Assignment 3

Submit Assignment

Due Monday by 10pm **Points** 75 **Submitting** a text entry box
Available Mar 2 at 8:15am - Mar 16 at 10pm 15 days

Assignment 3

Processes Scheduling

Task

Using C++, create a scheduling simulator that will implement various algorithms for scheduling processes on cores of a CPU. Pass the name of a configuration file to the simulator as a command line argument. For this project, you will not actually be spawning and scheduling processes on the CPU. Rather you will be creating simulated "processes" that consist of numerous CPU and I/O bursts.

The configuration file should specify the following:

- Line 1: Number of CPU cores (1-4)
- Line 2: Scheduling algorithm (RR: round-robin, FCFS: first come first serve, SJF: shortest job first, PP: preemptive priority)
- Line 3: Context switching overhead in milliseconds (100-1000)
- Line 4: Time slice in milliseconds (200 - 2000) - only used for round-robin algorithm, but include in configuration file regardless
- Line 5: Number of processes to run (1-24)
- Lines 6 - N: (one line per process - following values separated by commas)
  - PID (unique number >= 1024)
  - Start time (number of milliseconds after simulator starts that process should launch: 0 - 8000)
  - Number of CPU and I/O bursts (odd number: CPU bursts = I/O bursts + 1)
  - CPU and I/O burst times (sequence of milliseconds: 1000 - 6000 separated by vertical pipe)
    - Alternates between CPU and I/O (always starting and ending with CPU)
  - Priority (number: 0 - 4)

Example

```
2
SJF
400
750
```

```
5
1024,0,2500|1000|1250|1750|4000|2000|3500,2
1093,1750,6000|1500|4000,0
1054,0,1000|4250|1000|3750|1000,4
1025,5000,5000|1500|5000|1500|5000|1500|5000|1500|5000,1
1087,2400,1250|1000|2500|2000|3750,0
```

Use 2 CPU cores with the Shortest Job First scheduling algorithm. There will be a 400 ms overhead for context switching and a 750 ms time slice for round-robin scheduling (ignored since using SJF). This run will launch 5 processes - those with PIDs 1024 and 1054 will launch immediately, while process with PID 1093 will wait 1750 ms, process with PID 1087 will wait 2400 ms, and process with PID 1025 will wait 5000 ms before launching. The CPU and I/O burst times for each process are separated by a vertical pipe (|). The priority for each process is ignored since using SJF (and should be stored as 0 regardless of value in configuration file).

Implementation

Your simulator should apply the selected scheduling algorithm to simulate running the processes until all have reaching the terminated state. This simulator will represent a symmetric multiprocessing system that uses a shared ready queue. You should simulate each core as a thread that acts independent of the other cores in order to determine when to load or remove a "process". The main thread should keep track of process states (e.g. determine when an I/O burst finishes and the process is put back in the ready queue).

Throughout the execution of your simulator, you should print a table that shows the status of all "processes". The following provides sample output:

```
|   PID | Priority |      State | Core | Turn Time | Wait Time | CPU Time | Remain Time |
+-------+----------+------------+------+-----------+-----------+----------+-------------+
|  1024 |        0 |      ready |   -- |       7.2 |       3.0 |      3.0 |        11.4 |
|  1096 |        0 |    running |    0 |       7.2 |       2.2 |      2.5 |         4.4 |
|  1042 |        0 |        i/o |   -- |       6.0 |       0.8 |      0.8 |         9.6 |
|  1031 |        0 |      ready |   -- |       5.0 |       5.0 |      0.0 |        49.8 |
|  1082 |        0 | terminated |   -- |       4.9 |       1.0 |      2.9 |         0.0 |
|  1029 |        0 |    running |    1 |       4.2 |       1.4 |      2.4 |         6.1 |
```

- PID: unique identifier
- Priority: priority (0 if not using preemptive priority)
- State: ready, running, i/o, or finished
- Core: CPU core currently on (-- if not running)
- Turn Time: Total time since creation (until finished)
- Wait Time: Total time waiting in the ready queue
- CPU Time: Total time spent running on a CPU core
- Remain Time: CPU time remaining until terminated

After all "processes" finish, you should print the following statistics about the scheduler:

- CPU utilization
- Throughput
  - Average for first 50% of processes finished
  - Average for second 50% of processes finished
  - Overall average
- Average turnaround time
- Average waiting time

Scheduling Algorithms

Round-Robin:

- Add newly created "processes" to the end of the ready queue
- When a core becomes available, schedule the "process" at the front of the ready queue on that core
- If a "process's" time slice expires before the CPU burst is done, remove "process" from core and place at the end of the ready queue
- When a "process" finishes an I/O burst, place it at the end of the ready queue

First Come First Serve:

- Add newly created "processes" to the end of the ready queue
- When a core becomes available, schedule the "process" at the front of the ready queue on that core
- When a "process" finishes an I/O burst, place it at the end of the ready queue

Shortest Job First:

- Add newly created "processes" to the ready queue in a position that is based on their remaining CPU time
- When a core becomes available, schedule the "process" at the front of the ready queue (i.e. has the least amount of remaining CPU time) on that core
- When a "process" finishes an I/O burst, place it in the ready queue at a position that is based on its remaining CPU time

Preemptive Priority:

- Add newly created "processes" to the ready queue in a position that is based on priority number (0: highest prioity, 4: lowest priority)
  - If processes have the same priorty, then revert to first come first serve
- When a core becomes available, schedule the "process" at the front of the ready queue (i.e. has the highest priority) on that core
- When a "process" finishes an I/O burst, place it in the ready queue at a position that is based on its priority

- IMPORTANT NOTE: If a "process" is newly created or finishes an I/O burst and has a higher priority than a "process" currently running on a core, then the lowest priority "process" that is running should be preempted - removed from the CPU and placed in the ready queue

Instructions

You will be working in teams of 2-3 students to complete this project. You are allowed to work together on all aspects of the project or to divide tasks equally among the members. You are NOT however allowed to collaborate with students from other teams. The only exception to this is posting general questions/answers to the discussion board on Canvas.

Starter code is provided in **assignment03.zip**, which can be found on Canvas.

Code should be saved in a repository on GitHub with all team members as collaborators.

Submission

Each group member should independently submit the following in Canvas:

- Projects GitHub URL
- Compilation and running instructions
- List of what each team member contributed to the project

Deadline

This assignment is due Monday, March 16 at 10:00pm.

Teams

Team A	**Team B**	**Team C**	**Team D**
• Josh • Bernadette • John	• Tyler • Jesse	• Paige • Abby	• Jack • Chase
Team E	**Team F**	**Team G**	**Team H**
• Nick H. • Keith	• Matt • Devin	• Alex • Brandon	• Nicholas P. • Yuki