	L4.5

5

FEEDBACK - 3

- ***I was confused about when you discussed booting up Linux and forking. When Linux is booted up, it becomes the root process with pid = 1. I'm not sure what happens when we try to create a new process.***
- The Linux kernel boots as PID 1
- Every subsequent process is a child "forked" from PID 1

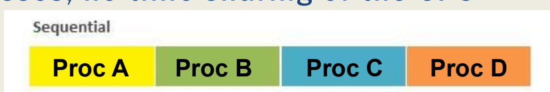
```
wlloyd@dione: ~
wlloyd@dione:~$ ps aux | head -n 10
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0 185492  4532 ?        Ss    2019   2:05 /lib/systemd/systemd --system --deserialize 22
root         2  0.0  0.0      0     0 ?        S     2019   0:04 [kthreadd]
root         3  0.0  0.0      0     0 ?        S     2019   1:08 [ksoftirqd/0]
root         5  0.0  0.0      0     0 ?        S<    2019   0:00 [kworker/0:0H]
root         7  0.0  0.0      0     0 ?        S     2019  61:20 [rcu_sched]
root         8  0.0  0.0      0     0 ?        S     2019   0:00 [rcu_bh]
root         9  0.0  0.0      0     0 ?        S     2019   0:24 [migration/0]
root        10  0.0  0.0      0     0 ?        S     2019   0:13 [watchdog/0]
root        11  0.0  0.0      0     0 ?        S     2019   0:15 [watchdog/1]
wlloyd@dione:~$
```

April 9, 2020	TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma	L4.6

6

FEEDBACK - 4

- ***The points that remain least clear to me is time sharing.***
- Time sharing is where multiple programs share system resources (e.g. CPU, memory, DISK, network) at the same time
- 4 Processes, no time sharing of the CPU



- 4 Processes, with time sharing of the CPU

April 9, 2020	TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma	L4.7

7

- ***In the Ubuntu VM, is it true that processes and threads are equivalent?***
- In Linux, threads and processes both receive process IDs (PIDs).
- Threads also receive a TGID (thread group ID) which is the PID of the parent process.
- The parent process will have a TGID equal to it's own PID.
- Here's how the numbering might work for a group of 3 processes:

```
process 1          thread 1          thread 2
---------          --------          --------
pid=123     --->   pid=124    --->   pid=125
tgid=123           tgid=123          tgid=123

process 2
---------
pid=126
tgid=126

process 3          thread 1          thread 2
---------          --------          --------
pid=127     --->   pid=128    --->   pid=129
tgid=127           tgid=127          tgid=127
```

L4.8

8

- ***In the Ubuntu VM, is it true that processes and threads are equivalent?***
- In Linux, threads and processes both receive process IDs (PIDs).
- Threads also receive a TGID (thread group ID) which is the PID of the parent process.
- The parent process will have a TGID equal to it's own PID.
- Here's how the numbering might work for a group of 3 processes:

All PIDs (processes & threads) share the same sequence of numbers from 1 to 32768.

When PID 32768 is created, the numbering wraps around.

```
pid=126
tgid=126

process 3          thread 1          thread 2
---------          --------          --------
pid=127     --->   pid=128    --->   pid=129
tgid=127           tgid=127          tgid=127
```

L4.9

9

FEEDBACK - 7

- ***In the Ubuntu VM, is it true that processes and threads are equivalent?***
- Threads share with the parent process:
  the data segment (global memory)
  the heap segment (used for malloc)
  the code segment
- When a new thread is created, they only need to allocate memory for their own **stack segment**.
- Creating **threads** is seen as **faster** than **processes** because far ***less memory*** needs to be allocated.
- When creating a process, it is necessary to allocate new memory for the data, heap, code, and stack segments.
- All threads will have a parent process identified by the TGID.

April 9, 2020	TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma	L4.10

10

OBJECTIVES – 4/9

- Questions from 4/7
- Assignment 0
- 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
- Chapter 7: Scheduling Introduction
  - Scheduling metrics
  - SJF, STCF, RR schedulers

11

"QUIZ" 0 – C PROGRAMMING BACKGROUND SURVEY

- Available via Canvas System
- Under:
  Assignments → Tutorials/Quizzes/In-class Activities
- Please disregard grade assigned by Canvas
- All submissions will receive 10 pts after assignment closes - (*closes Thursday 4/9 @ 11:59p*)

12

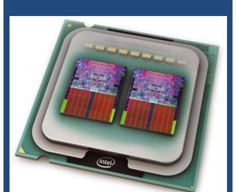
VIRTUAL MACHINE SURVEY

- Virtual Machine Survey
- Request for Ubuntu 18.04 VMs has been sent to the School of Engineering and Technology LABS
- Expect response soon regarding connection information
- Thank you!

April 9, 2020	TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma	L4.13

13

CH. 6: LIMITED DIRECT EXECUTION



April 9, 2020 TCSS422: Operating Systems [Spring 2020]
School of Engineering and Technology, University of Washington - Tacoma L4.14

14

OBJECTIVES – 4/9

- Questions from 4/7
- Assignment 0
- 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
- Chapter 7: Scheduling Introduction
  - Scheduling metrics
  - SJF, STCF, RR schedulers

15

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

16

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

17

CPU MODES

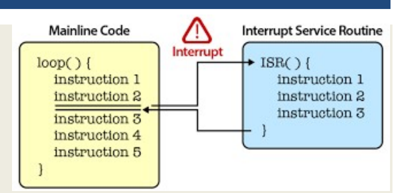
- <u>User mode: ring 3 - untrusted</u>
  - Some instructions and registers are disabled by the CPU
  - Exception registers
  - HALT instruction
  - MMU instructions
  - OS memory access
  - I/O device access
- <u>Kernel mode: ring 0 – trusted</u>
  - All instructions and registers enabled

18

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

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

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

April 9, 2020 | TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma | L4.21

21

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 9, 2020 | TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma | L4.22

22

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		
	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		

Computer BOOT Sequence: OS with Limited Direct Execution

April 9, 2020 | TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma | L4.23

23

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?

April 9, 2020 | TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma | L4.24

24

MULTITASKING

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

- Cooperative multitasking (mostly pre 32-bit)
  - < W
  - Op l
    - I
    - 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 9, 2020	TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma	L4.25

25

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

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

Total Results

26

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?

29

W

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?

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

Total Results

30

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?

31

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.

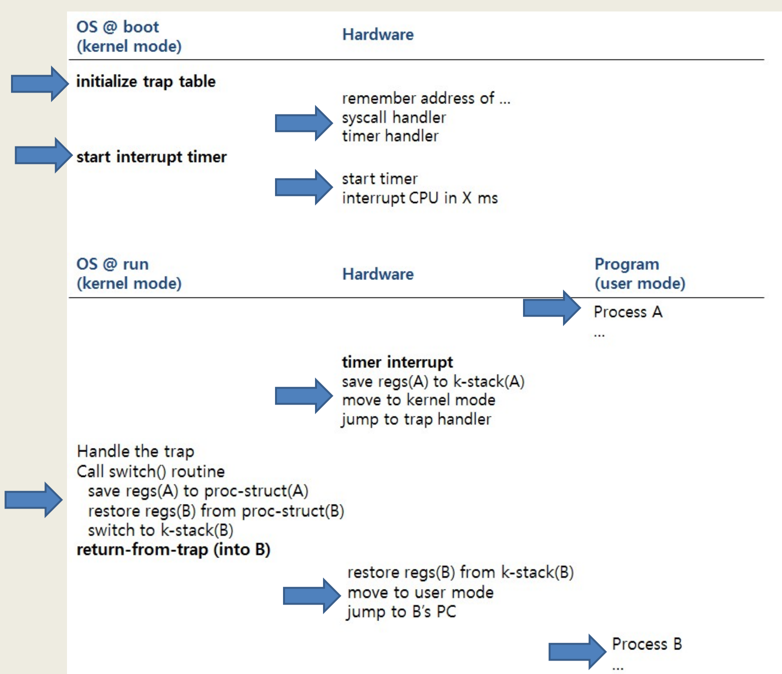
32

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

33



34

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 …

35

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

36

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

37

TCSS 422 WILL RETURN AT 2:40PM

38

CHAPTER 7- SCHEDULING: INTRODUCTION

39

OBJECTIVES – 4/9

- Questions from 4/7
- Assignment 0
- 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
- Chapter 7: Scheduling Introduction
  - Scheduling metrics
  - SJF, STCF, RR schedulers

40

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

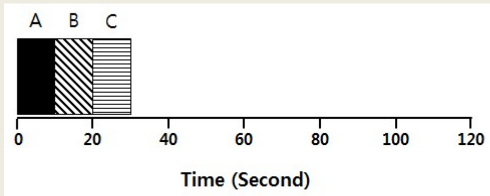
$$\mathcal{J}(x_1, x_2, \ldots, x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \cdot \sum_{i=1}^{n} {x_i}^2}$$

- 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

SCHEDULERS

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



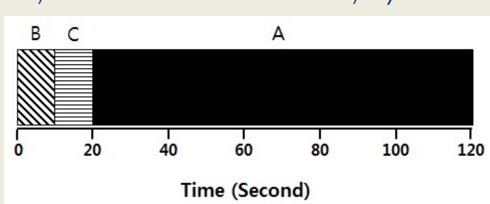
$$Average\ turnaround\ time = \frac{10+20+30}{3} = 20\ sec$$

April 9, 2020	TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma	L4.43

43

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



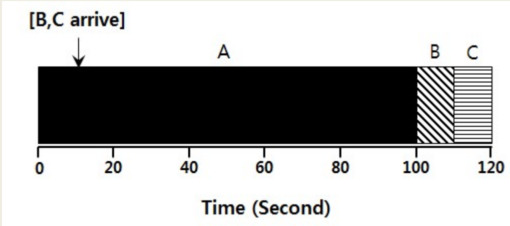
$$Average\ turnaround\ time = \frac{10+20+120}{3} = 50\ sec$$

April 9, 2020	TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma	L4.44

44

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

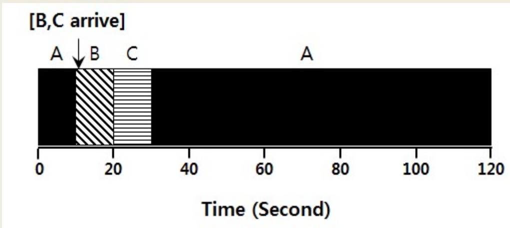


$$Average\ turnaround\ time = \frac{100 + (110 - 10) + (120 - 10)}{3} = 103.33\ sec$$

45

STCF - 2

- 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



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

46

SCHEDULING METRICS - 3

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

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

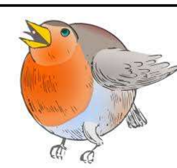
- STCF, SJF, FIFO
  - can perform poorly with respect to response time

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

April 9, 2020 | TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma | L4.47

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

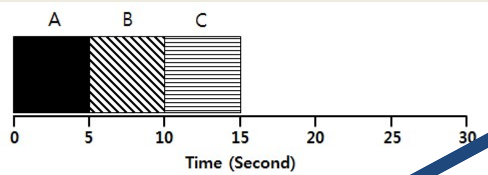
April 9, 2020 | TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma | L4.48

48

RR EXAMPLE

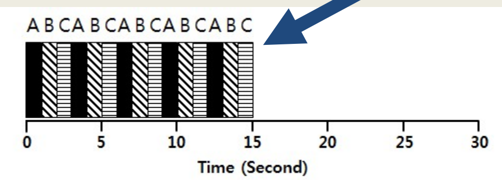
- ABC arrive at time=0, each run for 5 seconds

OVERHEAD not considered



SJF (Bad for Response Time)

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



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

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

49

ROUND ROBIN: TRADEOFFS

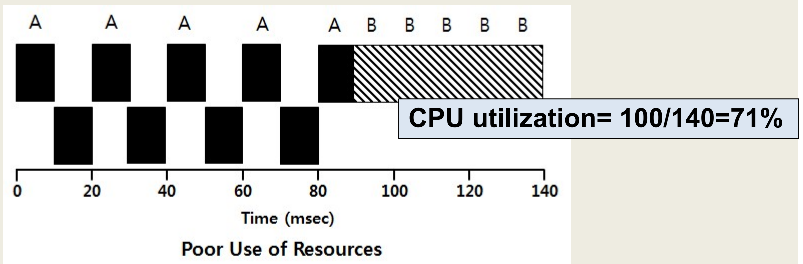
Short Time Slice		Long Time Slice
Fast Response Time	⟷	Slow Response Time
High overhead from context switching		Low overhead from context switching

- 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

50

SCHEDULING WITH I/O

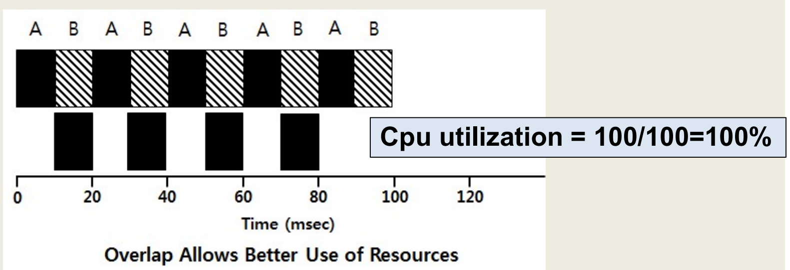
- STCF scheduler
  - A: CPU=50ms, I/O=40ms, 10ms intervals
  - B: CPU=50ms, I/O=0ms
  - Consider A as 10ms subjobs (CPU, then I/O)
- Without considering I/O:



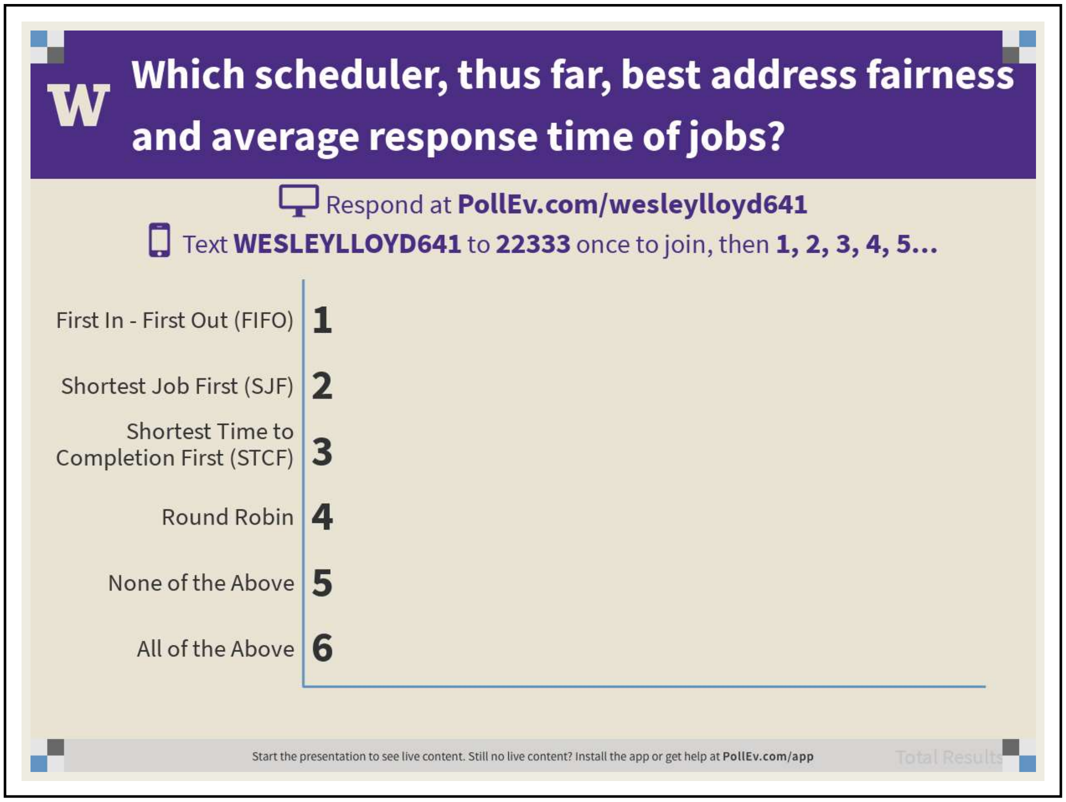
51

SCHEDULING WITH I/O - 2

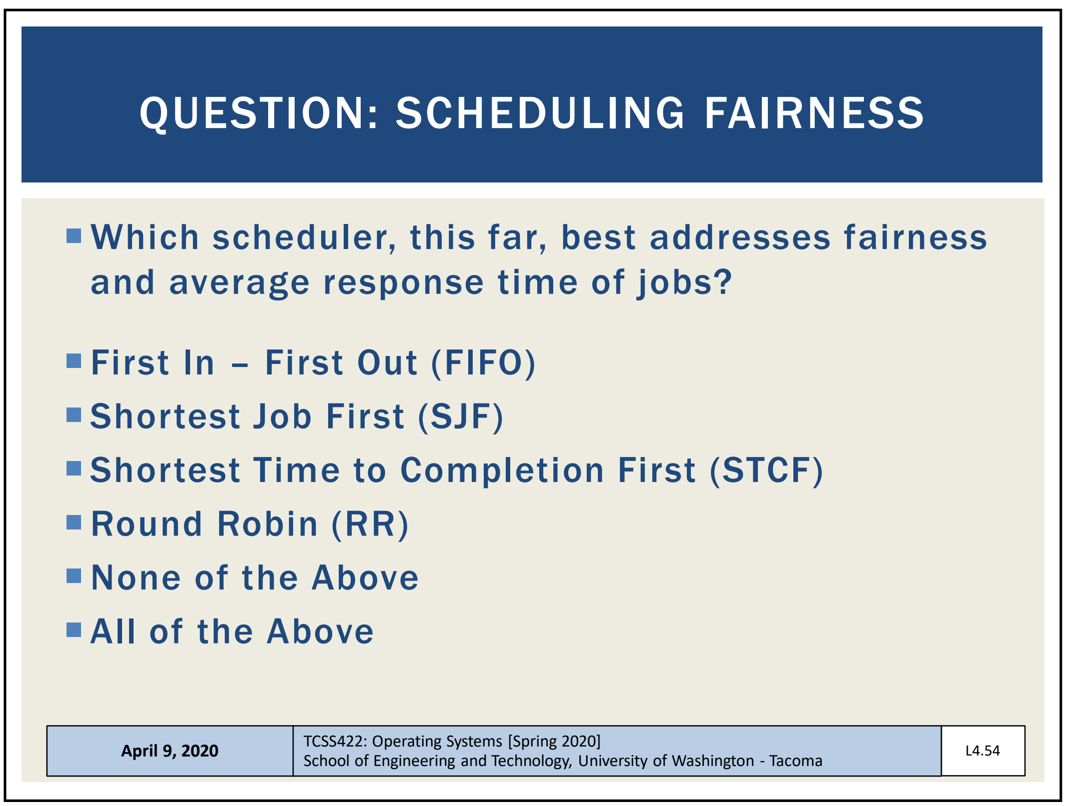
- When a job initiates an I/O request
  - A is blocked, waits for I/O to compute, frees CPU
  - STCF scheduler assigns B to CPU
- When I/O completes → raise interrupt
  - Unblock A, STCF goes back to executing A: (10ms sub-job)



52


W
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
Total Results

53

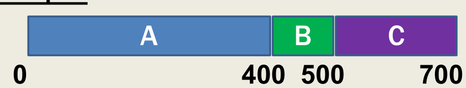

QUESTION: SCHEDULING FAIRNESS
Which scheduler, this far, best addresses fairness and average response time of jobs?
First In – First Out (FIFO)
Shortest Job First (SJF)
Shortest Time to Completion First (STCF)
Round Robin (RR)
None of the Above
All of the Above
April 9, 2020
TCSS422: Operating Systems [Spring 2020]
School of Engineering and Technology, University of Washington - Tacoma
L4.54

54

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.

Example:



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**

56

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?

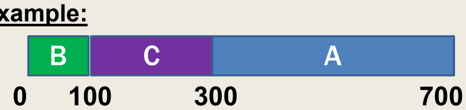
Start the presentation to see live content. Still no live content? Install the app or get help at **PollEv.com/app**

57

SCHEDULING METRICS

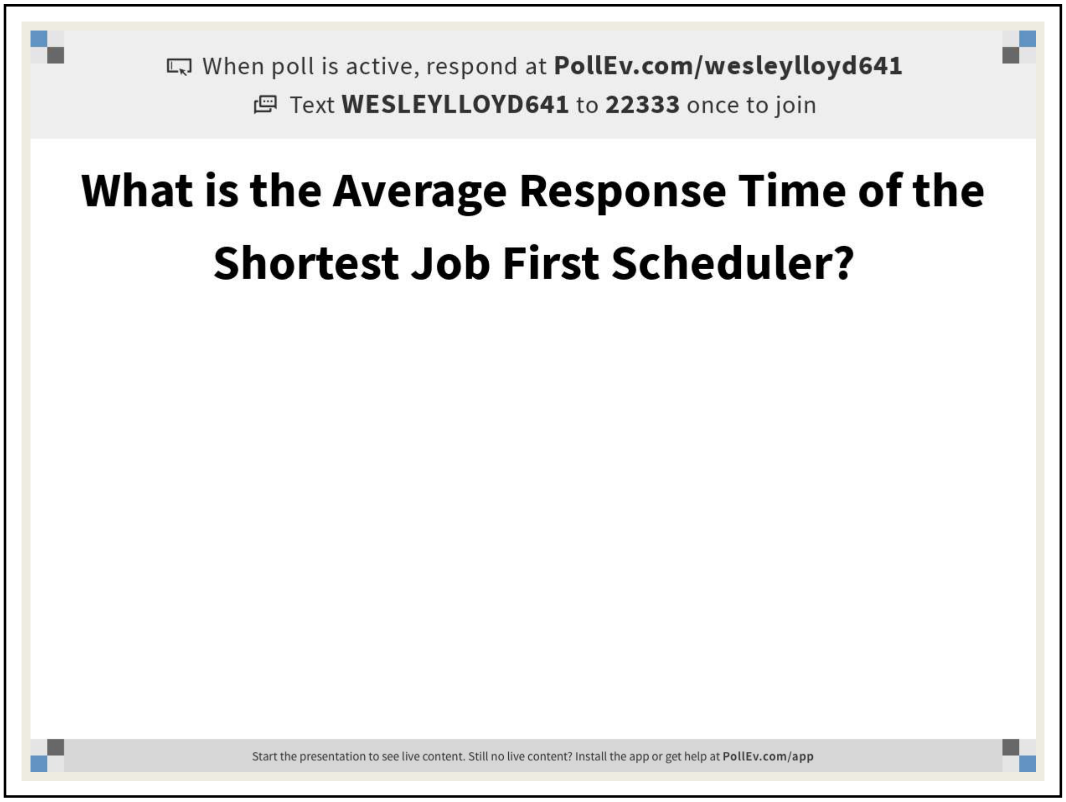
- Consider Three jobs (A, B, C) that require: $time_A$=400ms, $time_B$=100ms, and $time_C$=200ms
- All jobs arrive at time=0 in the sequence of A B C.
- Draw a scheduling graph to help compute the <u>average response time (ART)</u> and <u>average turnaround time (ATT)</u> scheduling metrics for the SJF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

<u>Example:</u>

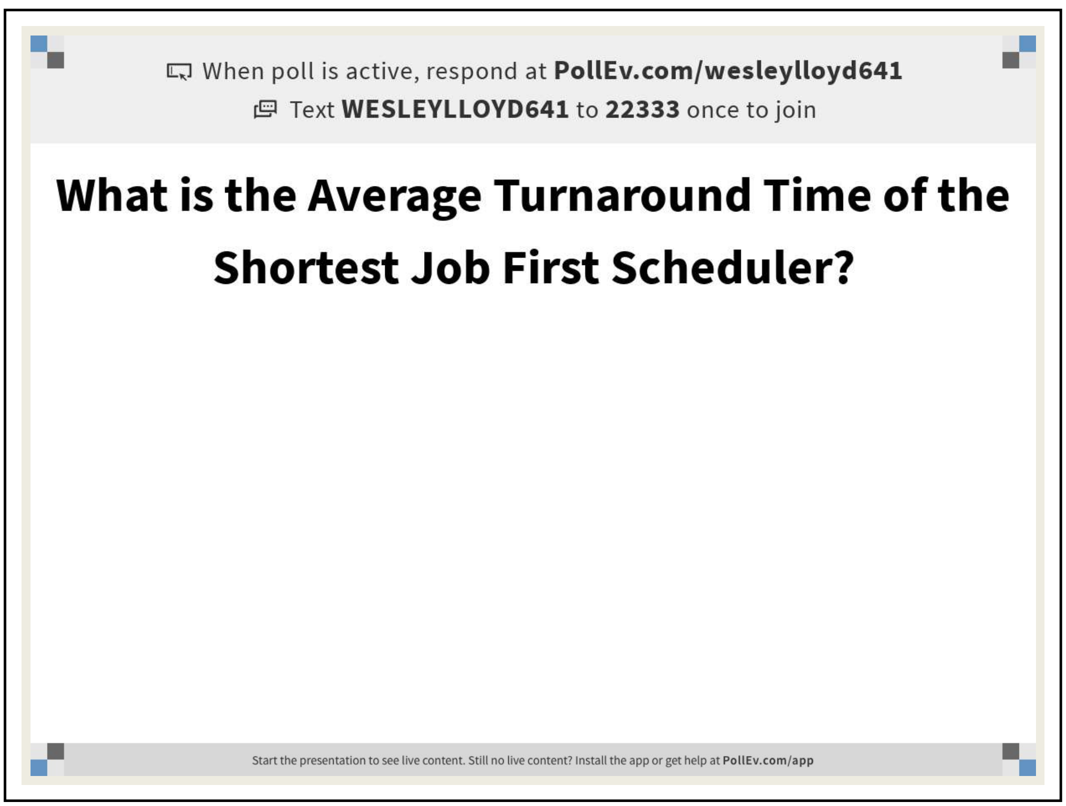


April 9, 2020	TCSS422: Operating Systems [Spring 2020] School of Engineering and Technology, University of Washington - Tacoma	L4.58

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**

59


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**

60

QUESTIONS

61

WILL RETURN IN A FEW MINUTES

62