# History & Scheduler (Start)

## Computer Operating Systems, Spring 2025

| | | | | |
|---|---|---|---|---|
| **Instructors:** | Joel Ramirez | Travis McGaha | | |
| **Head TAs:** | Ash Fujiyama | Emily Shen | Maya Huizar | |

**TAs:**

| | | | | |
|---|---|---|---|---|
| Ahmed Abdellah | Bo Sun | Joy Liu | Susan Zhang | Zihao Zhou |
| Akash Kaukuntla | Connor Cummings | Khush Gupta | Vedansh Goenka | |
| Alexander Cho | Eric Zou | Kyrie Dowling | Vivi Li | |
| Alicia Sun | Haoyun Qin | Rafael Sakamoto | Yousef AlRabiah | |
| August Fu | Jonathan Hong | Sarah Zhang | Yu Cao | |

## Poll Everywhere

pollev.com/tqm

- How are you doing? What do you know about ENIAC?

## Administrivia

- Penn-shell is out (this shouldn't be news)!
  - ***Full thing is due (Fri, Feb 28) (This more week!)***
  - ***Done in partners***
    - Everything was covered already that you would need…
- Midterm is Thursday next week
  - Old exams and exam policies are posted on the course website
  - Review session in Recitation Thursday this week!!!!!!!!!!!! (7pm in Towne 217)
  - Some midterm review in Lecture Tuesday Next Week
- SIGCSE TS
  - Lecture on Thursday will be on ZOOOOOM
  - Some office hours moving around as well. Calendar updated soon.

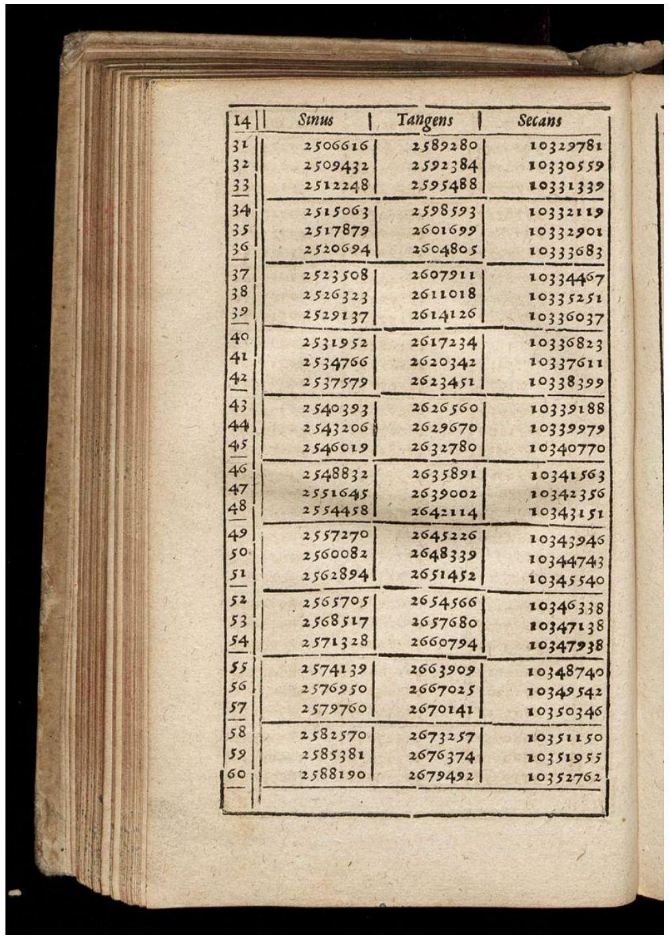
## Lecture Outline

- **Women as the first Operating System**
- Scheduling History
- Scheduling Overview
- Types of Scheduling Algorithms

## Mathematical Tables

- Before non-human calculators, you'd look up the result of a computation in a table. (Logarithm Tables, Square Root Tables …. )

- These were calculated by "Human-Computers"
  - Difficult to make tables, but made calculations faster once you had them



| 14 | Sinus | Tangens | Secans |
|---|---|---|---|
| 31 | 2506616 | 2589280 | 10329781 |
| 32 | 2509432 | 2592384 | 10330559 |
| 33 | 2512248 | 2595488 | 10331339 |
| 34 | 2515063 | 2598593 | 10332119 |
| 35 | 2517879 | 2601699 | 10332901 |
| 36 | 2520694 | 2604805 | 10333683 |
| 37 | 2523508 | 2607911 | 10334467 |
| 38 | 2526323 | 2611018 | 10335251 |
| 39 | 2529137 | 2614126 | 10336037 |
| 40 | 2531952 | 2617234 | 10336823 |
| 41 | 2534766 | 2620342 | 10337611 |
| 42 | 2537579 | 2623451 | 10338399 |
| 43 | 2540393 | 2626560 | 10339188 |
| 44 | 2543206 | 2629670 | 10339979 |
| 45 | 2546019 | 2632780 | 10340770 |
| 46 | 2548832 | 2635891 | 10341563 |
| 47 | 2551645 | 2639002 | 10342356 |
| 48 | 2554458 | 2642114 | 10343151 |
| 49 | 2557270 | 2645226 | 10343946 |
| 50 | 2560082 | 2648339 | 10344743 |
| 51 | 2562894 | 2651452 | 10345540 |
| 52 | 2565705 | 2654566 | 10346338 |
| 53 | 2568517 | 2657680 | 10347138 |
| 54 | 2571328 | 2660794 | 10347938 |
| 55 | 2574139 | 2663909 | 10348740 |
| 56 | 2576950 | 2667025 | 10349542 |
| 57 | 2579760 | 2670141 | 10350346 |
| 58 | 2582570 | 2673257 | 10351150 |
| 59 | 2585381 | 2676374 | 10351955 |
| 60 | 2588190 | 2679492 | 10352762 |

## Human Computers

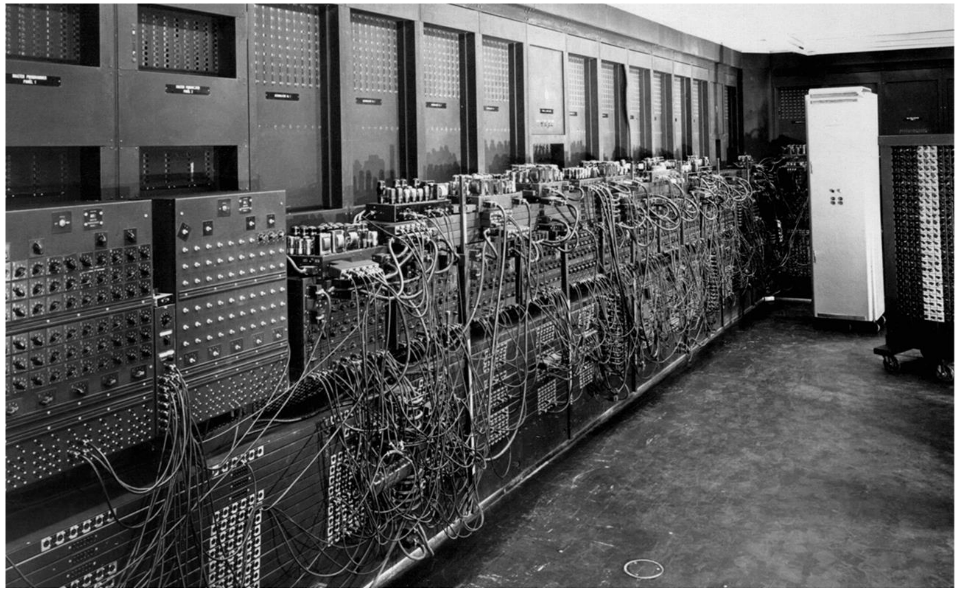
Human computers were employed for math calculations, not yet “computer programming” as we know it today

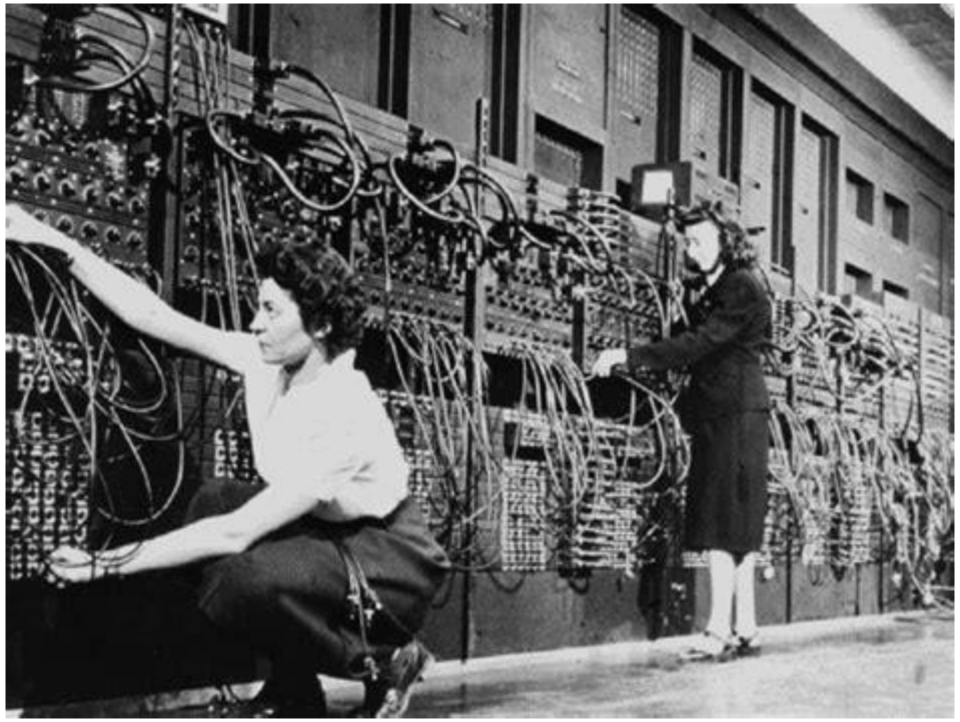
Gendered as a “woman’s job”

- **Kilogirl**: 1,000 hours of computing work
- “Requires <u>patience</u>, <u>persistence</u>, and a <u>capacity for detail</u>… that many girls have”
- Likened to secretary work: both manual and also meant to assist (male) professionals
- Still requires highly technical knowledge

## ENIAC - 1945

- Electronic Numerical Integrator and Computer
- First programmable, electronic, general-purpose, digital computer
- Developed in Moore Building for the military: artillery trajectory tables
- Non-seniors: keep your eyes out for Feb 15, 2026 (80th yr anni)





## ENIAC - 40 units

Arithmetic

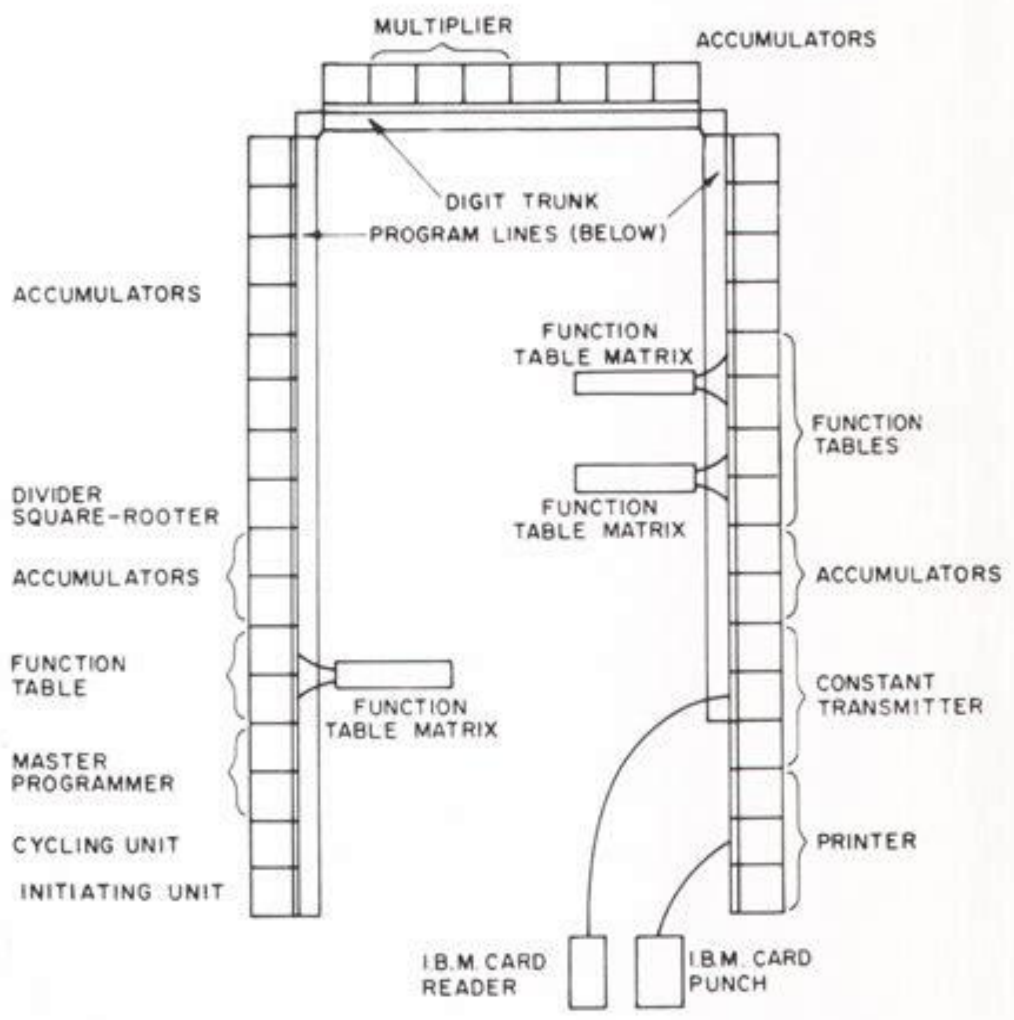
- 20 accumulators* (registers that can add and subtract within itself)
- 1 fast multiplier
- 1 divider and square rooter

Memory & I/O

- 3 function tables*
- Constant transmitter*
- Punch card reader*
- Punch card writer*

Governing (Control Flow)

- Initiating unit
- Cycling unit
- Master programmer*



## ENIAC - 40 units

Arithmetic

- 20 accumulators* (registers that can add and subtract within itself)
- 1 fast multiplier
- 1 divider and square rooter

Memory & I/O

- 3 function tables*
- Constant transmitter*
- Punch card reader*
- Punch card writer*

Governing (Control Flow)

- Initiating unit
- Cycling unit
- Master programmer*

For math calculations, how much faster?

- 500 multiplications and 5,000 additions every second
- 30 hours of hand calculations became 20 seconds of computer runtime
  - 3 days of setup, not including debug time

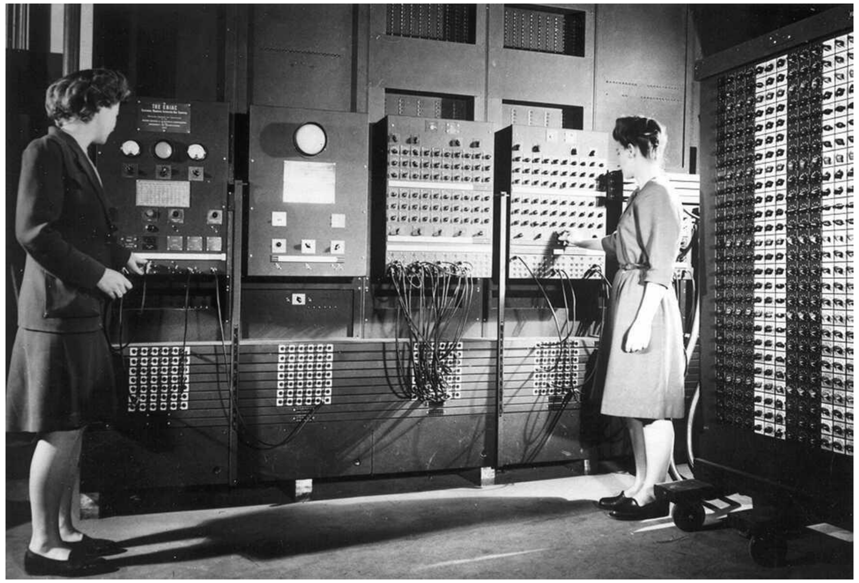
## Who got to touch the computer?

- The engineers who built the computer
- Secretaries who generated punch card inputs
- Computer programmers and operators
- University researchers
- Students
- Anyone who walked in when no-one was looking?

## The ENIAC 6

*“The Eniac’s Operating System”*

First Programmers of ENIAC:
- Betty Holberton
- Jean Bartik
- Ruth Teitelbaum
- Kathleen Antonelli
- Marlyn Meltzer
- Frances Spence



## The Necessity of Programmers

- Operating the ENIAC is complex!!
  - For a program that would take 2 lines of pseudocode,  there is a 10-page tutorial with 10 figures and 3 tables
- 40 separate units that can run in parallel
- Machines only know basic math! (and a little control flow)
  - A lot of overhead fell on the humans operating the computer

## The ENIAC 6: providing the overhead

Programmer / Compiler

- Translate the mathematical problem into smaller programs ENIAC can run

loader

- Set up the 40 panels of ENIAC to run the program

OS

- Operate the computer:
  - Start the computer, ensure everything is running well
  - Manage "I/O" to handle intermediate values if they are too large to handle
  - Manage where values are stored in hardware and ensure data is being written / read following correct timing requirements

operator

- Debug ☺
  - If the program was wrong
  - Inevitably: when a piece of hardware broke

Note: not much really in terms of scheduling (yet). ENIAC only did one program at a time.

## The ENIAC 6

"The ENIAC was a son-of-a-bitch to program" – Jean Jennings Bartik

- Simultaneously navigating an all-male landscape in Penn's Engineering facilities
  - Received sub-professional occupation classification
  - Unable to be authored in any papers or manuals
  - Used to solidify the software-hardware gender divide
  - Unprivileged access to the ENIAC - learned programming in a separate room
  - Served as "hostesses" during ENIAC's public release

## Lecture Outline

- Women as the first Operating System
- **Scheduling History**
- Scheduling Overview
- Types of Scheduling Algorithms

## Computer Access

- Revisiting the question of: who do we let access the computer…?
- This affects how we will be able to schedule multiple programs to run within a day

## Idea #1: Everyone gets access!

What issues will we run into?

## Idea #1: Everyone gets access!

What issues will we run into?

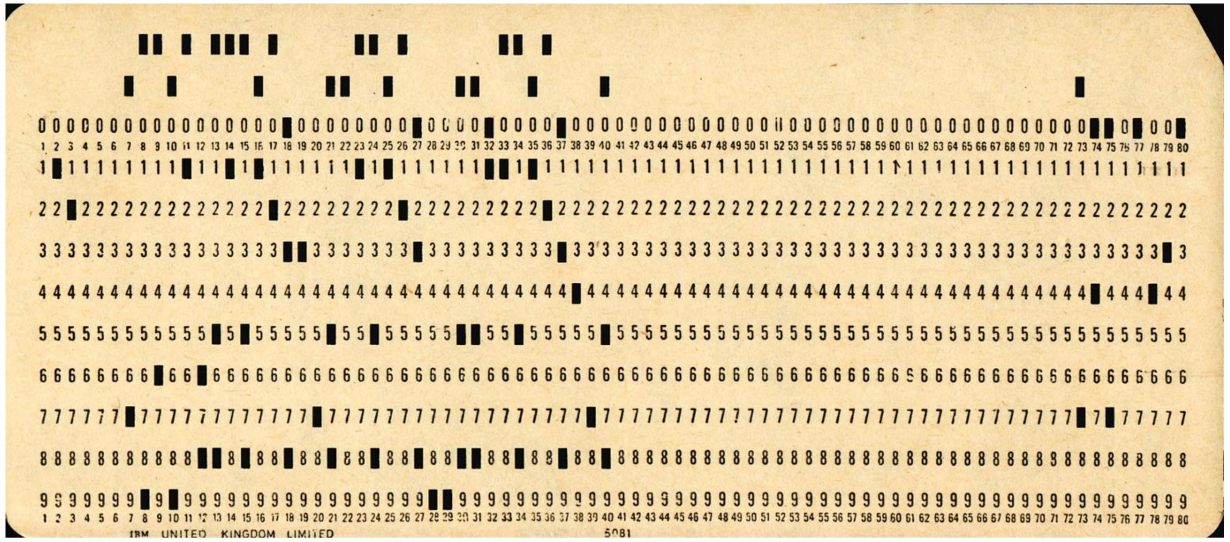
- "Rush hour" - what if people all try to get their program run at the same time?
- Large gaps of time when computer runs without a program - wasted cycles
- Does everyone in the building know how to use the computer properly?
- What happens if the computer fails / halts?

## Idea #2: Reservations

- Block out a time when only you are allowed to use the computer - shared calendar
- fixes the problem of multiple people trying to access the computer at the same time
- Need multiple operators working round-the-clock in shifts in case of failure
- What is wrong with this idea?

## Idea #3: Dropoff

- People who want a job done submit their stack(s) of punch cards to a "front desk"
- Operators gather the stacks together and inserts programs in chronological order*
- Computer runs through the jobs sequentially, and will provide outputs once finished with *all jobs* in the batch



## Batch Processing

- 1950’s
- First instance of computer handling the scheduling
- Operator gets stacks of punch cards from multiple jobs, groups into a mega-stack and then shoves into computer
- Computer handles the order of running and outputting
- Required its own language: Job Control Language (JCL)

## Poll Everywhere

pollev.com/tqm

- Comparing batch processing to the reservation system:
  - Which has better latency? (average speed of 1 job)
  - Which has better throughput? (# jobs per time)

## Efficiency towards Automation

- Why do we care about latency and throughput?

- Computing has kinda been about automation

- How to make things faster
  - Making the calculations itself faster
  - But also automating the “operators”

- Is efficiency the end-all be-all? This is for after spring break :)

## Lecture Outline

- Women as the first Operating System
- Scheduling History
- Scheduling Overview
- Types of Scheduling Algorithms

## 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

## Lecture Outline

- Women as the first Operating System
- Scheduling History
- Scheduling Overview
- **Types of Scheduling Algorithms**

## 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)

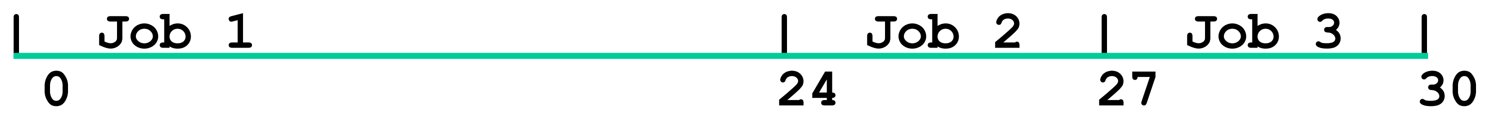
- Idea: Whenever a thread is ready, schedule it to run until it is finished (or blocks).
- Maintain a queue of ready threads
  - Threads go to the back of the queue when it arrives or becomes unblocked
  - The thread at the front of the queue is the next to run

## 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

## Poll Everywhere

pollev.com/tqm

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

- 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 (finite) 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:

```
| Job 2 | Job 3  |   Job 1                    |
0       3        6                           30
```

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

## Poll Everywhere

**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 (finite) 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:

```
|Job 1|Job 2|Job 3|Job 1|Jo2|Jo3|Job 1|  …     |Job 1|
0     2     4     6     8   9   10    12,14…        30
```

- 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**

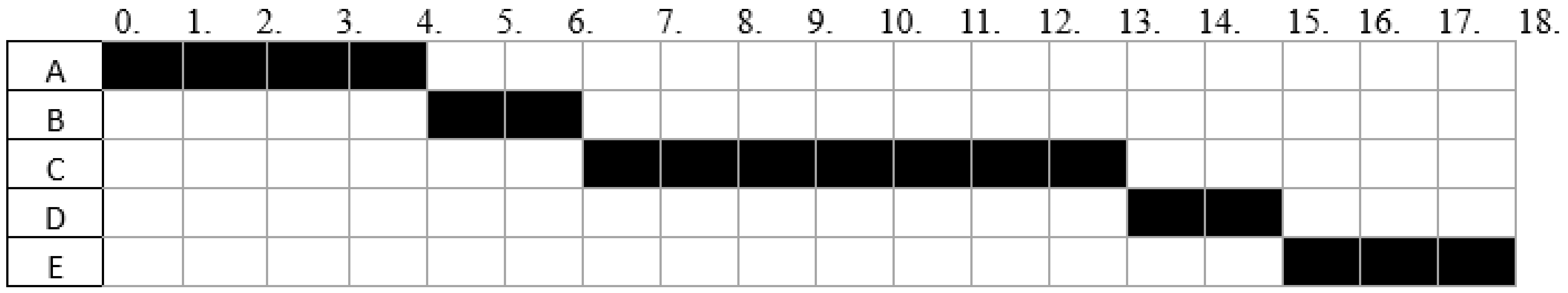
- Things a scheduler should prioritize:
  - Minimizing wait time
  - Minimizing Latency
  - Maximizing fairness
  - Maximizing throughput
  - Task priority
  - Cost to schedule things
  - Cost to context Switch

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

## Round Robin 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 | | | | | | | | | | | | | | | | | | | |

## 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.

## That’s all!

- See you next time!