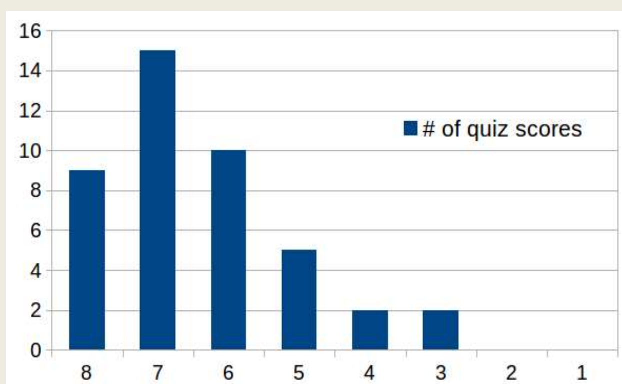
TCSS 422: OPERATING SYSTEMS

Processes, Process API, Limited Direct Execution

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

QUIZ 0 SCORES

- Average – 6.419
- Min – 3
- Max/4th quartile – 8
- 2nd & 3rd Quartile– 7
- 1st quartile – 6
- Mode – 7
- Std. Dev. – 1.33



FEEDBACK FROM 10/3

- Real world example of "a child of a parent of a process"
- … when do processes have children?
- Check process ID of BASH shell:
  - echo $$
- Check parent's process ID:
  - echo $PPID
- Exec launches a different process or program
  - What is the difference between a process and a program?
  - Exec does not create a new process. It transfers control: **Man page:** "The exec() family of functions replaces the current process image with a new process image."

FEEDBACK - 2

- Can you create more than 1 fork?
  - i.e. call fork() more than one time in a program
- If you create more than one fork(), how do you handle them?
- How would you use fork in a potential application?
- Code examples online under "Schedule" tab:

Source Code Examples

Source code for examples from class are posted **[HERE]**.

FEEDBACK - 3

- Most of the Linux calls are still unclear
- Is it possible to record the lectures?

OBJECTIVES

- C Tutorial
- Quiz 1 – Active Reading
- Chapter 6 – Limited Direct Execution – cont'd
- Chapter 7 – Introduction to Scheduling
- Chapter 8 – Multi-level Feedback Queue

CH. 6: LIMITED DIRECT EXECUTION

CHAPTER 6 REVIEW

- As per Chapter 6, What is DIRECT Execution?
- What is Limited Direct Execution?
- What is a context switch?
- What is a system call?
- What is an operating system “Trap”?
- What is the difference between a maskable and a non-maskable interrupt?

DIRECT EXECUTION - 2

- **With direct execution:**

  How does the OS stop a program from running, and switch to another to support **time sharing**?

  How do programs share disks and perform I/O if they are given direct control? Do they know about each other?

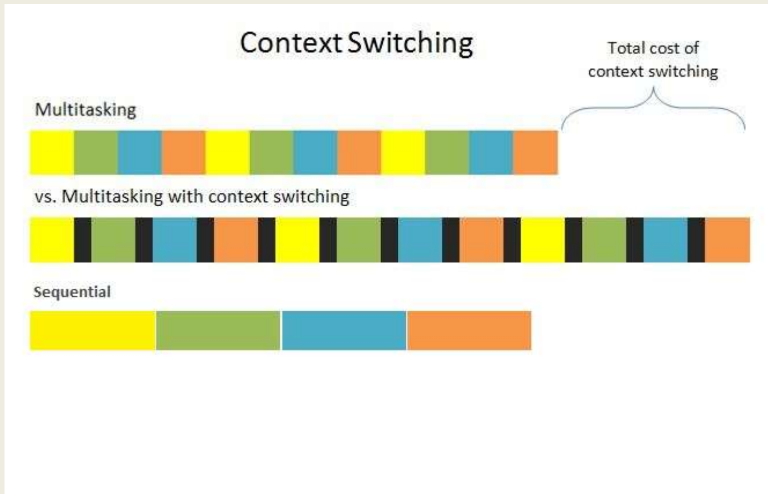
  With direct execution, how can dynamic memory structures such as linked lists grow over time?

CONTROL TRADEOFF

- **Too little control:**
  - No security
  - No time sharing

- **Too much control:**
  - Too much OS overhead
  - Poor performance for compute & I/O
  - Complex APIs (system calls), difficult to use

CONTEXT SWITCHING OVERHEAD



LIMITED DIRECT EXECUTION

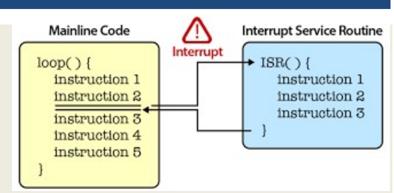
- OS implements LDE to support time/resource sharing
- Limited direct execution means "only limited" processes can execute DIRECTLY on the CPU in ***trusted*** mode
- TRUSTED means the process is trusted, and it can do anything… (e.g. it is a system / kernel level process)
- Enabled by ***protected (safe)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

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

October 8, 2018	TCSS422: Operating Systems [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma	L4.13

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

October 8, 2018	TCSS422: Operating Systems [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma	L4.14

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?

October 8, 2018	TCSS422: Operating Systems [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma	L4.15

MULTITASKING

- How/when should the OS regain control of the CPU to switch between processes?
- Cooperative multitasking (mostly pre 32-bit)
  - < W
  - 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**

October 8, 2018	TCSS422: Operating Systems [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma	L4.16

W 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

QUESTION: MULTITASKING

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

October 8, 2018	TCSS422: Operating Systems [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma	L4.18

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?

October 8, 2018 | TCSS422: Operating Systems [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma | L4.19

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)

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

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

October 8, 2018 | TCSS422: Operating Systems [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma | L4.20

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

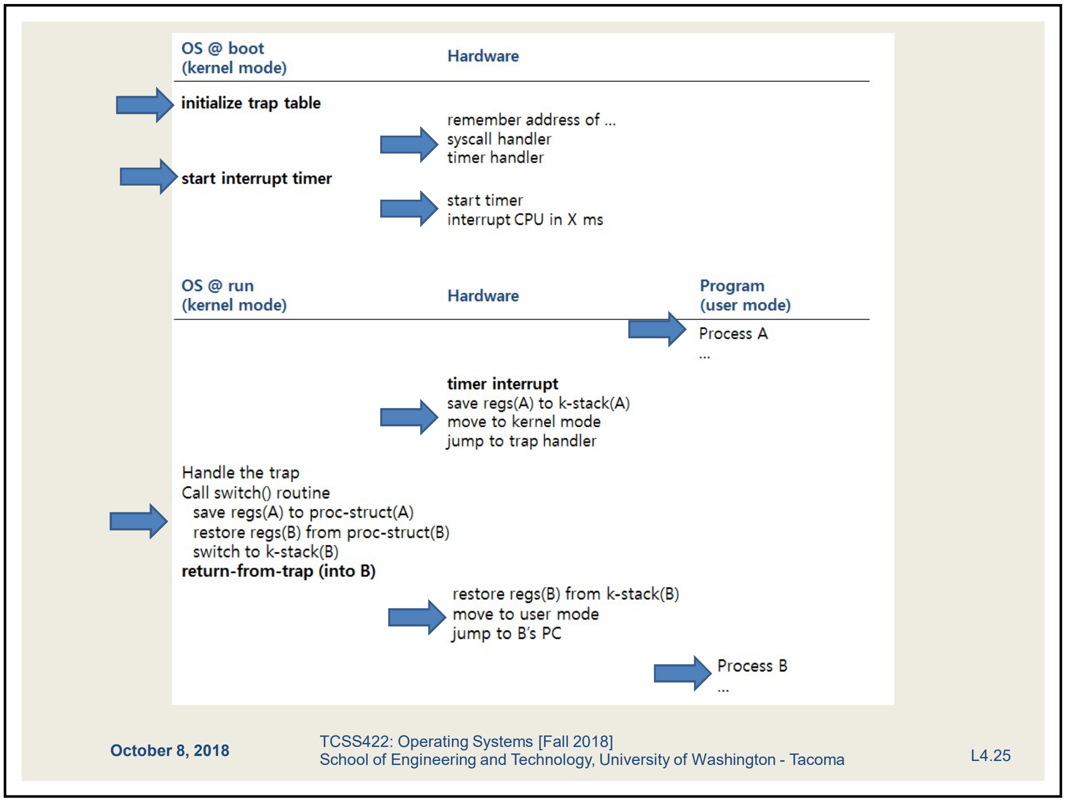
- For an OS that uses a system timer to force arbitrary context switches to share the CPU, what is a good value (in seconds) for the timer interrupt?

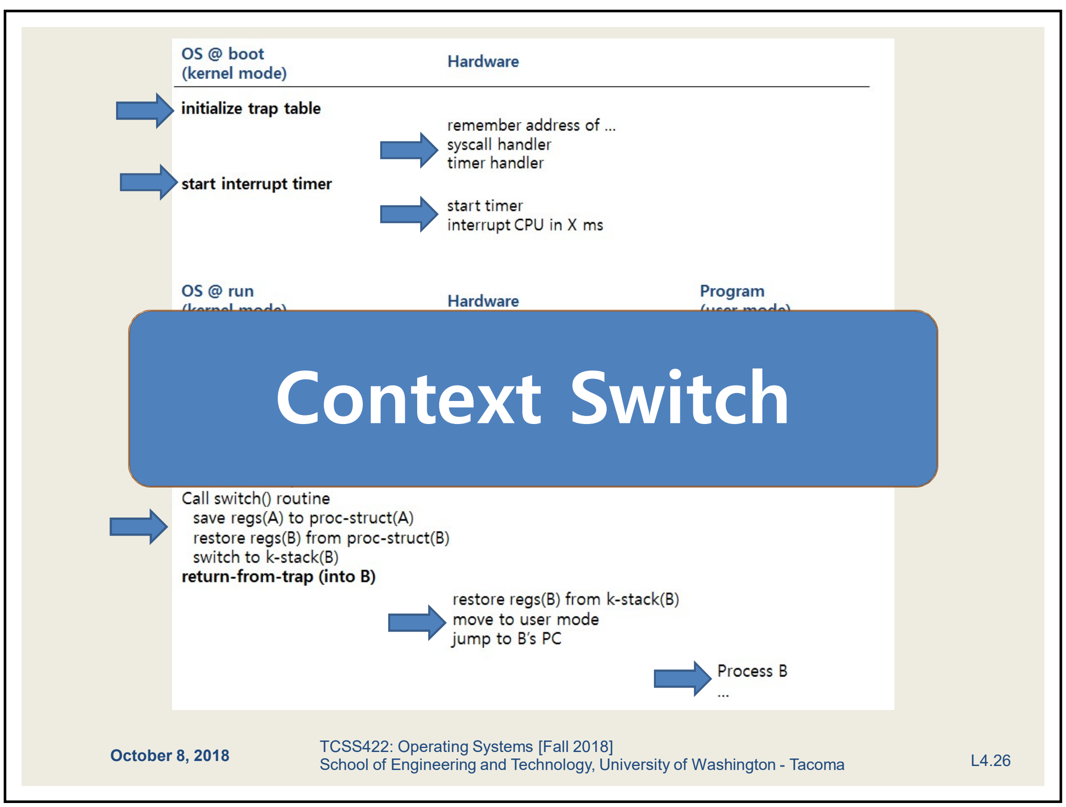
CONTEXT SWITCH

- 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)
Process A
…
timer interrupt
save regs(A) to k-stack(A)
move to kernel mode
jump to trap handler
Handle the trap
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
…
October 8, 2018
TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma
L4.25


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
Hardware
Program
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
…
October 8, 2018
TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma
L4.26

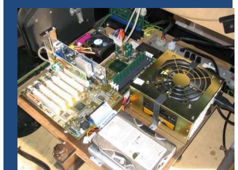
INTERRUPTED INTERRUPTS

- 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

CHAPTER 7- SCHEDULING: INTRODUCTION



October 8, 2018
TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma
L4.29

SCHEDULING INTRODUCTION

- For simplicity, consider job scheduling with limitations:
  - Each job requires the same CPU time
  - All jobs arrive at the same time
  - All jobs only use the CPU (no I/O)
  - The run-time of each job is known a priori

October 8, 2018
TCSS422: Operating Systems [Fall 2018]
School of Engineering and Technology, University of Washington - Tacoma
L4.30

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

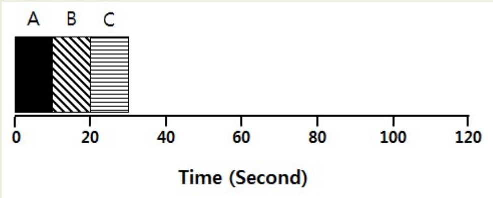
$$\mathcal{J}(x_1, x_2, \ldots, x_n) = \frac{(\sum_{i=1}^{n} x_i)^2}{n \cdot \sum_{i=1}^{n} {x_i}^2}$$

- 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

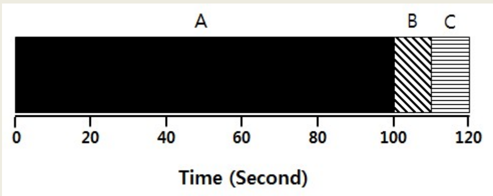
- FIFO: first in, first out
  - Very simple, easy to implement
- Consider
  - 3 x 10sec jobs, arrival: A B C



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

FIFO: CONVOY EFFECT

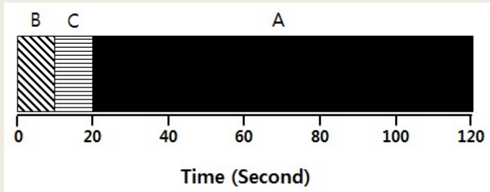
- FIFO with different jobs lengths
- Consider
  - A_{len}=100sec, B_{len}=10sec, C_{len}=10sec



$$Average\ turnaround\ time = \frac{100 + 110 + 120}{3} = 110\ sec$$

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

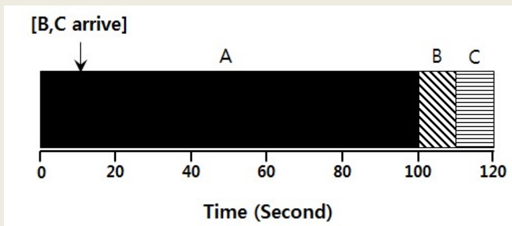


$$Average\ turnaround\ time = \frac{10 + 20 + 120}{3} = 50\ sec$$

October 8, 2018	TCSS422: Operating Systems [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma	L4.35

SJF: WITH RANDOM ARRIVAL

- If jobs arrive at any time:
- A @ t=0sec, B @ t=10sec, C @ t=10sec



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

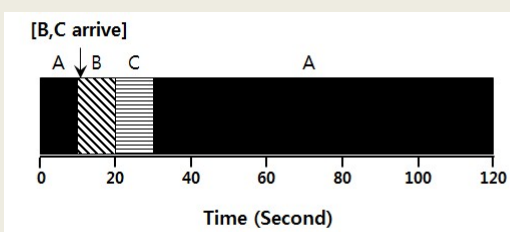
October 8, 2018	TCSS422: Operating Systems [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma	L4.36

STCF – SHORTEST TIME TO COMPLETION FIRST

- Add preemption to the Shortest Job First scheduler
  - Also called preemptive shortest job first (PSJF)
- When a new job enters the system:
  - Of all jobs, *Which has the least time left?*
  - PREMPT job execution, and schedule the **new** shortest job
- More realistic, but how do we know execution time in advance?
  - Oracle: All knowing one
  - Only schedule static (fixed size) batch workloads
  - Can we predict execution time?

STCF - 2

- Consider:
  - A_{len}=100 $A_{arrival}$=0
  - B_{len}=10, $B_{arrival}$=10, C_{len}=10, $C_{arrival}$=10



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

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?

October 8, 2018	TCSS422: Operating Systems [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma	L4.39

Which scheduler metric does the Shortest Time to Completion First (STCF) scheduler provide the best improvement on vs. First In First Out (FIFO)?

1. average turnaround time of jobs
2. fairness of job scheduling (Jain's fairness index)
3. average response time of jobs
4. All of the above
5. None of the above

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

Total Results

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 timer interrupt period.

Process	Burst Time
P1	12
P5	5

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

Round Robin scheduling algorithm Gantt chart

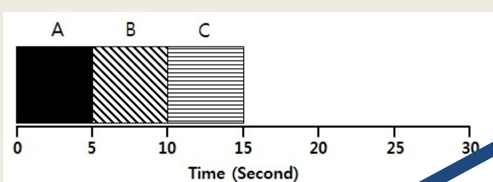
P1	P2	P3	P4	P5	P1	P2	P4	P1

0 5 10 14 19 24 29 32 37 39

Scheduling Quantum = 5 seconds

RR EXAMPLE

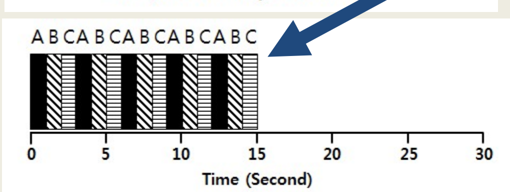
- ABC arrive at time=0, each run for 5 seconds



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)

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

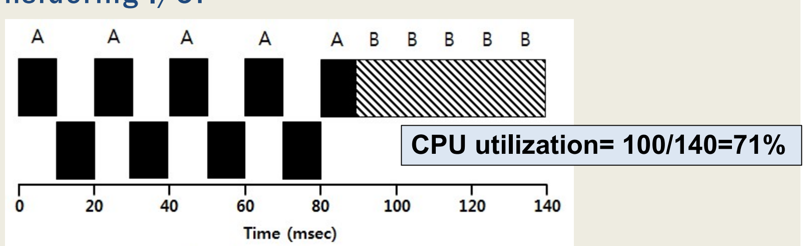
ROUND ROBIN: TRADEOFFS

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

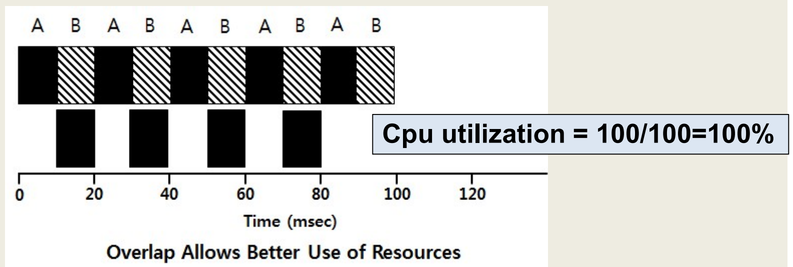
- STCF scheduler
  - A: CPU=50ms, I/O=40ms, 10ms intervals
  - B: CPU=50ms, I/O=0ms
  - Consider A as 10ms subjobs (CPU, then I/O)
- Without considering I/O:



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)

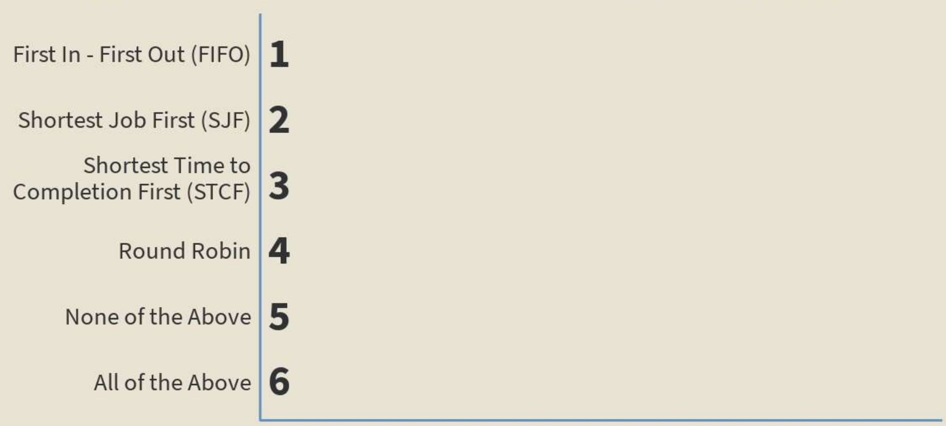


October 8, 2018	TCSS422: Operating Systems [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma	L4.45

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

Respond at **PollEv.com/wesleylloyd641**

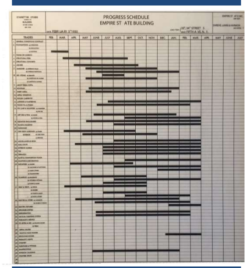
Text **WESLEYLLOYD641** to **22333** once to join, then **1, 2, 3, 4, 5…**



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

Total Results

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



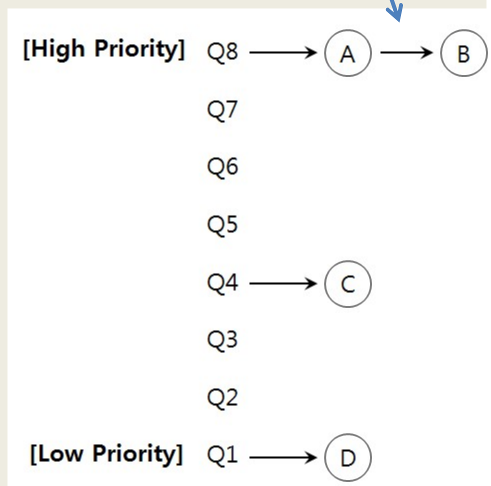
MULTI-LEVEL FEEDBACK QUEUE

- 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

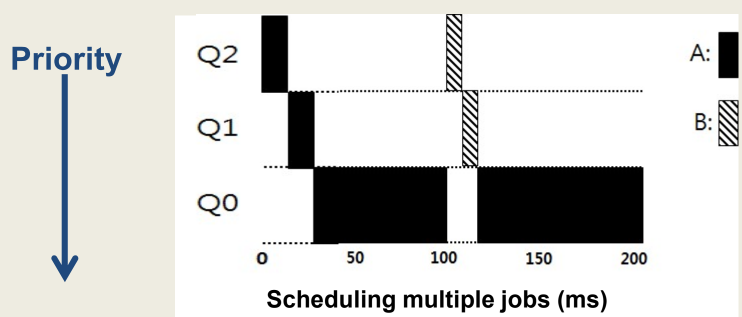
MLFQ approximates SJF

MLFQ: LONG RUNNING JOB

- Three-queue scheduler, time slice=10ms

MLFQ: BATCH AND INTERACTIVE JOBS

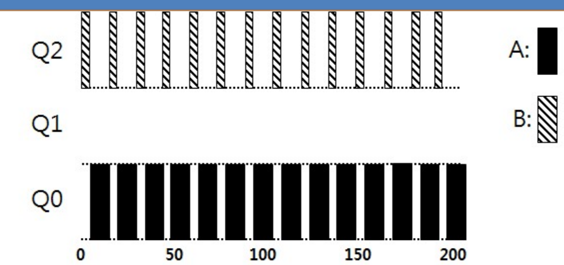
- $A_{arrival\_time}$ =0ms, A_{run_time}=200ms,
- B_{run_time} =20ms, $B_{arrival\_time}$ =100ms



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

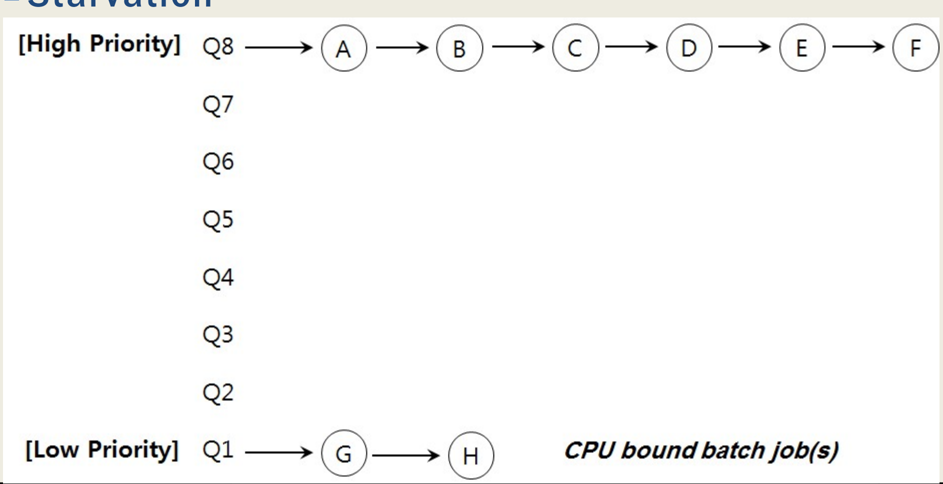
The MLFQ approach keeps interactive job(s) at the highest priority



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

MLFQ: ISSUES

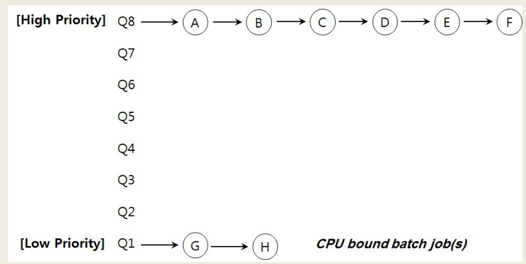
- Starvation



MLFQ: ISSUES - 2

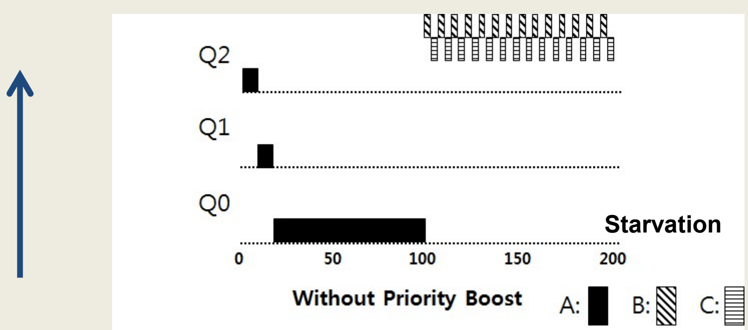
- Gaming the scheduler
  - Issue I/O operation at 99% completion of the time slice
  - Keeps job priority fixed – never lowered
- Job behavioral change
  - CPU/batch process becomes an interactive process

Priority becomes stuck



October 8, 2018	TCSS422: Operating Systems [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma	L4.55

RESPONDING TO BEHAVIOR CHANGE

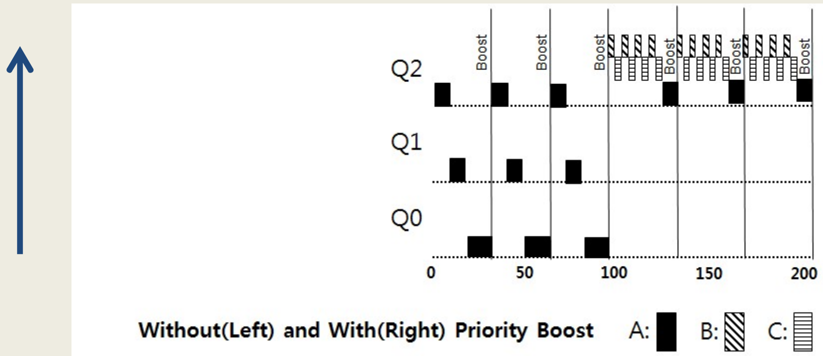


- Priority Boost
  - Reset all jobs to topmost queue after some time interval S

October 8, 2018	TCSS422: Operating Systems [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma	L4.56

RESPONDING TO BEHAVIOR CHANGE - 2

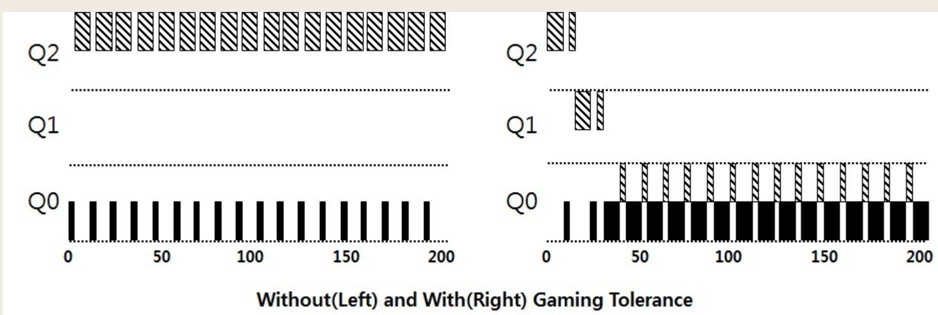
- With priority boost
  - Prevents starvation



October 8, 2018 | TCSS422: Operating Systems [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma | L4.57

PREVENTING GAMING

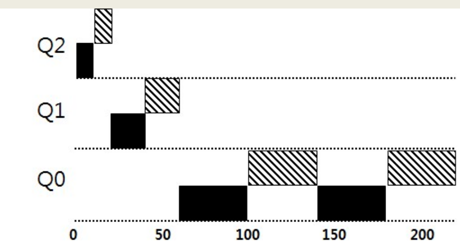
- Improved time accounting:
  - Track total job execution time in the queue
  - Each job receives a fixed time allotment
  - When allotment is exhausted, job priority is lowered



October 8, 2018 | TCSS422: Operating Systems [Fall 2018] School of Engineering and Technology, University of Washington - Tacoma | L4.58

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

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

QUESTIONS