Scheduling

Computer Operating Systems, Spring 2024

Instructor: Travis McGaha

Head TAs: Nate Hoaglund & Seungmin Han

TAs:

Adam Gorka	Haoyun Qin	Kyrie Dowling	Ryoma Harris
Andy Jiang	Jeff Yang	Oliver Hendrych	Shyam Mehta
Charis Gao	Jerry Wang	Maxi Liu	Tom Holland
Daniel Da	Jinghao Zhang	Rohan Verma	Tina Kokoshvili
Emily Shen	Julius Snipes	Ryan Boyle	Zhiyan Lu

Administrivia

- PennOS:
  - To be done in groups of 4
  - Group signup to be released soon
    - Group signup due next week (Wednesday?)
    - Those who do not form a group will be randomly assigned
    - Random assignment will prefer to keep people in pairs (unless you reach out and specify otherwise)
  - Specification to be released soon

pollev.com/tqm

- Any questions, comments or concerns from last lecture?

Lecture Outline

- **Scheduling**
- Threads High Level
- Pthreads
- Threads vs processes

OS as the Scheduler

- The scheduler is code that is part of the kernel (OS)

- The scheduler runs when a thread:
  - starts ("arrives to be scheduled"),
  - Finishes
  - Blocks (e.g., waiting on something, usually some form of I/O)
  - Has run for a certain amount of time

- It is responsible for scheduling threads
  - Choosing which one to run
  - Deciding how long to run it

Scheduler Terminology

- The scheduler has a scheduling algorithm to decide what runs next.

- Algorithms are designed to consider many factors:
  - Fairness: Every program gets to run
  - Liveness: That “something” will eventually happen
  - Throughput: amount of work completed over an interval of time
  - Wait time: Average time a “task” is “alive” but not running
  - Turnaround time: time between task being ready and completing
  - Response time: time it takes between task being ready and when it can take user input
  - Etc…

Goals

- The scheduler will have various things to prioritize
- Some examples:
- Minimizing wait time
  - Get threads started as soon as possible
- Minimizing latency
  - Quick response times and task completions are preferred
- Maximizing throughput
  - Do as much work as possible per unit of time
- Maximizing fairness
  - Make sure every thread can execute fairly

- These goals depend on the system and can conflict

Scheduling: Other Considerations

- It takes time to context switch between threads
  - Could get more work done if thread switching is minimized
- Scheduling takes resources
  - It takes time to decide which thread to run next
  - It takes space to hold the required data structures
- Different tasks have different priorities
  - Higher priority tasks should finish first

Types of Scheduling Algorithms

- **Non-Preemptive:** if a thread is running, it continues to run until it completes or until it gives up the CPU
  - First come first serve (FCFS)
  - Shortest Job First (SJF)

- **Preemptive:** the thread may be interrupted after a given time and/or if another thread becomes ready
  - Round Robin
  - Priority Round Robin
  - …

First Come First Serve (FCFS)

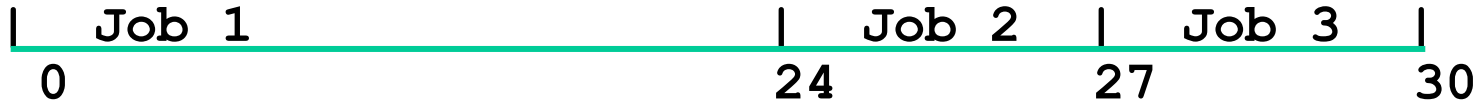
- Idea: Whenever a thread is ready, schedule it to run until it is finished (or blocks).
- Maintain a queue of ready threads
  - Threads go to the back of the queue when it arrives or becomes unblocked
  - The thread at the front of the queue is the next to run

Example of FCFS

1 CPU
Job 2 arrives slightly after job 1.
Job 3 arrives slightly after job 2

- Example workload with three "jobs":
  Job 1: 24 time units; Job 2: 3 units; Job 3: 3 units

- FCFS schedule:



- Total waiting time: 0 + 24 + 27 = 51
- Average waiting time: 51/3 = 17
- Total turnaround time: 24 + 27 + 30 = 81
- Average turnaround time: 81/3 = 27

pollev.com/tqm

- What are the advantages/disadvantages/concerns with **First Come First Serve**

- Things a scheduler should prioritize:
  - Minimizing wait time
  - Minimizing Latency
  - Maximizing fairness
  - Maximizing throughput
  - Task priority
  - Cost to schedule things
  - Cost to context Switch

- Imagine we have 1 core, and tasks of various lengths…

FCFS Analysis

- Advantages:
  - Simple, low overhead
  - Hard to screw up the implementation
  - Each thread will DEFINITELY get to run eventually.

- Disadvantages
  - Doesn’t work well for interactive systems
  - Throughput can be low due to long threads
  - Large fluctuations in average turn around time
  - Priority not taken into considerations

Shortest Job First (SJF)

- Idea: variation on FCFS, but have the tasks with the smallest CPU-time requirement run first
  - Arriving jobs are instead put into the queue depending on their run time, shorter jobs being towards the front
  - Scheduler selects the shortest job (1st in queue) and runs till completion

Example of SJF

1 CPU
Job 2 arrives slightly after job 1.
Job 3 arrives slightly after job 2

- Same example workload with three "jobs":
  Job 1: 24 time units; Job 2: 3 units; Job 3: 3 units

- FCFS schedule:

- Total waiting time: 6 + 0 + 3 = 9
- Average waiting time: 3
- Total turnaround time: 30 + 3 + 6 = 39
- Average turnaround time: 39/3 = 13

pollev.com/tqm

- What are the advantages/disadvantages/concerns with **Shortest Job First**

- Things a scheduler should prioritize:
  - Minimizing wait time
  - Minimizing Latency
  - Maximizing fairness
  - Maximizing throughput
  - Task priority
  - Cost to schedule things
  - Cost to context Switch

- Imagine we have 1 core, and tasks of various lengths…

Types of Scheduling Algorithms

- **Non-Preemptive:** if a thread is running, it continues to run until it completes or until it gives up the CPU
  - First come first serve (FCFS)
  - Shortest Job First (SJF)

- **Preemptive:** the thread may be interrupted after a given time and/or if another thread becomes ready
  - Round Robin
  - Priority Round Robin
  - …

Round Robin

- Sort of a preemptive version of FCFS
  - Whenever a thread is ready, add it to the end of the queue.
  - Run whatever job is at the front of the queue

- BUT only led it run for a fixed amount of time (quantum).
  - If it finishes before the time is up, schedule another thread to run
  - If time is up, then send the running thread back to the end of the queue.

Example of Round Robin

- Same example workload:

  Job 1: 24 units, Job 2: 3 units, Job 3: 3 units
- RR schedule with time quantum=2:

- Total waiting time: (0 + 4 + 2) + (2 + 4) + (4 + 3) = 19
  - Counting time spent waiting between each “turn” a job has with the CPU
- Average waiting time: 19/3 (~6.33)
- Total turnaround time: 30 + 9 + 10 = 49
- Average turnaround time: 49/3 (~16.33)

Poll Everywhere

pollev.com/tqm

- What are the advantages/disadvantages/concerns with **Round Robin**

- Things a scheduler should prioritize:
  - Minimizing wait time
  - Minimizing Latency
  - Maximizing fairness
  - Maximizing throughput
  - Task priority
  - Cost to schedule things
  - Cost to context Switch

- Imagine we have 1 core, and tasks of various lengths…

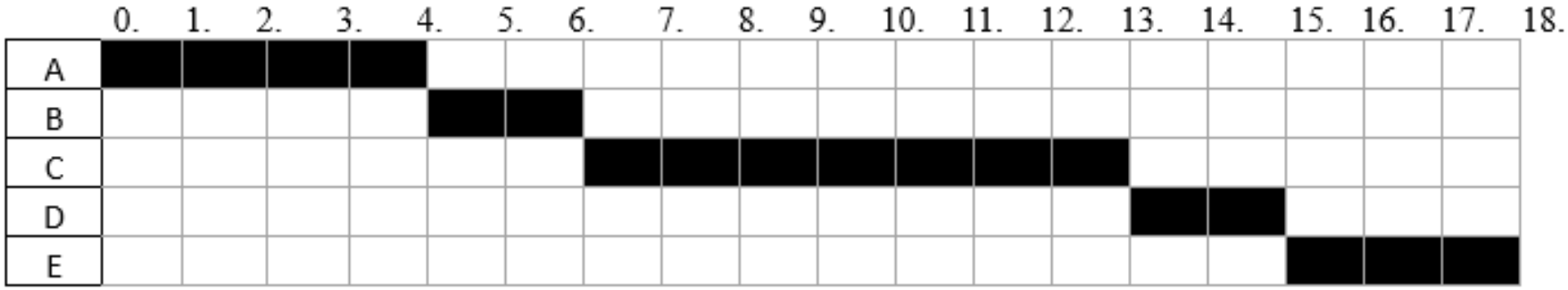
Round Robin Analysis

- Advantages:
  - Still relatively simple
  - Can works for interactive systems
- Disadvantages
  - If quantum is too small, can spend a lot of time context switching
  - If quantum is too large, approaches FCFS
  - Still assumes all processes have the same priority.
- Rule of thumb:
  - Choose a unit of time so that most jobs (80-90%) finish in one usage of CPU time

Round Robin Practice!

- Assume that we had the following set of processes that ran on a single CPU following the First Come First Serve (FCFS) scheduling policy.
- If we expressed this with a drawing, where a black square represents that the process is executing, we would get:

Process	Arrival Time	Finishing Time
A	0	4
B	1	6
C	2	13
D	3	15
E	4	18



Round Robin Practice!

- Instead, lets switch our algorithm to round-robin with a time quantum of 3 time units.

Process	Arrival Time	Finishing Time
A	0	4
B	1	6
C	2	13
D	3	15
E	4	18

- You can assume:
  - Context switching and running the Scheduler are instantaneous.
  - If a process arrives at the same time as the running process' time slice finishes, the one that just arrived goes into the ready queue before the one that just finished its time slice.

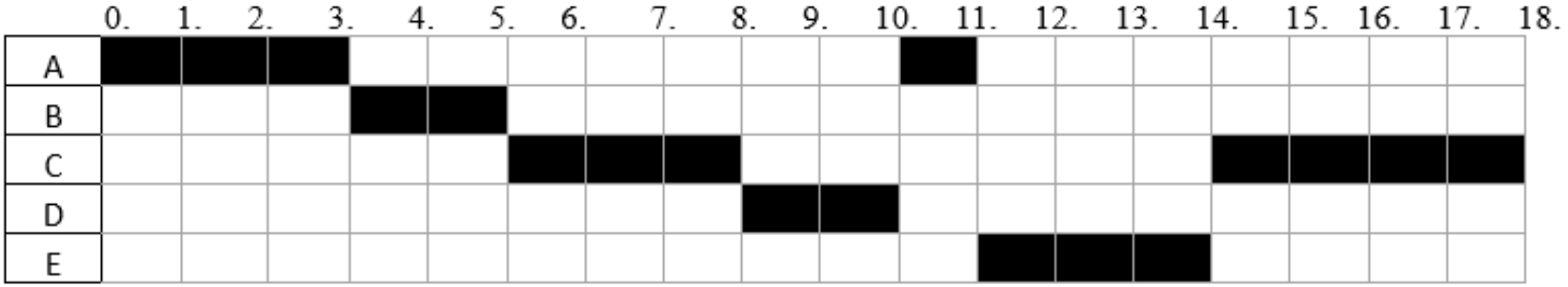
	0.	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.	11.	12.	13.	14.	15.	16.	17.	18.
A																			
B																			
C																			
D																			
E																			

Round Robin Practice!

- Instead, lets switch our algorithm to round-robin with a time quantum of 3 time units.

Process	Arrival Time	Finishing Time
A	0	4
B	1	6
C	2	13
D	3	15
E	4	18

- You can assume:
  - Context switching and running the scheduler are instantaneous.
  - If a process arrives at the same time as the running process' time slice finishes, the one that just arrived goes into the ready queue before the one that just finished its time slice.



RR Variant: PennOS Scheduler

- In PennOS you will have to implement a priority scheduler based mostly off of round robin.

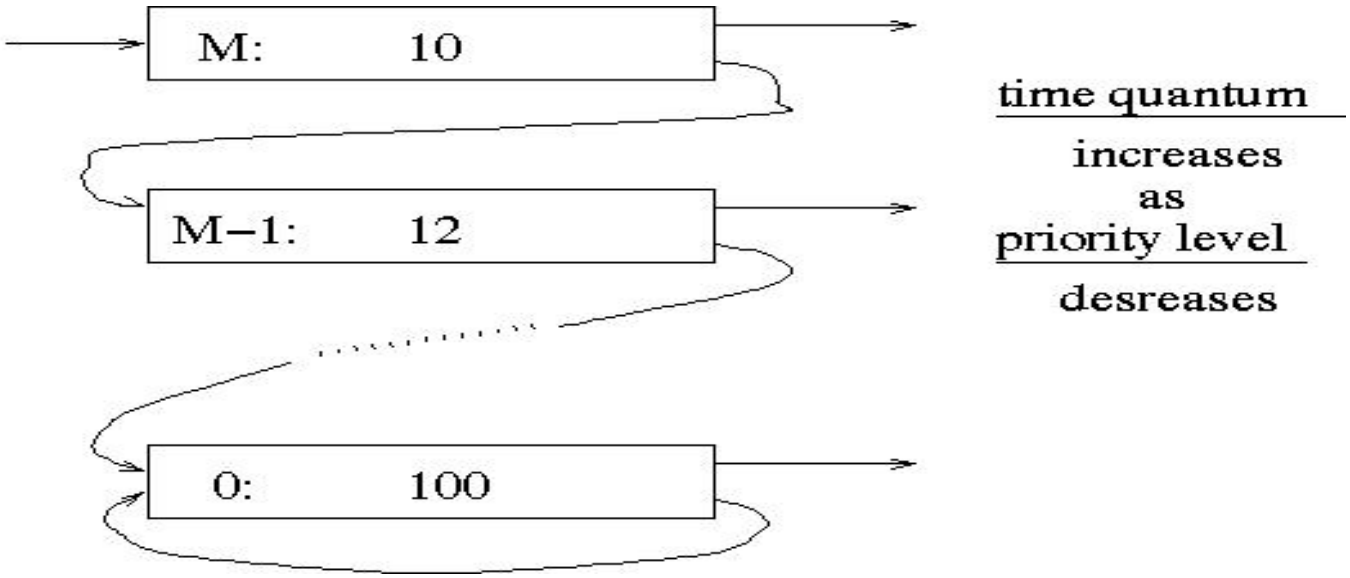
- You will have 3 queues, each with a different priority (0, 1, 2)
  - Each queue acts like normal round robin within the queue

- You spend time quantum processing each queue proportional to the priority
  - Priority 0 is scheduled 1.5 times more often than priority 1
  - Priority 1 is scheduled 1.5 times more often than priority 2

RR Variant: Priority Round Robin

- Same idea as round robin, but with multiple queues for different priority levels.
- Scheduler chooses the first item in the highest priority queue to run
- Scheduler only schedules items in lower priorities if all queues with higher priority are empty.

RR Variant: Multi Level Feedback



- Each priority level has a ready queue, and a time quantum
- Thread enters highest priority queue initially, and lower queue with each timer interrupt
- If a thread voluntarily stops using CPU before time is up, it is moved to the end of the current queue
- Bottom queue is standard Round Robin
- Thread in a given queue not scheduled until all higher queues are empty

Multi Level Feedback Analysis

- Threads with high I/O bursts are preferred
  - Makes higher utilization of the I/O devices
  - Good for interactive programs (keyboard, terminal, mouse is I/O)
- Threads that need the CPU a lot will sink to lower priority, giving shorter threads a chance to run
- Still have to be careful in choosing time quantum
- Also have to be careful in choosing how many layers

Multi Level Feedback Variants: Priority

- Can assign tasks different priority levels upon initiation that decide which queue it starts in
  - E.g. the scheduler should have higher priority than HelloWorld.java
- Update the priority based on recent CPU usage rather than overall cpu usage of a task
  - Makes sure that priority is consistent with recent behavior
- Many others that vary from system to system

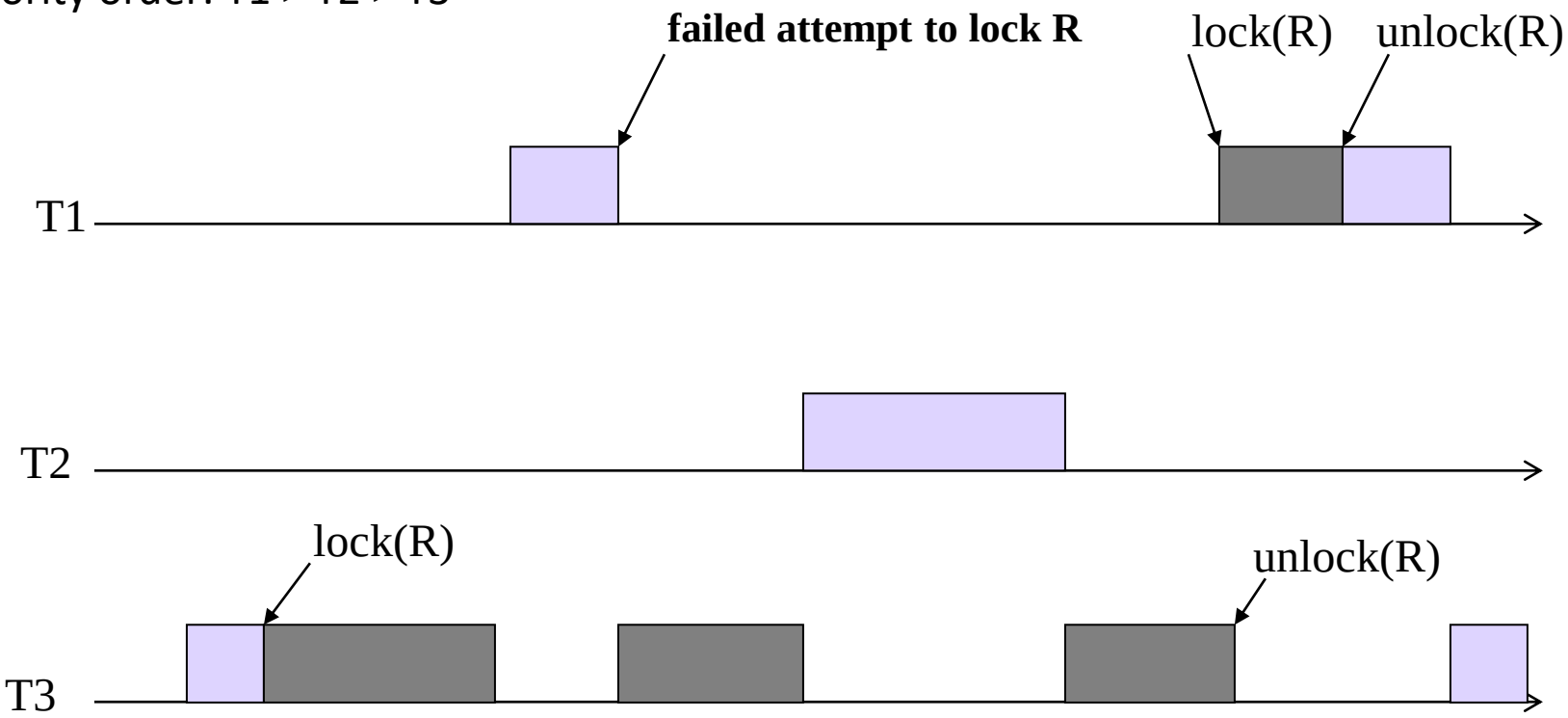
Why did we talk about this?

- Scheduling is fundamental towards how computer can multi-task
- This is a great example of how “systems” intersects with algorithms :)
- It shows up occasionally in the real world :)
  - Scheduling threads with priority with shared resources can cause a priority inversion, potentially causing serious errors.

What really happened on Mars Rover Pathfinder, Mike Jones.
http://www.cs.cornell.edu/courses/cs614/1999sp/papers/pathfinder.html

The Priority Inversion Problem

Priority order: T1 > T2 > T3



T2 is causing a higher priority task T1 wait !

More

- For those curious, there was a LOT left out

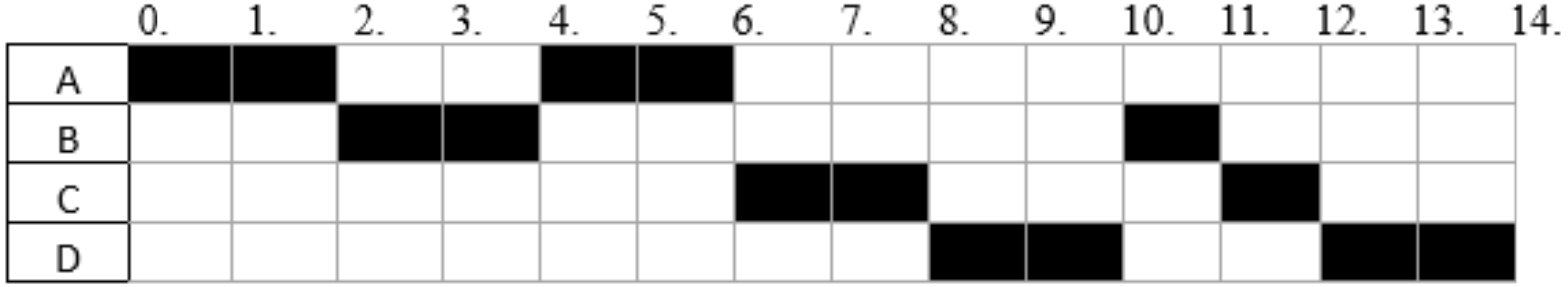
- RTOS (Real Time Operating Systems)
  - For real time applications
  - CRITICAL that data and events meet defined time constraints
  - Different focus in scheduling. Throughput is de-prioritized

- Fair-share scheduling
  - Equal distribution across different users instead of by processes

- Etc.

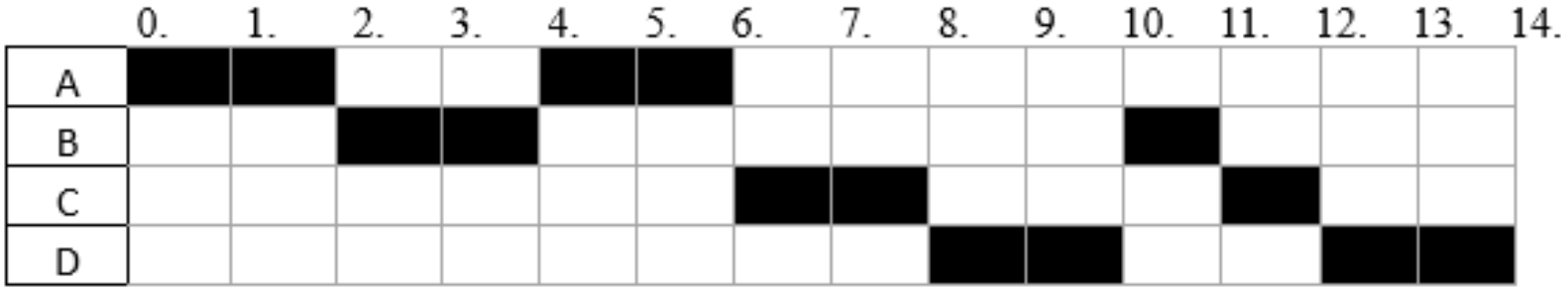
More Round Robin Practice

- Four processes are executing on one CPU following round robin scheduling:



- You can assume:
  - All processes do not block for I/O or any resource.
  - Context switching and running the Scheduler are instantaneous.
  - If a process arrives at the same time as the running process' time slice finishes, the one that just arrived goes into the ready queue before the one that just finished its time slice.

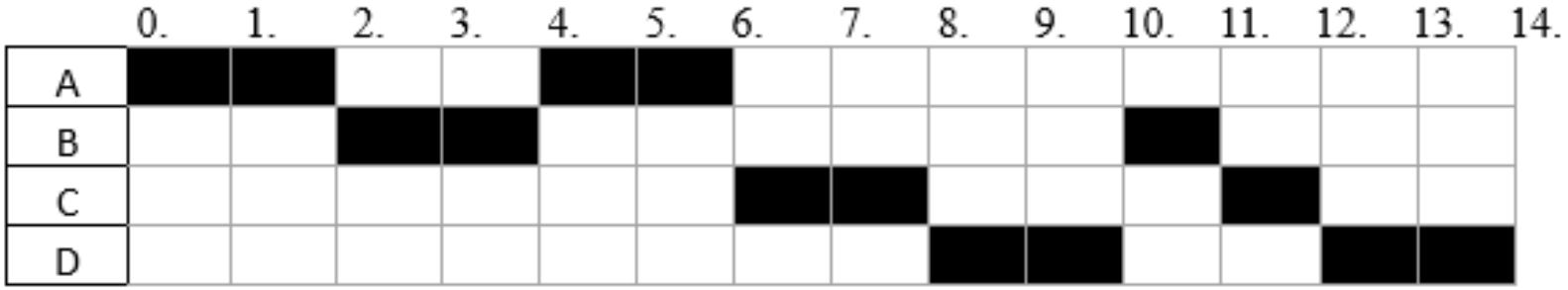
More Round Robin Practice



- All processes do not block for I/O or any resource.
- Context switching and running the Scheduler are instantaneous.
- If a process arrives at the same time as the running process' time slice finishes, the one that just arrived goes into the ready queue before the one that just finished its time slice.

- What is the earliest time that process C could have arrived?
- Which processes are in the ready queue at time 9?
- If this algorithm used a quantum of 3 instead of 2, how many fewer context switches would there be?

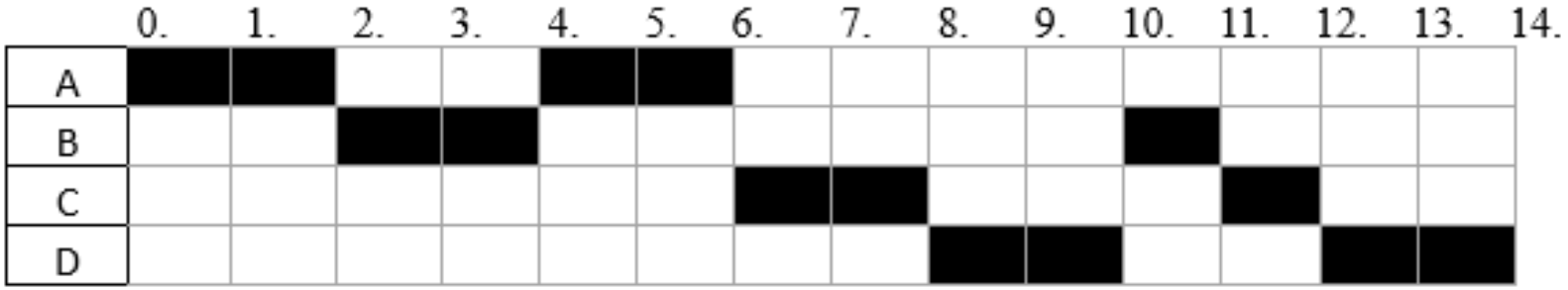
More Round Robin Practice



- All processes do not block for I/O or any resource.
- Context switching and running the Scheduler are instantaneous.
- If a process arrives at the same time as the running process' time slice finishes, the one that just arrived goes into the ready queue before the one that just finished its time slice.

- What is the earliest time that process C could have arrived?
  - If C arrived at time 0, 1, or 2, it would have run at time 4
  - C could have shown up at time 3 and come after A in the queue
  - C showed up at time 3 at earliest

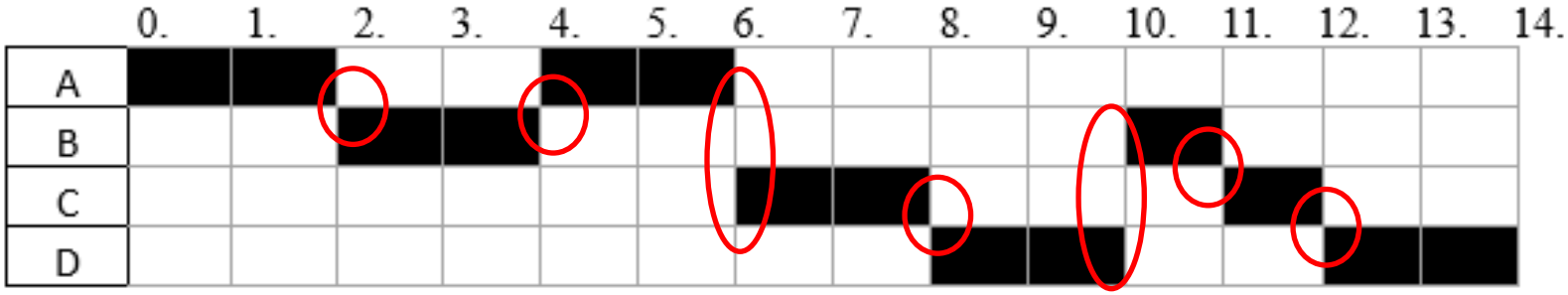
More Round Robin Practice



- All processes do not block for I/O or any resource.
- Context switching and running the Scheduler are instantaneous.
- If a process arrives at the same time as the running process' time slice finishes, the one that just arrived goes into the ready queue before the one that just finished its time slice.

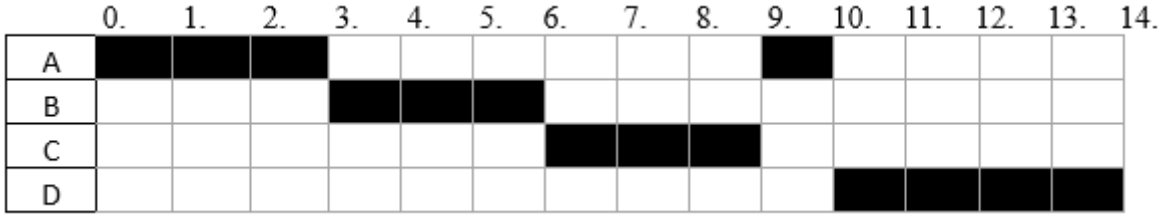
- Which processes are in the ready queue at time 9?
  - D is running, so it is not in the queue
  - A has finished
  - B and C still have to finish, so they are in the queue.

More Round Robin Practice

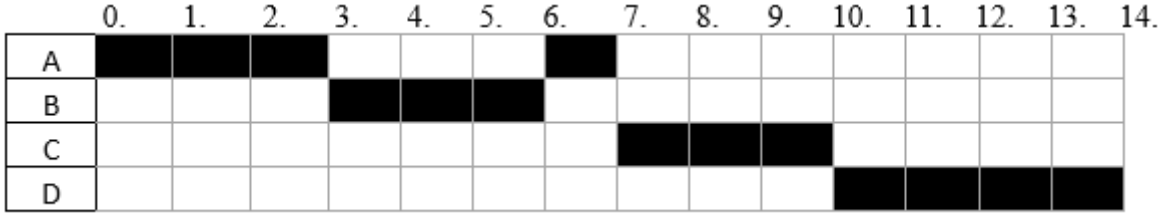


- If this algorithm used a quantum of 3 instead of 2, how many fewer context switches would there be?
  - Currently there are 7 context switches
  - If quantum was 3:

Depends on if C shows up at time 3 or 4



  - Or:



Either way, only 4 context switches, so 3 less than quantum = 2