CS 405 - Fall 2023
Project 01: CPU Scheduling Simulator

History:

Date	Descriptions
Aug 22, 2023	First released - Project available

1. Objectives

This teamwork project is designed to simulate the short-term scheduler in a multiprocessor operating system. You need to form a team of a maximum of 3 people. Then, your team needs to develop a program to implement a simulator of CPU scheduling with different scheduling algorithms.

2. Overview

Your team needs to design and implement a program (in any language, C/C++, Java, Python, etc.) that simulates the CPU scheduling of multiple processes on a multiprocessor operating system.

2.1 Basic Process Scheduling

We have various processes waiting in queues for many resources, in which the CPU is at the top of the list. One of the most important problems here is scheduling, where we need to arrange access to the CPU based on the needs of the processes, minimize the waiting time, and reduce the system idle for resource utilization. Here, we focus on the short-term scheduler, which chooses the process from the ready queue to run when the CPU is idle.

Processes can contain `CPU burst,` and `IO burst` alternatively. While the CPU burst is the time the process needs the CPU to complete its operations, the IO burst is the time it does not need the CPU as it is waiting for one or slower I/O devices. Moreover, we classify the process as `CPU bound` if it mostly requires CPU rather than I/O and `I/O bound` if it mostly waits for I/O.

The scheduling happens in one of the following cases:

- The process changes its state from RUNNING to WAITING: because of system interrupt, wait for I/O before continuing, wait for some conditions (such as synchronization objects such as mutex, semaphores, etc.)
- The process changes its state from RUNNING to READY: because of the result of preemption scheduling - its time slice expires (i.e., in the time-sharing scheduling approach, each process only occupies the CPU for a fixed amount of time)
- The process changes its state from WAITING to READY: I/O devices are ready for the process to be used.
- The process terminates: OS will collect resources assigned to this process, remove the process from any queues, and pick up the next process to be executed by the CPU.

We need to implement an efficient scheduler that can minimize the waiting time and maximize CPU utilization. Depending on the system performance factors and the trade-off between scheduling algorithm complexity and system performance, different scheduling algorithms can be considered and selected appropriately.

2.2 Scheduling Algorithms

Implement and compare the performance of two scheduling algorithms below:

1. `Priority scheduling (PS)`: As described in slide 5.24 of lecture 5, in this scheduling algorithm, the processes with the highest priorities always get the CPU. Lower-priority processes may be preempted if a process with a higher priority becomes runnable.
2. `Priority Scheduling w/ Round-Robin (PSwRR)`: As described in slide 5.26 of lecture 5, this scheduling algorithm is based on the integration of round-robin and priority scheduling algorithm. From the ready queue, the scheduler always picks up the process with the highest priority to run. If there is more than one process with the same priority, run them in a round-robin fashion.

2.3 Performance metrics

The performance of a scheduler can be assessed by the following metrics:

- `CPU utilization`: what percentage of time the CPU is not idle. A good scheduler keeps the CPU busy. In particular, we want to space processes out so they aren't all trying to do I/O at once.
- `Throughput`: how many processes are completed per time unit. (Depends on what the processes are doing).
- `Turnaround time`: how long it takes a particular process to finish.
- `Waiting time`: total time spent by all processes in the ready queue.
- `Response time`: the time between user input and corresponding system action.

Response time likely will be the sole criterion for real-time processes and the main criterion for interactive processes (and possibly system processes that they depend on). Batch processes are likely to care more about turnaround time. The other measures (which tend to be closely related) are most useful to let you know if your scheduler is screwing up somehow, but the waiting time can vary dramatically depending on scheduling policy and is the factor that most closely affects response time.

3. Requirement Specs

3.1 General requirements

- This is a teamwork project in which you need to form a team of a maximum of 3 persons.
- You can implement the simulation program in any language of your choice (C/C++, Java, Python)
- This program is not a system program, so there is no need to create processes, interrupt handling, I/O handling, etc.
- Your program needs to use some form of "*visualization*" of the following components:
  - A CPU (could be multiple CPUs if you go for extra credits)
  - A ready queue that shows a list of processes waiting for their turns to use the CPUs
  - An I/O device
  - An I/O queue for each I/O device that shows a list of processes waiting to use the I/O device

3.2 Input Format

3.2.1 Program Input:

When your program starts, it receives the following required parameters:

- Simulation mode: can be 0 for auto or 1 for manual. In the auto mode, when your program starts, it will update the simulation step automatically after each simulation unit time until finishing the simulation (when all processes finish). In the manual mode, the user needs to interact with your program to display the next simulation step.
- Simulation unit time (ms): the time between two simulation steps to be displayed in auto mode. You can have options like frame per second (how many simulation steps in a second).
- Quantum (time slice): This parameter is only used in round-robin scheduling as each process can keep the CPU for this maximum amount of time and then preempt (give away) the CPU for the next process

3.2.2 Scenario

Also, your program can load a simulation scenario from a file to be executed by your program. The scenario file simply stores the required information of the processes needed to be scheduled to run on the CPU. Each process will be stored in one line in the scenario file. The information on each process is in the following format:

<name> <s> <p> <C_0> <I_0> <C_1> <I_1> … <C_{n-1}> <I_{n-1}> <C_n>

- **<name>**: the name of the process (string without space in the middle)
- **<s>**: the arrival time of the process (in terms of system unit time)
- **<p>**: the priority level of the process (0-9), in which 0 means the highest priority, and 9 is the slowest one. This one is used in the priority scheduling algorithm.
- **<C_i>**: the i^{th} CPU burst time
- **<I_i>**: the i^{th} IO burst time

Note: All processes start and end with CPU bursts. Interleaved among those CPU burst times are IO burst times. Except for the process name is a String, all other fields are integers.

Example:

Process information	Description
apache 3 2 5 4 8 2 9	This "apache" process appears at system time step 3 and has the priority of 2. It needs the first 5 unit time of CPU, then 4 unit time of I/O, then 8 unit time of CPU, then 2 unit time of I/O, and finally 9 unit time of CPU before termination.
proc1 10 4 25	This proc1 process appears at the system time step 10 and has the priority of 4. This process is an example of CPU bound process, which only needs 25 units of time of CPU and does not need I/O

3.3 Process

You need to define a class to represent the process control block (PCB) in your simulation environment. Similar to the PCB mentioned in lecture 03, you may need to store the following information for each process:

- **PID**: the process id, which can be the process index in the scenario file
- **State**: the current state of the process. A process can be in the following state:
    - **NEW**: The process has just been created, not yet READY to be executed.
    - **READY**: The process is ready to be executed by the CPU but is currently in the ready queue
    - **RUNNING**: The process is currently executed by the CPU

    - **WAITING**: The process is in the I/O queue to wait for the I/O device
    - **TERMINATED**: The process has already finished its tasks
- **Arrival time**: the system time that the process is created
- **Finish time**: the system time that the process terminates
- **Turnaround time**: the total execution time of a process from start to finish
- **Wait time**: the total time the process needs to wait in the ready queue
- **IO wait time**: the total time the process needs to wait in the I/O queue

Additional information from the scenario file needs to be stored, such as priority and the list of CPU burst times and IO burst times. The current execution state of the process needs to be saved - you need to mark which CPU burst / IO burst has been done.

3.4 Scheduler

You need to implement a scheduler component that implements CPU scheduling algorithms mentioned in section 2.2.

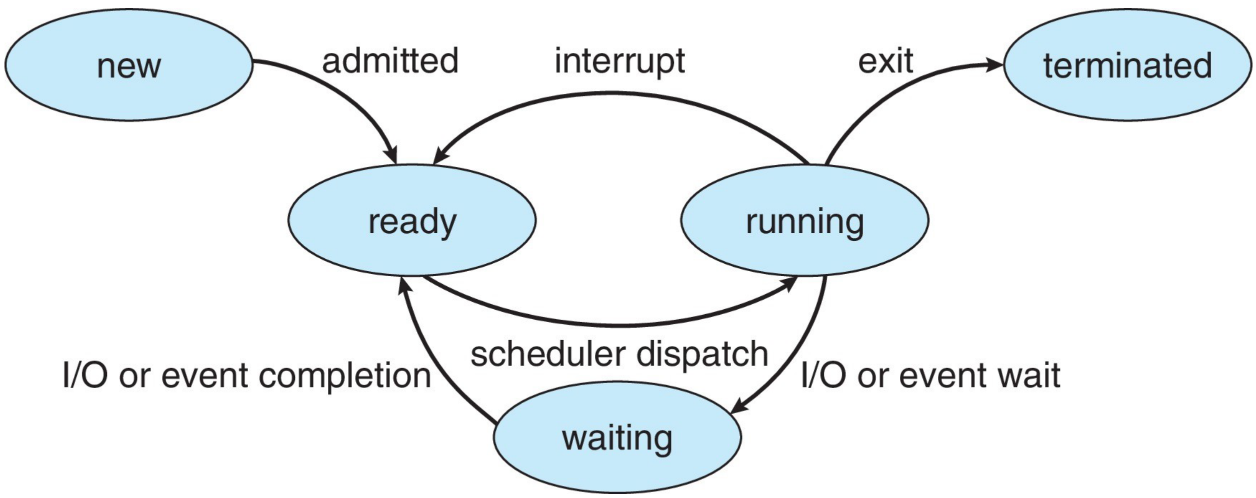


Fig 1: The scheduling state diagram (from "Operating System Concept 10th edition of Silberschatz, Galvin, and Gagne).

The processes are controlled by your scheduler following the previous image.

- A process p_i is created with the state NEW and changed to READY state at the system time s_i (the arrival time of p_i - check 3.2.2).
- Depending on the actual scheduling algorithms, if the CPU is idle, your scheduler will dispatch the next process from the ready queue to execute. Such a process changes its state to RUNNING.
- In the round-robin scheduling algorithm, if the CPU burst of the currently running process is greater than the quantum, such process will be interrupted and moved back to the ready queue (its state changes to READY).
- While the process is running, if its CPU burst is completed, it loads the next I/O burst. It will be placed on the I/O queue and changed the state to WAITING.
- When the I/O device is free, your scheduler will select the first process in the I/O queue to use the I/O device. This process will keep the I/O device for its current I/O

burst time. After finishing its I/O burst time, it will be moved back to the ready queue and changed to the READY state. The I/O device is ready to be used by the next process in the I/O queue.
- When the process is in a RUNNING state (use the CPU) and its current CPU burst is finished and is the last one, it terminates. Your scheduler will dispatch the next process to use the CPU.

3.5 Main program

You can develop your scheduling simulator program in **Console** mode or **GUI** mode.
- You can pass arguments to your program (check section 3.2.1) or implement GUI components for users to enter when your program starts.
- The scenario file can also be passed as an argument of your program or selected from GUI.
- The users can select the scheduling algorithm to run (FCFS, RR, SJF, or PS). They can use the same scenario and run with each scheduling algorithm. Each time the user changes the algorithm, the program resets the state and statistics.
- After each simulation completes, your program can ask the users to save the execution logs and system performance metrics of that run.

3.5.1 Visualization requirements

In either Console/GUI mode, you still need to "visualize" the four components: CPU, ready queue, I/O device, and I/O queue. In this project, you need to support 1 CPU and one I/O device.
- In the Console mode, you can use ASCII characters to draw the text visualization of such components, i.e., the following examples show the CPU 0 and I/O 0 are currently idle:

```
╭──────────────╮
│  CPU 0: idle │
╰──────────────╯

┌──────────────┐
│  I/O 0: idle │
└──────────────┘
```

- In the GUI mode, you can use Graphics 2D to draw visual presentations of those components.
- You can also visualize/simulate processes moving between those components during each simulation step.

3.5.2 Scheduling Simulation and Output

Your program should support the start/pause/resume of the simulation.
In each simulation step, besides displaying the current state of four components (CPU, I/O device, ready queue, and I/O queue), you are required to display each process

information (i.e., current state, list of remaining CPU burst, I/O burst, etc.). You are also required to display the system performance metrics listed in section 2.3, including CPU utilization, average turnaround time, average wait time, and throughput.
Also, you need to display a message on the screen and log the following events to the output file:
- when a process is newly created or put in the ready queue
- when a process is dispatched from the ready queue to use the CPU
- when a process is interrupted or put into an I/O queue
- when a process is preempted and put back to the ready queue
- when a process completes I/O and return to the ready queue
- when a process terminates (after finishing its tasks). In this case, you need to print the turnaround time, total CPU wait time and I/O wait time of this process.

3.6 Testing and Reporting the Results

- After completing the implementation of your program, you may need to test your program with different scenarios.
- Each scenario will need to be scheduled with each algorithm, and the corresponding system performance metrics will be collected.
- Finally, you need to draw charts to compare the performance of these scheduling algorithms. Also, **write a report to discuss your results**.

3.7 Extra Credits

Your team can earn extra credits for this project if you can implement the following extra work:
- (5 pts) Implement successfully an additional scheduling algorithm, either SJF or RR.
- (10 pts) Support multiprocessor and/or multiple I/O devices: your program can schedule to run processes on more than one CPU, and there is more than one I/O device.
- (10 pts) Support multi-thread: You can implement a main thread for the main program, a children thread for each CPU, and a thread for the scheduler component.

4. Project Deliverables

You need to submit the following items:
1. Compress the source code of your project, test data (scenario files), test outputs, and comparison results.
2. A DOCX/PDF report (less than ten pages) to summarize the implementation details of your team, the performance comparison and discussion (check the requirements in section 3.6), and the individual contributions of each member.
Submit the two above items to D2L.

5. Grading Rubrics

5.1 Program design and implementation (70 pts)

- 20 pts: Correct implementation of the main simulator program that takes input arguments, load scenario file, start/pause/stop the simulation, and save/log simulation results
- 20 pts: Correct implementation of the four scheduling algorithms
- 20 pts: Correct visualization of scheduling algorithms and display execution events
- 10 pts: Good data structures to represent the process, CPU, queue, etc., and good coding style and comments

5.2 Report (30 pts)

- 10 pts: Present the comparison results (charts, images, etc.) of your simulation between different scheduling algorithms
- 10 pts: Discuss your findings of the comparison and lessons
- 10 pts: Report how your team works together and individual contributions

5.3 Extra Credits (maximum 25 pts if applicable)

Q&A:

Q1. "Wait IO Time" vs. "Response Time"?
A: Wait IO Time is not the same as Response Time: The first one is the total time a process needs to wait for I/O devices in the waiting queue. This metric is similar to Wait Time (the waiting time for a CPU in the ready queue). The response time measures the difference between the start time and arrival time of a process. Arrival time is the time the process is created and ready to be scheduled to run. However, although the process is created, it may need to wait until the start time, when it is first executed by the CPU.

Q2: How to incorporate the 2D graphics into the swing window?
A: You can override any Swing controls to perform Graphics 2D drawing operations. One of the common approaches is subclassing (extending) a JPanel component, overriding the public void paint (Graphics g) method to perform custom drawing.
Here is a tutorial on Graphics 2D from Java Oracle:
https://docs.oracle.com/javase/tutorial/2d/TOC.html

Q3: Should average wait time include waiting in an I/O queue, or only waiting in CPU? queue?
A: The average wait time is calculated from the waiting time for the CPU (in the ready queue) of all processes. There is a waiting time for IO, but I did not show it in my simulator program.

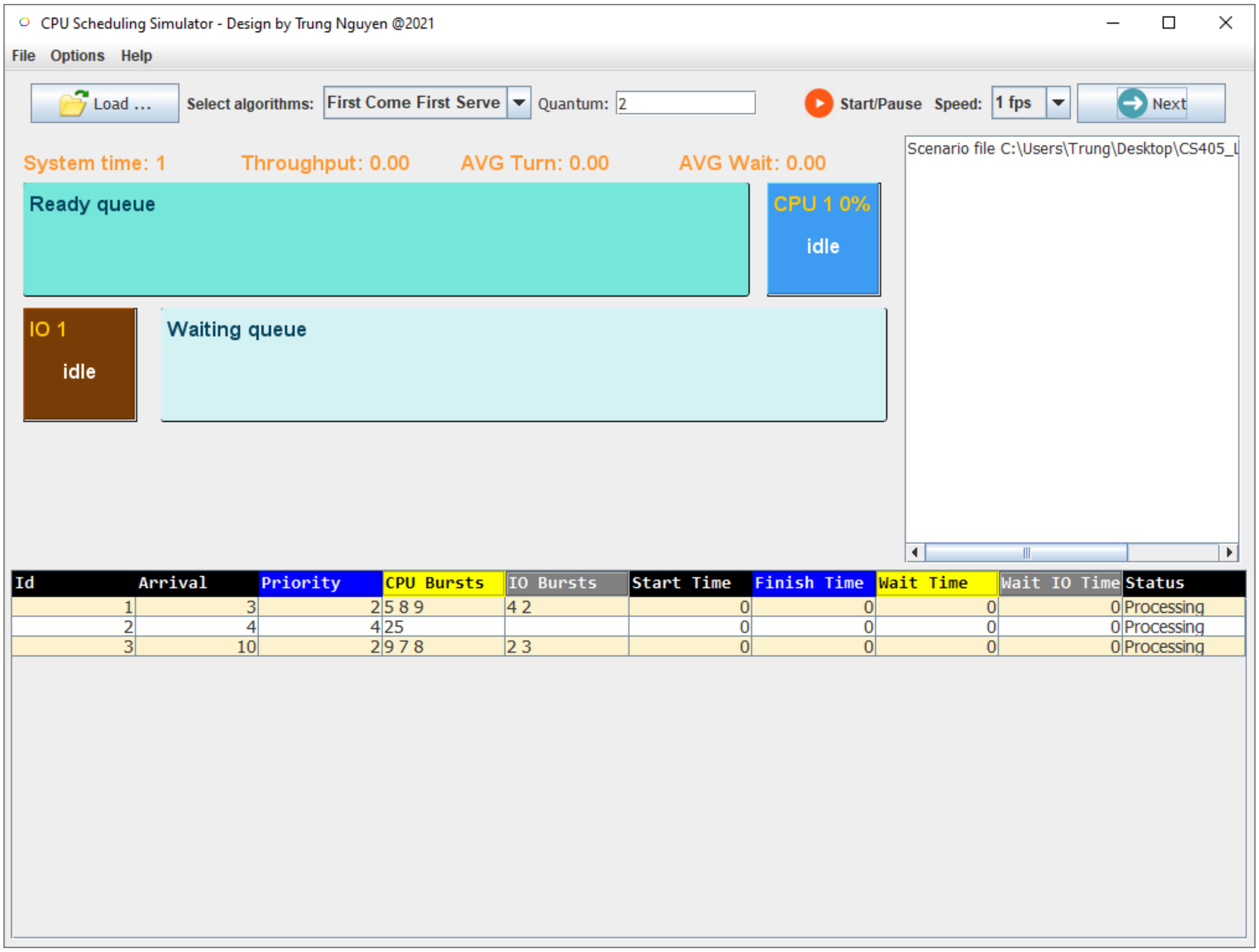
Q4: How to measure average wait time?
A: Waiting time = (total wait time of all processes until now) / (number of processes). For each process, there should be a data field that represents the total waiting time for the CPU. When the process is created, you should initialize its waiting time = 0, then, each time it is in the ready queue, increase its waiting time until the process finishes. Then, its waiting time will stay the same (no change after the process terminates).

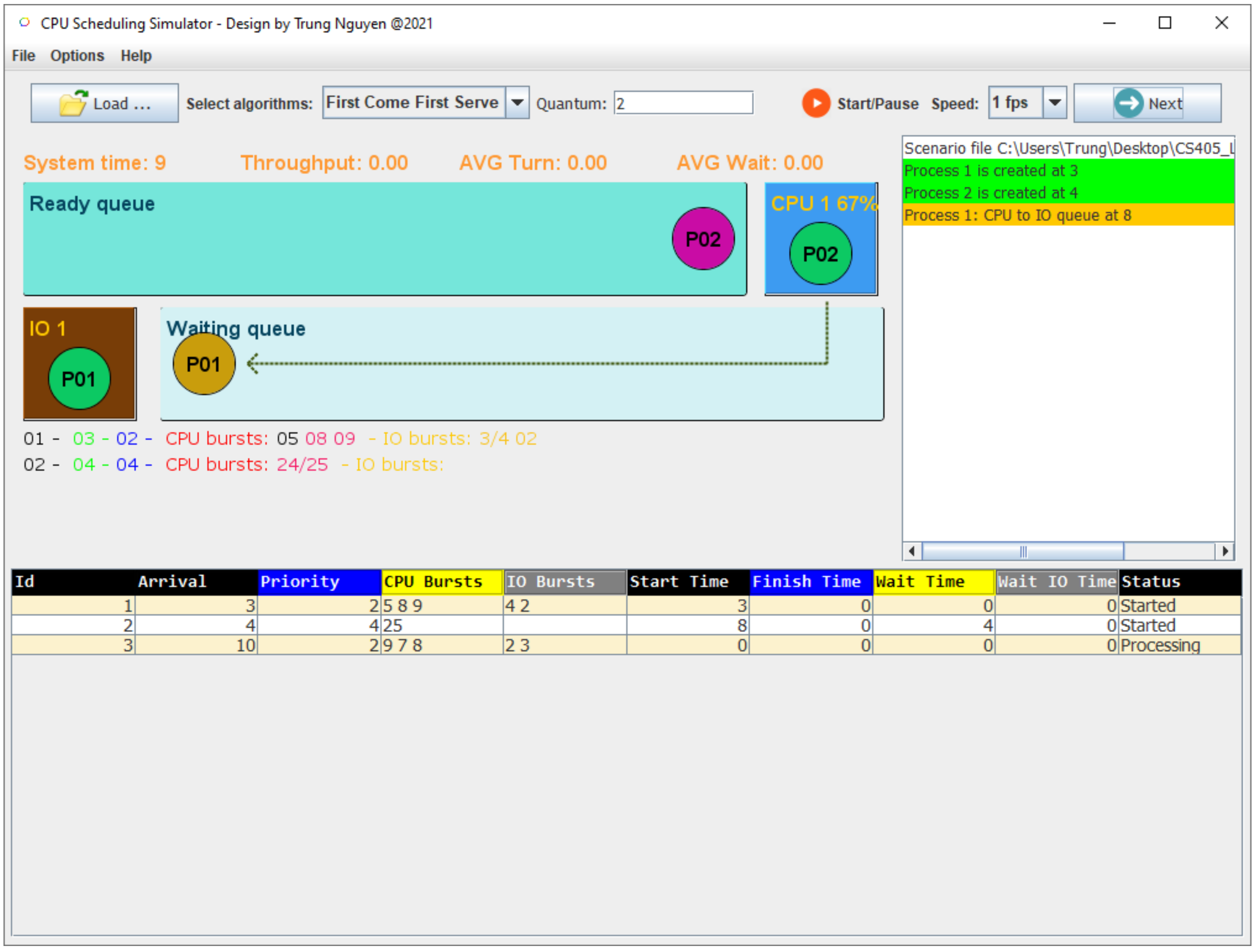
Q5: Maintain scheduling consistency and integrity.
A: Two requirements have to be maintained in your program:
- Make sure a process can be in one queue at a time (either ready queue or IO waiting queue, not both at the same time)
- In the PCB, there should be an indicator of which current burst is - CPU or IO burst, and its progress, i.e., proc1 curBurst = 2/4 (which means proc1 is currently on CPU burst, still need to use CPU 2 unit time out of total 4). When its progress is 0, switch to the next IO burst if available or terminate. (Remember, CPU bursts are the first and the last bursts in the process, and interleaves of those CPU bursts are IO bursts).

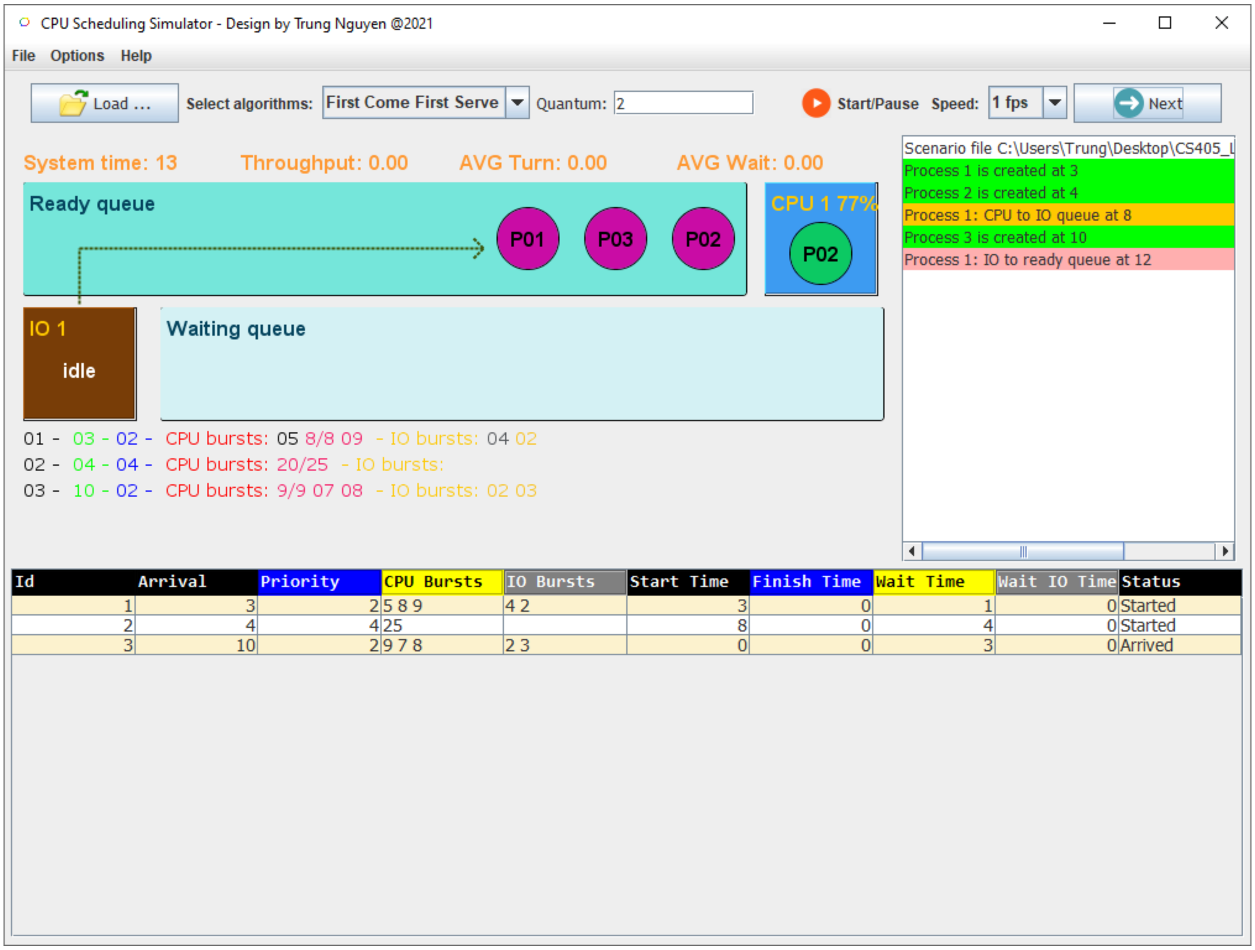
Q6: How to pick up the next process in SJF?
A: In Shortest Job First, you choose the process with the smallest next CPU burst to take the CPU.

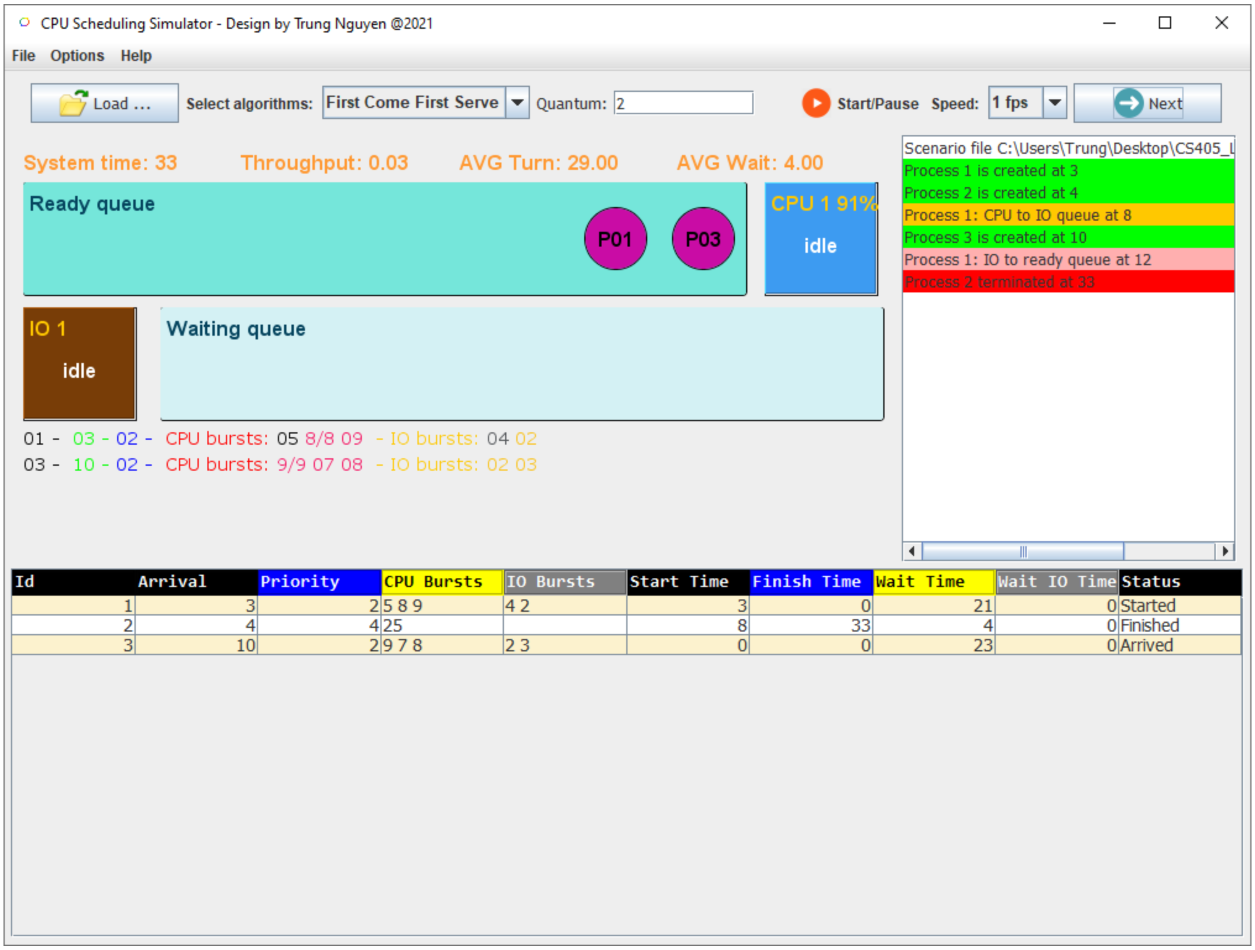
Q7: How to schedule the I/O?
A: For scheduling I/O, we always use FCFS (the first process in the IO queue takes control of the I/O device to perform IO). It does not use scheduling algorithms like the CPU.

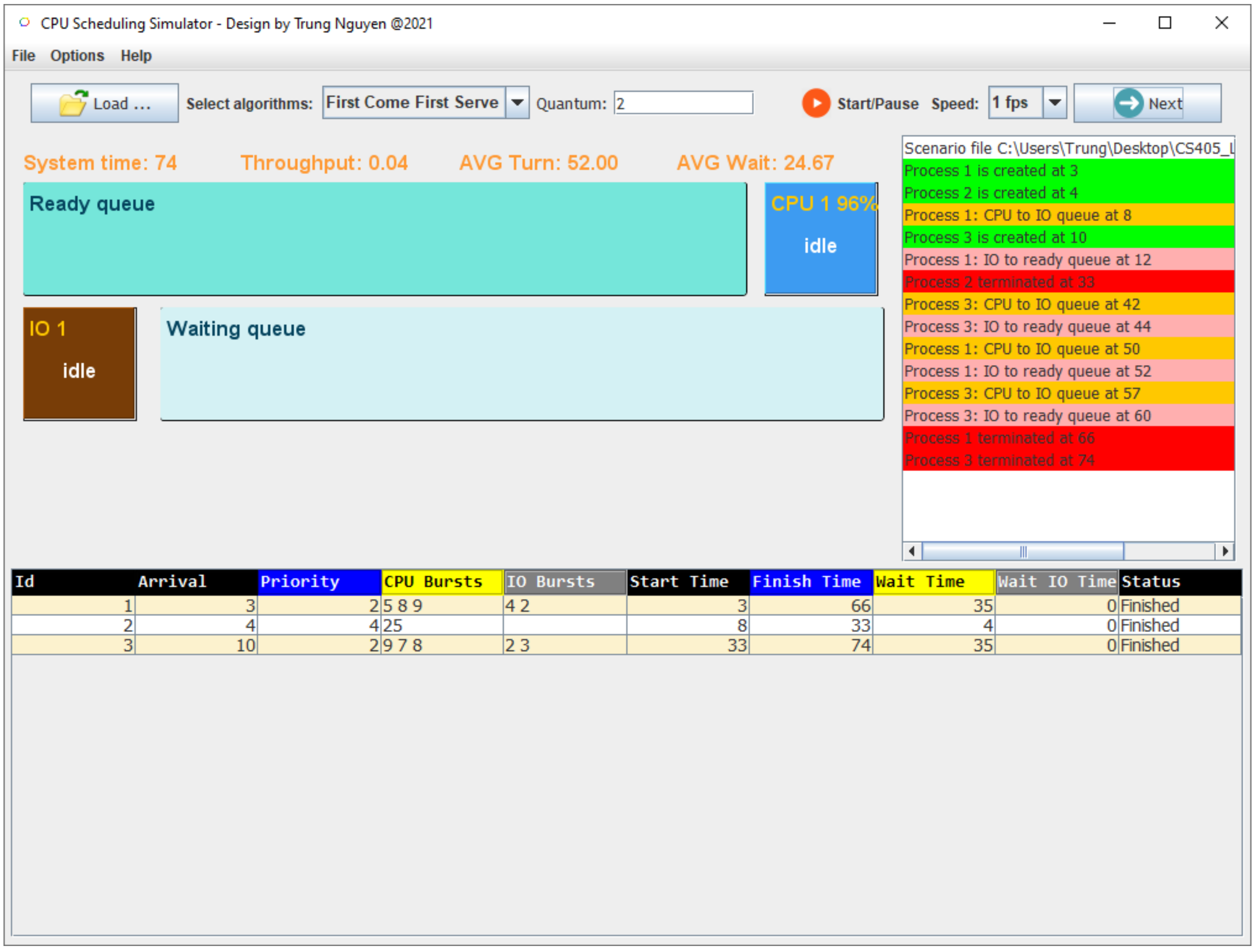
Sample screenshots of the demonstration program:




CPU Scheduling Simulator - Design by Trung Nguyen @2021
File Options Help
Load ...
Select algorithms: First Come First Serve
Quantum: 2
Start/Pause Speed: 1 fps
Next
System time: 9
Throughput: 0.00
AVG Turn: 0.00
AVG Wait: 0.00
Ready queue
P02
CPU 1 67%
P02
IO 1
P01
Waiting queue
P01
01 - 03 - 02 - CPU bursts: 05 08 09 - IO bursts: 3/4 02
02 - 04 - 04 - CPU bursts: 24/25 - IO bursts:
Scenario file C:\Users\Trung\Desktop\CS405_
Process 1 is created at 3
Process 2 is created at 4
Process 1: CPU to IO queue at 8
Id Arrival Priority CPU Bursts IO Bursts Start Time Finish Time Wait Time Wait IO Time Status
1 3 2 5 8 9 4 2 3 0 0 0 Started
2 4 4 25 8 0 4 0 Started
3 10 2 9 7 8 2 3 0 0 0 0 Processing


CPU Scheduling Simulator - Design by Trung Nguyen @2021
File Options Help
Load ...
Select algorithms: First Come First Serve
Quantum: 2
Start/Pause Speed: 1 fps
Next
System time: 13
Throughput: 0.00
AVG Turn: 0.00
AVG Wait: 0.00
Ready queue
P01 P03 P02
CPU 1 77%
P02
IO 1
idle
Waiting queue
01 - 03 - 02 - CPU bursts: 05 8/8 09 - IO bursts: 04 02
02 - 04 - 04 - CPU bursts: 20/25 - IO bursts:
03 - 10 - 02 - CPU bursts: 9/9 07 08 - IO bursts: 02 03
Scenario file C:\Users\Trung\Desktop\CS405_
Process 1 is created at 3
Process 2 is created at 4
Process 1: CPU to IO queue at 8
Process 3 is created at 10
Process 1: IO to ready queue at 12
Id Arrival Priority CPU Bursts IO Bursts Start Time Finish Time Wait Time Wait IO Time Status
1 3 2 5 8 9 4 2 3 0 1 0 Started
2 4 4 25 8 0 4 0 Started
3 10 2 9 7 8 2 3 0 0 3 0 Arrived


CPU Scheduling Simulator - Design by Trung Nguyen @2021
File Options Help
Load ...
Select algorithms: First Come First Serve
Quantum: 2
Start/Pause Speed: 1 fps
Next
System time: 33
Throughput: 0.03
AVG Turn: 29.00
AVG Wait: 4.00
Ready queue
P01
P03
CPU 1 91%
idle
IO 1
idle
Waiting queue
01 - 03 - 02 - CPU bursts: 05 8/8 09 - IO bursts: 04 02
03 - 10 - 02 - CPU bursts: 9/9 07 08 - IO bursts: 02 03
Scenario file C:\Users\Trung\Desktop\CS405_
Process 1 is created at 3
Process 2 is created at 4
Process 1: CPU to IO queue at 8
Process 3 is created at 10
Process 1: IO to ready queue at 12
Process 2 terminated at 33
Id
Arrival
Priority
CPU Bursts
IO Bursts
Start Time
Finish Time
Wait Time
Wait IO Time
Status
1 3 2 5 8 9 4 2 3 0 21 0 Started
2 4 4 25 8 33 4 0 Finished
3 10 2 9 7 8 2 3 0 0 23 0 Arrived


CPU Scheduling Simulator - Design by Trung Nguyen @2021
File Options Help
Load ...
Select algorithms: First Come First Serve
Quantum: 2
Start/Pause Speed: 1 fps
Next
System time: 74
Throughput: 0.04
AVG Turn: 52.00
AVG Wait: 24.67
Ready queue
CPU 1 96%
idle
IO 1
idle
Waiting queue
Scenario file C:\Users\Trung\Desktop\CS405_l
Process 1 is created at 3
Process 2 is created at 4
Process 1: CPU to IO queue at 8
Process 3 is created at 10
Process 1: IO to ready queue at 12
Process 2 terminated at 33
Process 3: CPU to IO queue at 42
Process 3: IO to ready queue at 44
Process 1: CPU to IO queue at 50
Process 1: IO to ready queue at 52
Process 3: CPU to IO queue at 57
Process 3: IO to ready queue at 60
Process 1 terminated at 66
Process 3 terminated at 74
Id Arrival Priority CPU Bursts IO Bursts Start Time Finish Time Wait Time Wait IO Time Status
1 3 2 5 8 9 4 2 3 66 35 0 Finished
2 4 4 25 8 33 4 0 Finished
3 10 2 9 7 8 2 3 33 74 35 0 Finished