# OS Project Spring 2019

•••

Professor Patrick Bobbie

Nawal Ahmed & Jordan Hasty

## Intro

Virtual machine written in C++

C++ allows greater control of memory than languages like Java

Developed in CodeLite

## Project Difficulties

Problems coordinating

Loss of most group members

Issues with interpreting the project specifications

Issues with determining how to effectively represent metrics data

## Design Approach

Modular design

Modules related to physical components are written as instantiable classes (Disk, RAM, MMU)

Non-instantiable modules are written as namespaces

main() serves as the Driver of the VM

All data is stored as unsigned integers to maintain consistency

Interfaces through command line

## Implementation - Memory Module

An instantiable class

All data stored as unsigned 8-bit integers so RAM is byte-addressable

Only contains functions for reading and writing to the array of data

Read/write functions similar to the Windows API's Read/WriteProcessMemory functions

Allows any data type to be read/written to/from memory and bytes are automatically concatenated by using templated functions that utilize a buffer

## Implementation - Disk Module

Essentially the same as the Memory module

## Implementation - Loader Module

Written as a namespace

Contains all functions responsible for transferring data between the Disk and RAM

Opens the DataFile and loads data onto the disk according to the format of the file

Parses data and writes job data to each job’s PCB

Loads program data into given frames in RAM

## Implementation - Memory Management Unit (MMU)

An instantiable class

Effectively serves as an interface between the RAM and the rest of the OS

Manages the allocation and release of page frames

Handles translation from logical to “physical” addresses

## Implementation - Scheduler

Long-term and short-term schedulers are not written as distinctive modules

Written inside main() as each of them needs access to many VM modules

Long-term scheduler determines the order in which jobs are placed into the ready queue

Short-term scheduler works closely with CPUs and Dispatcher and works with moving jobs in and out of the wait queue & context switching

## Implementation - Dispatcher

Dispatcher also written in main()

Embedded within the Short-term scheduler as it works closely with the scheduler and CPUs

Picks processes from the ready queue if the wait queue is empty

Assigns processes to idle CPUs

## Implementation - CPU Module

Written as an instantiable class

Contains functions for decoding process instructions

Conversion routines between hex/decimal/binary unnecessary as all data are treated as unsigned 32-bit integers

Instructions are decoded through bit-shifting and masking

Uses a switch statement that contains instructions for each opcode case

Works closely with assigned process's PCB and the MMU for I/O operations

# Experimental Runs - Program 1

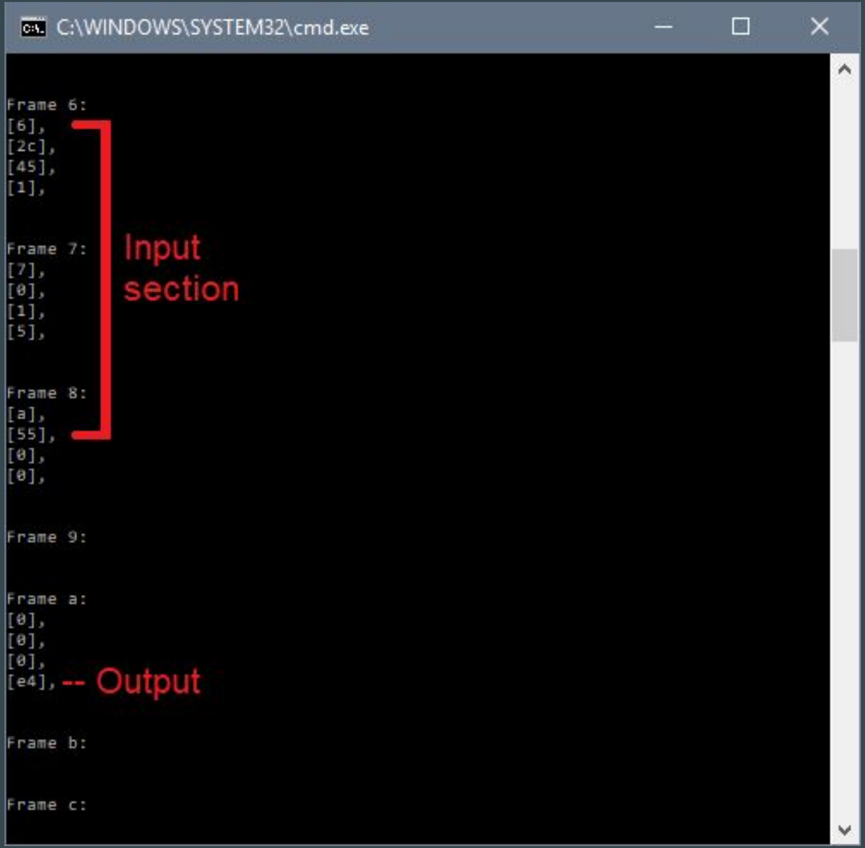
Program designed to find the sum of an array of integers

Shown is the data in RAM

Input = 6 + 0x2c + 0x45 + 1 + 7 + 0 + 1 + 5 + 0xA + 0x55

Output = 0xE4 (correct)

The program runs correctly



```
Frame 6:
[6],
[2c],
[45],
[1],

Frame 7:
[7],
[0],
[1],
[5],

Frame 8:
[a],
[55],
[0],
[0],

Frame 9:

Frame a:
[0],
[0],
[0],
[e4], -- Output

Frame b:

Frame c:
```

## Experimental Runs - Program 2

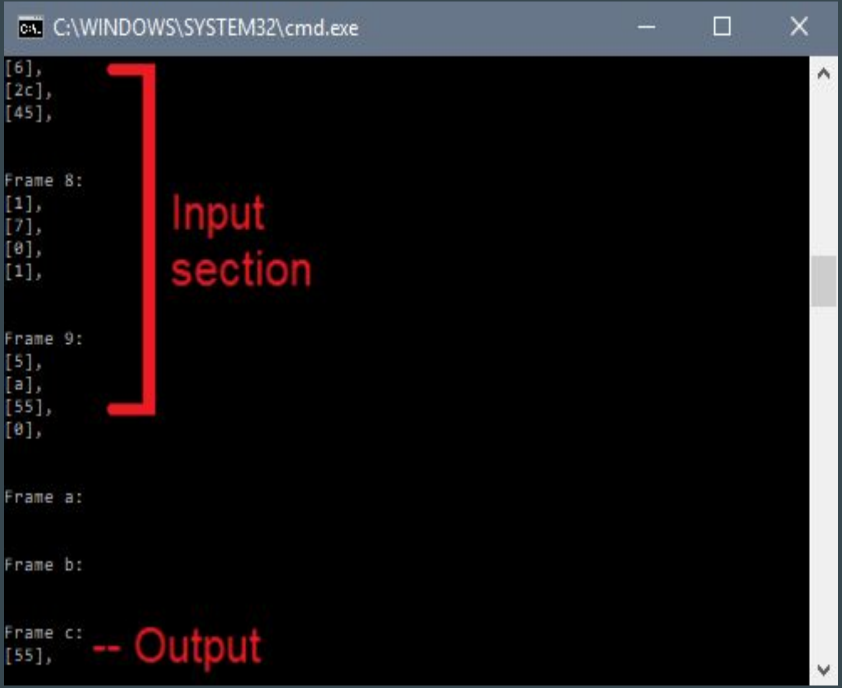
Program designed to find the largest integer in an array

Shown is the data in RAM

Input = 6, 0x2c, 0x45, 1, 7, 0, 1, 5, 0xA, 0x55

Output = 0x55 (correct)

The program runs correctly



```
[6],
[2c],
[45],


Frame 8:
[1],
[7],
[0],
[1],


Frame 9:
[5],
[a],
[55],
[0],


Frame a:


Frame b:


Frame c:
[55],
```

## Experimental Runs - Program 3

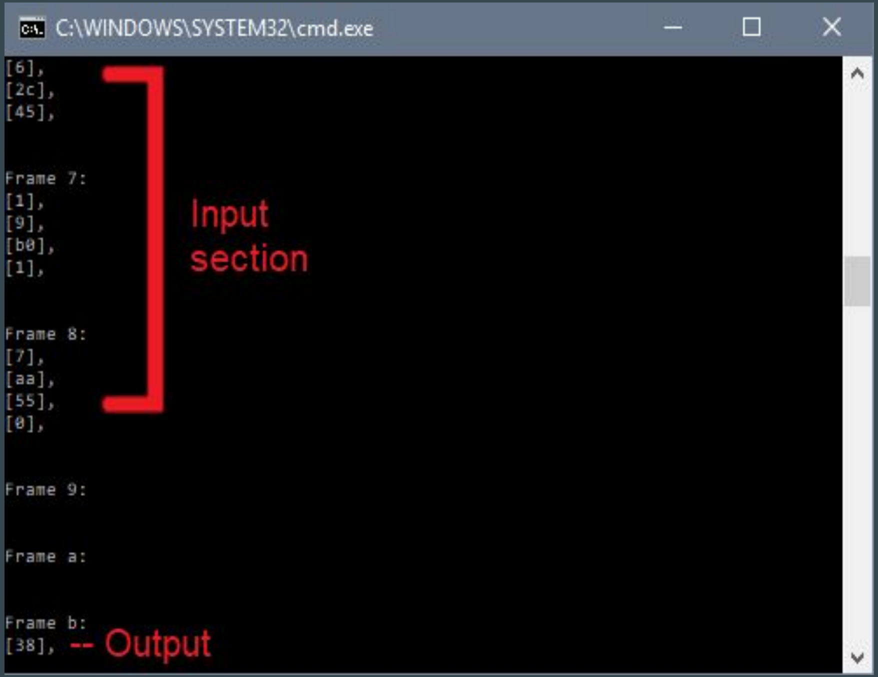
Program designed to find the average value from an array

Shown is the data in RAM

Input = 6, 0x2c, 0x45, 1, 9, 0xB0, 1, 7, 0xAA, 0x55

Output = 0x38 (correct)

The program runs correctly



```
[6],
[2c],
[45],


Frame 7:
[1],
[9],
[b0],
[1],


Frame 8:
[7],
[aa],
[55],
[0],


Frame 9:


Frame a:


Frame b:
[38],
```

# Experimental Runs - Program 4

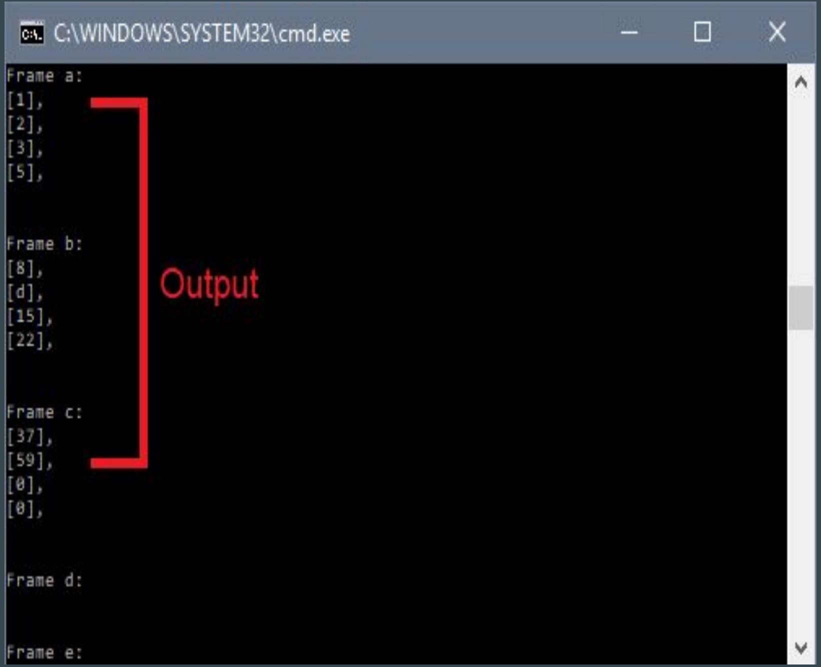
Program designed to output the first 10 numbers of the Fibonacci Sequence

Shown is the data in RAM

Output = 1, 2, 3, 5, 8, 0xD, 0x15, 0x22, 0x37, 0x59

Each number is the sum of the previous two

The program runs correctly



```
Frame a:
[1],
[2],
[3],
[5],


Frame b:
[8],
[d],
[15],
[22],


Frame c:
[37],
[59],
[0],
[0],


Frame d:


Frame e:
```

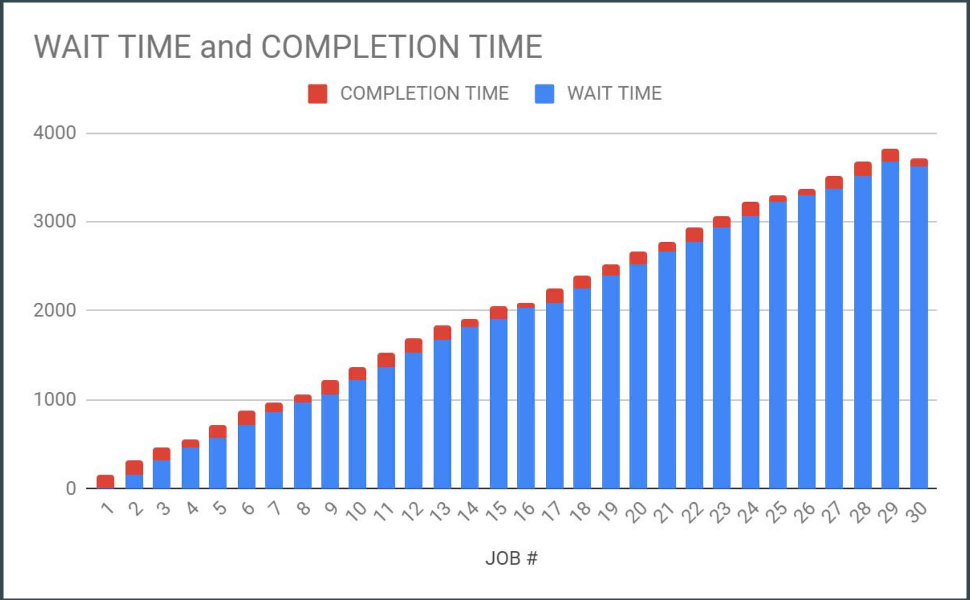
## Data - 1 CPU, FCFS Scheduling

Shown in this graph are the execution times for the simulation using: 1 CPU, FCFS scheduling policy

Wait times increase with job # as jobs are scheduled in a first come, first serve policy

Maximum RAM usage is only 5% as the VM is only utilizing one CPU so there aren't many processes in memory concurrently

Very long wait times



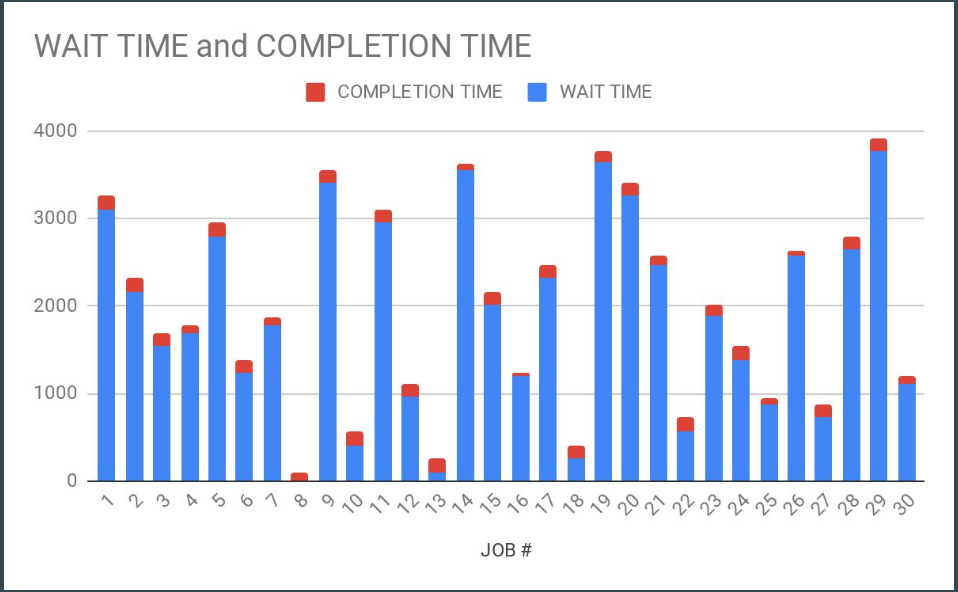
# Data - 1 CPU, Priority Scheduling

Shown in this graph are the execution times for the simulation using: 1 CPU, Priority scheduling policy

Jobs are selected in order of priority

No visible trends as program size varies

VM still only utilizes a maximum of 5% RAM as few programs are in memory concurrently



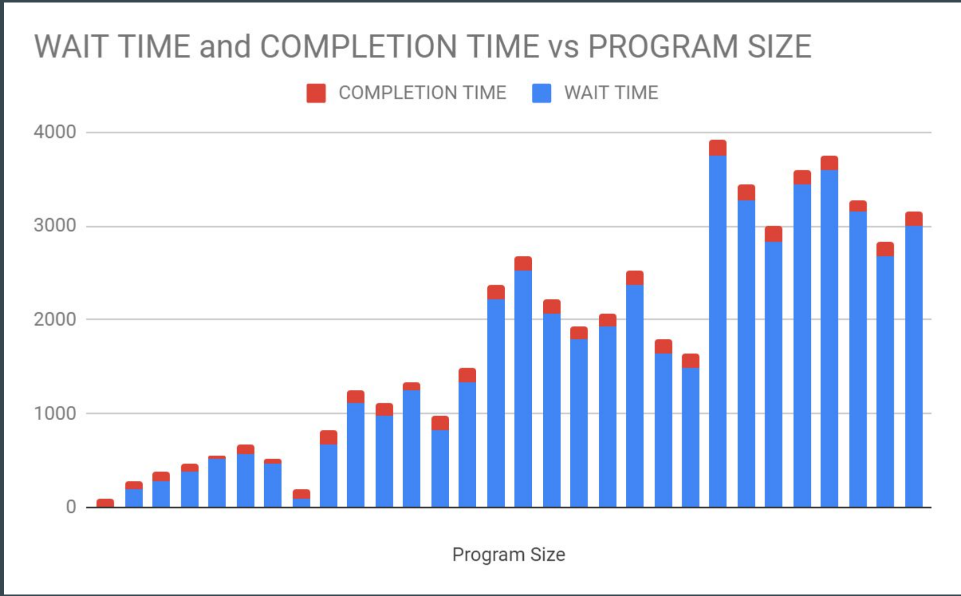
## Data - 1 CPU, SJF Scheduling

Shown in this graph are the execution times for the simulation using: 1 CPU, SJF scheduling policy

Programs are scheduled ordered by program size

Can see an upward trend in total execution time vs program size

Total execution times are not in perfect ascending order due to blocking and context switching



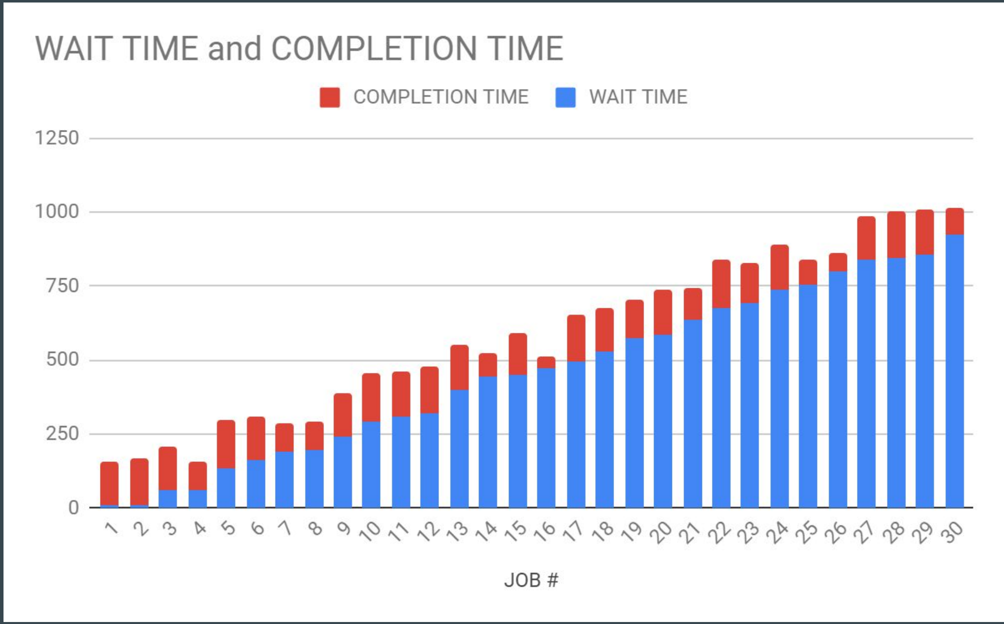
# Data - 4 CPU, FCFS Scheduling

Shown in this graph are the execution times for the simulation using: 4 CPU, FCFS scheduling policy

Wait times increase with job # as jobs are scheduled in a first come, first serve policy

Maximum RAM usage reaches 15%. Still low but much more utilization than with a single CPU as more programs are in memory concurrently

Much lower wait times as the workload is distributed



## Data - 4 CPU, Priority Scheduling

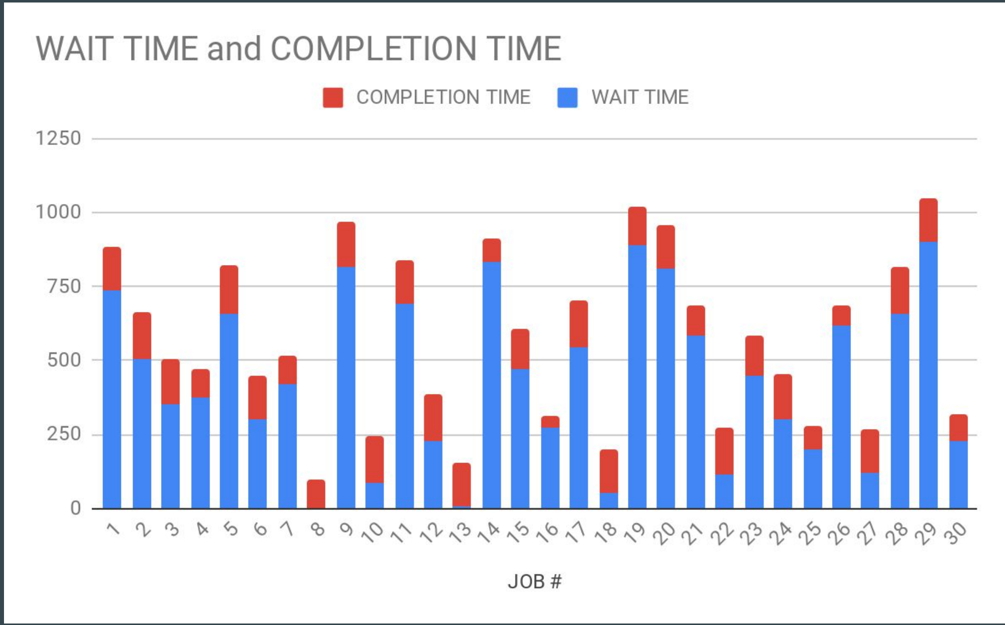
Shown in this graph are the execution times for the simulation using: 4 CPU, Priority scheduling policy

Jobs are selected in order of priority

No visible trends as program size varies

VM still utilizes a maximum of 16% RAM

Wait times much lower compared to 1 CPU



# Data - 4 CPU, SJF Scheduling

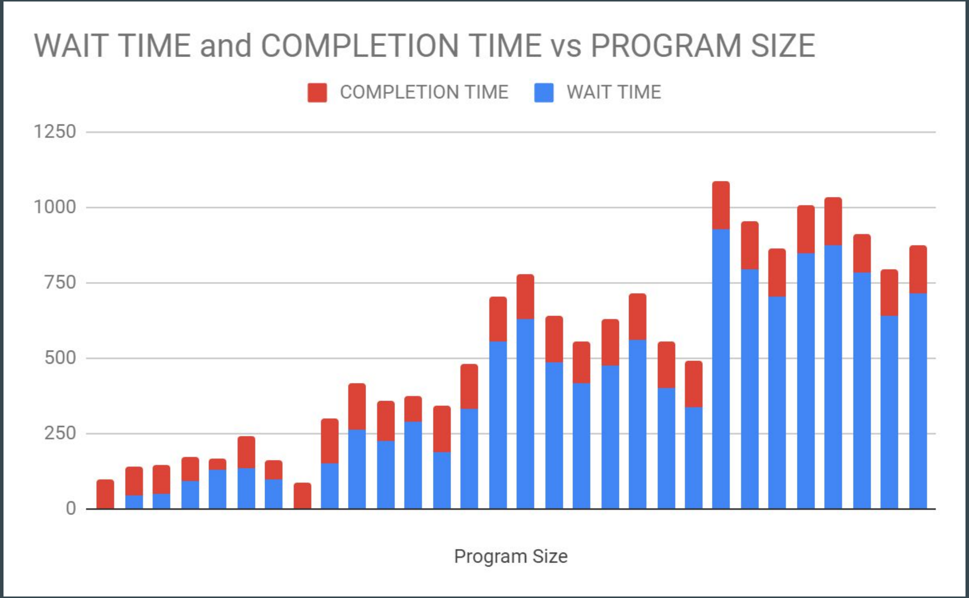
Shown in this graph are the execution times for the simulation using: 4 CPU, SJF scheduling policy

Programs are scheduled ordered by program size

Can see an upward trend in total execution time vs program size

Total execution times are not in perfect ascending order due to blocking and context switching

Much lower waiting times vs 1 CPU



## Live Demo

## Questions