Scheduling Algorithms System

Submitted by:

Mohamed Ahmed Rabea

Mostafa Wael Hussien

Ahmed Sayed Mohamed

Mohamed Yousef Helmy

Ali Abdelmenaim Mohamed

Submitted To:

Dr. Marwa Yousef

Eng. Mohamed Abdelmegid

Contents:

Overview

The **Process Scheduler Simulator for Multiprocessor Systems** is a comprehensive tool designed to emulate and evaluate different scheduling policies within a multiprocessor environment. The primary objective of this project is to offer a versatile platform that allows users to simulate, analyze, and compare the performance of distinct scheduling algorithms, including First-Come, First-Served (FCFS), Shortest Job First (SJF), Shortest Time-to-Completion First (STCF), Round Robin (RR), Priority, Multilevel Feedback Queue (MLFQ), and Stride scheduling.

Objectives:

1. **Scheduling Policy Evaluation:** The simulator facilitates the examination and comparison of diverse scheduling policies to comprehend their behavior in a multi-core processor setting.
2. **Performance Assessment:** It measures and reports crucial performance metrics like turnaround time.
3. **User Interaction and Customization:** The tool offers an intuitive user interface that enables users to interact with the system, input custom processes, modify parameters, and select scheduling policies for simulation.
4. **Visualization and Analysis:** Through Gantt charts, the simulator illustrates the execution and allocation of processes across processors, aiding in the assessment of scheduling policies' efficiency.

Significance:

Understanding the nuances of scheduling policies in a multiprocessor environment is pivotal in optimizing system performance, resource utilization, and responsiveness. This simulator provides an invaluable platform for students, researchers, and system designers to explore.

Scope

The **Process Scheduler Simulator for Multiprocessor Systems** encompasses a wide array of functionalities and capabilities, providing a comprehensive platform for simulating and analyzing various scheduling policies in a multiprocessor environment.

Scheduling Policies Covered:

1. **First-Come, First-Served (FCFS):** Processes are executed in the order they arrive.
2. **Shortest Job First (SJF):** Prioritizes the shortest burst time processes.
3. **Shortest Time-to-Completion First (STCF):** Selects processes based on remaining time to completion.
4. **Round Robin (RR):** Time-sliced scheduling for fair execution among processes.
5. **Priority Scheduling:** Assigns priority levels to processes for execution.
6. **Multilevel Feedback Queue (MLFQ):** Implements multiple queues with varied priorities.
7. **Stride Scheduling:** Utilizes proportional allocation of CPU time based on stride values.

Types of Simulated Processes:

The simulator accommodates a diverse range of processes, comprising both CPU and I/O-bound tasks. Each process is defined by a series of CPU bursts and I/O bursts, mimicking real-world workloads.

User Interaction Capabilities:

Users have an interactive interface that allows for:

- Inputting custom processes with specific burst times and characteristics.
- Modifying system parameters, like quantum size (for RR), Tickets (for Stride).
- Selecting and initiating simulations for different scheduling policies.
- Visualizing results through Gantt charts.

System Architecture

The **Process Scheduler Simulator** exhibits a modular architecture comprising crucial components for emulating a multiprocessor system and implementing various scheduling policies.

Components:

1. **Process Representation:**
    - Defined as a structure encompassing essential attributes like process name, arrival time, phases (execution or I/O), priority, state, tickets, and performance metrics.
    - Employs a vector-based representation for each process, allowing flexible management and manipulation during simulation.
2. **Scheduler Module:**
    - Core component responsible for executing the Scheduling policies.
    - Houses algorithms and logic for scheduling decisions based on the chosen policy.
    - Integrates with process queues, handles context switching, and manages process states.
3. **Multiprocessor Environment:**
    - Simulates a four-processor system, utilizing an array-based representation to manage processor states and process assignments.
    - Facilitates parallel execution of processes based on availability and task requirements.
4. **User Interface:**
    - Interaction primarily via the command-line interface for simplicity.
    - Provides user prompts to input the number of processes and generates sample processes with randomized attributes within a file and the system takes the input from the file itself.
    - Initiates the Scheduler chosen policy and displays the output, including processor-wise execution snapshots.

High-Level Workflow:

1. **Initialization and Input:**
    - Accepts user input for the number of processes or generates sample processes with randomized attributes from the file.
    - Initializes process structures and computes the necessary values for each process based on the provided phases and arrival times.
2. **Scheduler Execution:**
    - Utilizes a loop-based mechanism to simulate processor allocation and execution of processes based on Scheduling.
    - Manages CPU bursts, I/O phases, context switches, and process completions.
3. **Output and Analysis:**
    - Generates processor-wise execution snapshots using a two-dimensional vector (**output**) to represent the progress of each process over time.
    - Calculates process statistics such as turnaround time and completion time for each process.

Scalability and Extension:

The current architecture is tailored specifically for the Scheduling policies within a four-processor environment. Extending this simulator to accommodate additional scheduling policies or processors would require structural modifications and enhanced user interaction capabilities.

Data Structures

1. **Process Structure (struct process):**
    - Attributes: **process_name**, **arrive_time**, **turn_around_time**, **complete_time**, **time_consumed**, **n_phases**, **phase_idx**, **last_processor**, **phases**, **state**, **priority**, **current_brust_time**, **pass_value**, **stride**, **tickets**, **qunta**.
    - Represents an individual process with its specific characteristics, including name, arrival time, phases, state, and performance metrics.
    - Utilizes a vector to store phase details (execution time and type - CPU burst or I/O burst).
2. **Output Matrix (vector<vector<string>> output):**
    - A two-dimensional vector used to visualize the execution progress of processes on each processor over time.
    - Provides a snapshot of process execution in a tabular format, aiding in creating Gantt charts or processor-wise execution logs.
3. **Scheduler Data Structures:**
    - **queue<int>** for handling waiting processes during execution.
    - Arrays (**processor[]**, **WillGoToTheQ[]**) to manage process allocation to processors based on the Stride Scheduling policy.

Algorithms Implementation

1. **Process Initialization and Input Generation:**
    - **Mechanism:** Generates or accepts input for the number of processes, their attributes, and phases from a file.
    - **Working Principle:**
        - Generates sample processes with randomized attributes for testing and simulation purposes.
        - Accepts user input for process attributes like arrival time, phases (CPU bursts or I/O), and other relevant parameters.
        - Computes necessary values for each process based on provided phase details for each scheduling policy.
    - **Integration into Simulator:**
        - Initializes process structures and sets up the simulation environment with sample or user-defined input from a file.
        - Enables the execution of scheduling algorithms by creating a pool of processes for scheduling.

2. SJF Algorithm Implementation

1. **Initialization:**
    - Sorts the processes based on their burst times using the **comparedByBurst** comparison function.
    - Initializes variables like **finished_processes**, **processor[]**, **WillGoToTheQ[]**, and a priority queue **waiting** to manage waiting processes.
2. **Scheduler Execution:**
    - Loops through each time unit to simulate the scheduler's behavior.
    - Checks for arriving processes at each time unit and allocates available processors to the shortest job among the arrived processes.
    - Queues processes that arrive but cannot be immediately executed due to all processors being busy.

- Executes processes on the processors based on their burst times, allowing the shortest job to run on the available processor.
- Tracks the execution progress in the **output** matrix for visual representation.
- Handles completion of phases, updating states, and marking processes as finished when all phases are executed.

3. **Processor Allocation and Execution:**
    - Assigns available processors to the shortest job among the arrived processes.
    - Manages the execution of processes based on their burst times, ensuring completion of phases, and marking processes as finished accordingly.
4. **IO_handler_bypq Function:**
    - Handles I/O operations for processes in the priority queue **waiting**, similar to the IO handler function used in other scheduling policies.
5. **Completion Check:**
    - Checks if all processes have finished execution. If so, generates the final output and exits the simulation.

3. FCFS Algorithm Implementation

1. **Initialization:**
    - Initializes variables such as **finished_processes**, **processor[]**, **WillGoToTheQ[]**, and a queue **waiting** to manage waiting processes.
2. **Scheduler Execution:**
    - Loops through each time unit to simulate the scheduler's behavior.
    - Checks for arriving processes at each time unit and allocates available processors to the arriving processes based on the FCFS principle.
    - Queues processes that arrive but cannot be immediately executed due to all processors being busy.

3. **Processor Allocation and Execution:**
    - Allocates processors to arriving processes in the order of their arrival.
    - Manages the execution of processes on processors, allowing them to execute their phases according to their burst times.
    - Tracks the execution progress in the **output** matrix for visual representation.
4. **IO_handler Function:**
    - Handles I/O operations for processes in the waiting queue, similar to other scheduling policies.
5. **Completion Check:**
    - Checks if all processes have finished execution. If so, generates the final output and exits the simulation.

4. STCF (Shortest Time-to-Completion First) Algorithm Implementation

1. **Initialization:**
    - Initializes variables like **finished_processes**, **processor[]**, **WillGoToTheQ[]**, and a priority queue **waiting** to manage waiting processes based on their burst times.
2. **Scheduler Execution:**
    - Loops through each time unit to simulate the scheduler's behavior.
    - Checks for arriving processes at each time unit and allocates available processors based on the STCF principle, favoring processes with shorter burst times.
3. **Processor Allocation and Execution:**
    - Allocates processors to arriving processes based on their burst times.
    - Manages the execution of processes on processors, allowing them to execute their phases according to their burst times.
    - Tracks the execution progress in the **output** matrix for visualization.

4. **IO_handler_bypq Function:**
    - Handles I/O operations for processes in the waiting queue, similar to other scheduling policies.
5. **Completion Check:**
    - Checks if all processes have finished execution. If so, generates the final output and exits the simulation.

5. Round Robin Algorithm Implementation

1. **Initialization:**
    - Initializes variables like **my_queue** (queue to store processes), **finished_processes**, **processor[]**, **WillGoToTheQ[]**, and **time_slice** (quantum time) for process execution.
2. **Scheduler Execution:**
    - Simulates the scheduler's behavior by iterating over each time unit.
    - Checks for arriving processes and assigns them to available processors or enqueues them in the **my_queue**.
3. **Processor Allocation and Execution:**
    - Allocates processors to arriving processes or those waiting in the queue.
    - Executes processes for a fixed time slice (**time_slice**) or until the process finishes its current burst time, whichever occurs first.
    - Tracks the progress of process execution in the **output** matrix for visualization.
4. **Match Preferences and IO Handling:**
    - Matches preferences for processor allocation.
    - Handles I/O operations for processes.
5. **Completion Check:**
    - Checks if all processes have finished execution. If so, generates the final output and exits the simulation.

6. Multi-Level Feedback Queue (MLFQ) Algorithm Implementation:

1. **Initialization:**
    - Initializes variables like **my_queue[]** (queues for multiple priority levels), **finised_processes**, **processor[]**, **WillGoToTheQ[]**, **time_slice**, and other related parameters.
    - Resets the processors and queues.
2. **Scheduler Execution:**
    - Iterates over each time unit.
    - Assigns arriving processes to available processors or to appropriate queues based on priority levels (**my_queue**).
3. **Processor Allocation and Execution:**
    - Executes processes in processors based on their priority levels.
    - Tracks the progress of process execution in the **output** matrix for visualization.
    - Handles time quantum expiry and phase completion for processes, moving them to the appropriate queue or updating their priorities.
4. **Queue Management and Priority Handling:**
    - Manages queues by shifting processes between different priority queues based on their quantum usage and priorities.
    - Handles processor allocation considering process priorities and available queues.
5. **Match Preferences and IO Handling:**
    - Matches preferences for processor allocation.
    - Handles I/O operations for processes.
6. **Priority Adjustment:**
    - Adjusts the priority of processes after a certain number of time units (**time_to_moveup**), likely to maintain fairness or adjust scheduling parameters dynamically.

7. **Completion Check:**
    - Checks if all processes have finished execution. If so, generates the final output and exits the simulation.

7. Stride Scheduling Algorithm Implementation:

1. **Initialization:**
    - Initializes variables such as **waiting**, **q**, **finished_processes**, **WillGoToTheQ**, **processor[]**, and **i**.
2. **Scheduler Execution:**
    - Executes a loop representing each time unit.
    - Sorts the processes based on their arrival time or their pass value and fills the queue accordingly.
    - Assigns arriving processes to available processors or to the waiting queue (**waiting**).
3. **Processor Allocation and Execution:**
    - Executes processes in processors based on their arrival times or pass values.
    - Tracks the progress of process execution in the **output** matrix for visualization.
    - Updates the pass value for each process based on its stride.
    - Handles completion of processes after their time slice or phase completion.
4. **Queue Management and Preference Handling:**
    - Manages the waiting queue and assigns processes to processors based on their states and availability.
    - Matches preferences for processor allocation.
5. **IO Handling and Completion Check:**
    - Handles I/O operations for processes.
    - Checks if all processes have completed execution. If so, generates the final output and exits the simulation.

6. **Completion Check:**
    - Checks if all processes have finished execution. If so, generates the final output and exits the simulation.

8. Priority Scheduling Algorithm:

1. **Initialization:**
    - Sorts the processes based on priority.
    - Initializes variables like **finished_processes**, **processor[]**, **WillGoToTheQ[]**, and queues.
2. **Scheduler Execution:**
    - Executes a loop representing each time unit.
    - Assigns arriving processes to available processors based on their priority.
    - Allocates processes that arrive when no processors are available to a waiting queue.
3. **Processor Allocation and Execution:**
    - Executes processes in processors based on their priority.
    - Tracks the progress of process execution in the **output** matrix for visualization.
    - Handles the completion of processes after their time slice or phase completion.
4. **Queue Management and Preference Handling:**
    - Manages the waiting queue and assigns processes to processors based on their states and availability.
    - Matches preferences for processor allocation using the **WillGoToTheQ** array.
5. **IO Handling and Completion Check:**
    - Handles I/O operations for processes.
    - Checks if all processes have completed execution. If so, generates the final output and exits the simulation.

Code Samples:

Round Robin

7

A 4 0

3 EXE

2 I/O

20 EXE

1 I/O

B 3 0

6 EXE

2 I/O

1 EXE

C 3 0

10 EXE

2 I/O

10 EXE

D 3 0

6 EXE

3 I/O

6 EXE

E 3 0

6 EXE

3 I/O

6 EXE

F 3 0

6 EXE

3 I/O

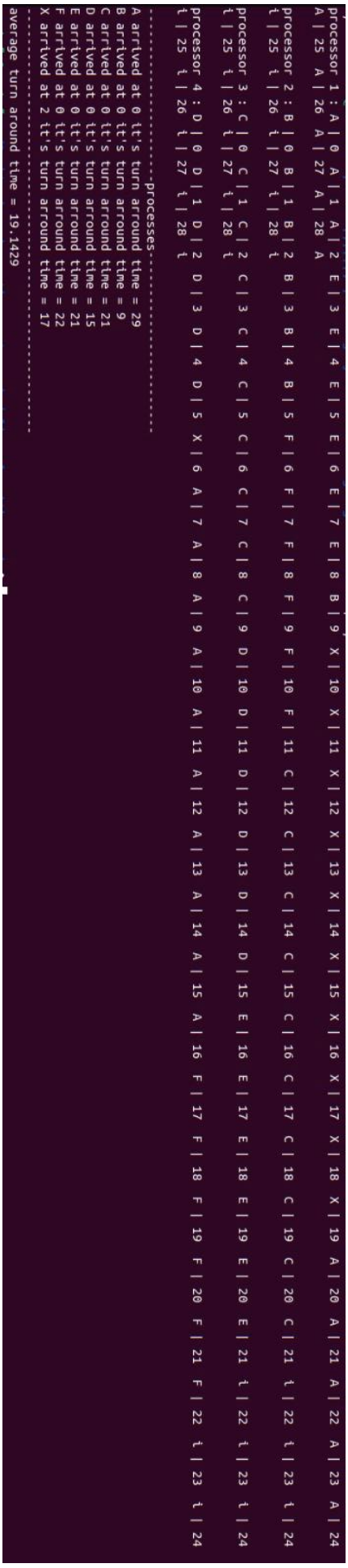
6 EXE

X 3 2

1 EXE

1 I/O

10 EXE



SJF

7

A 4 0

3 EXE

2 I/O

20 EXE

1 I/O

B 3 0

6 EXE

2 I/O

1 EXE

C 3 0

10 EXE

2 I/O

10 EXE

D 3 0

6 EXE

3 I/O

6 EXE

E 3 0

6 EXE

3 I/O

6 EXE

F 3 0

6 EXE

3 I/O

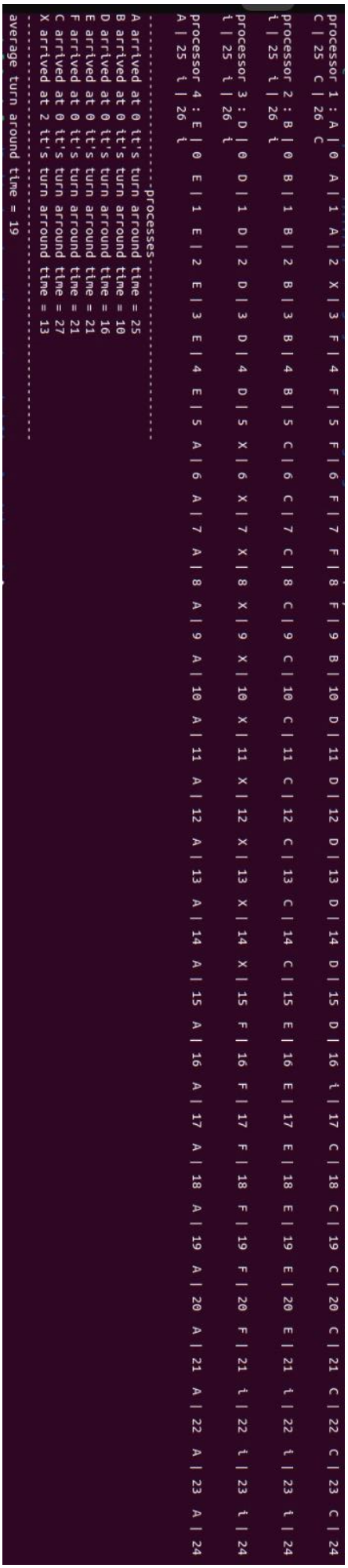
6 EXE

X 3 2

1 EXE

1 I/O

10 EXE



FCFS

7

A 4 0

3 EXE

2 I/O

20 EXE

1 I/O

B 3 0

6 EXE

2 I/O

1 EXE

C 3 0

10 EXE

2 I/O

10 EXE

D 3 0

6 EXE

3 I/O

6 EXE

E 3 0

6 EXE

3 I/O

6 EXE

F 3 0

6 EXE

3 I/O

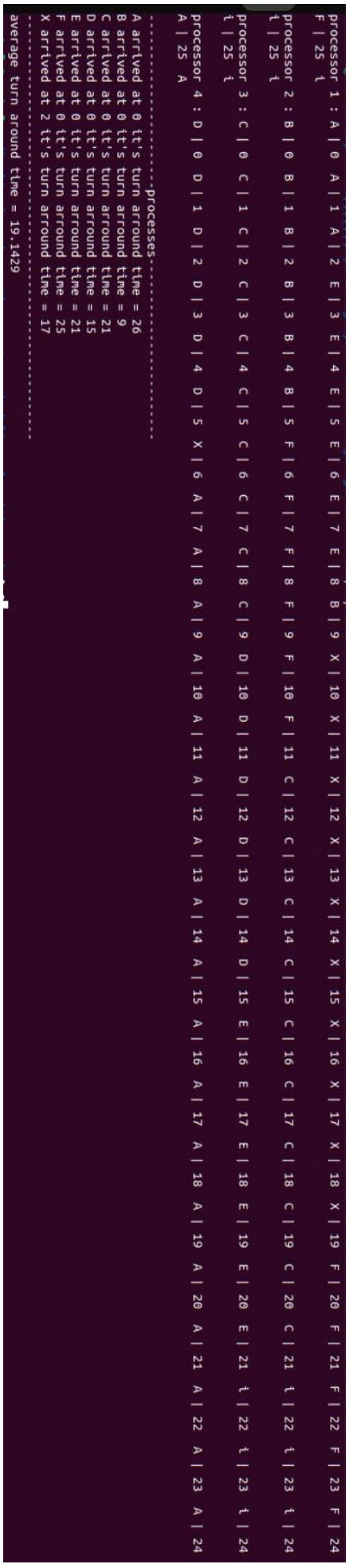
6 EXE

X 3 2

1 EXE

1 I/O

10 EXE



MLFQ

7

A 4 0

3 EXE

2 I/O

20 EXE

1 I/O

B 3 0

6 EXE

2 I/O

1 EXE

C 3 0

10 EXE

2 I/O

10 EXE

D 3 0

6 EXE

3 I/O

6 EXE

E 3 0

6 EXE

3 I/O

6 EXE

F 3 0

6 EXE

3 I/O

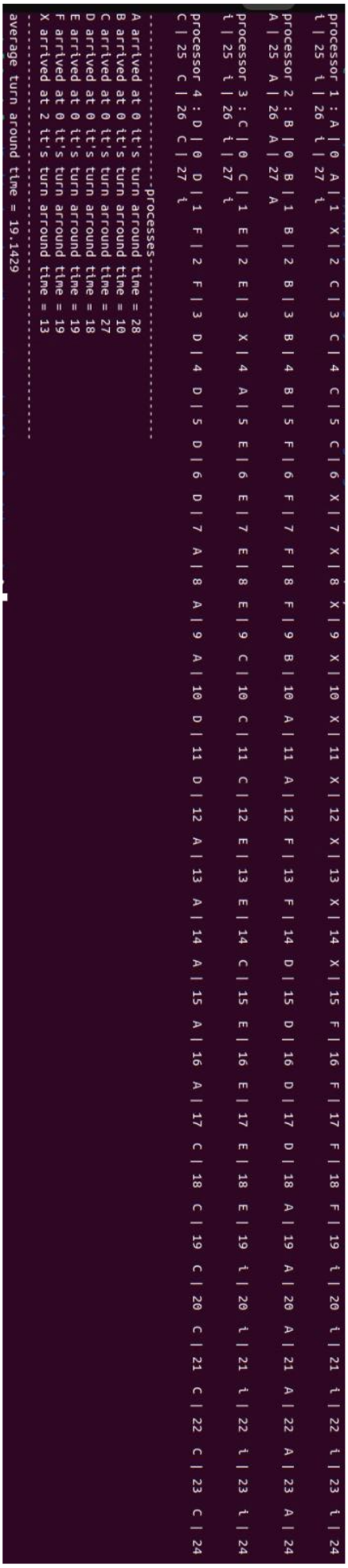
6 EXE

X 3 2

1 EXE

1 I/O

10 EXE



STRIDE

7

A 4 0

3 EXE

2 I/O

20 EXE

1 I/O

B 3 0

6 EXE

2 I/O

1 EXE

C 3 0

10 EXE

2 I/O

10 EXE

D 3 0

6 EXE

3 I/O

6 EXE

E 3 0

6 EXE

3 I/O

6 EXE

F 3 0

6 EXE

3 I/O

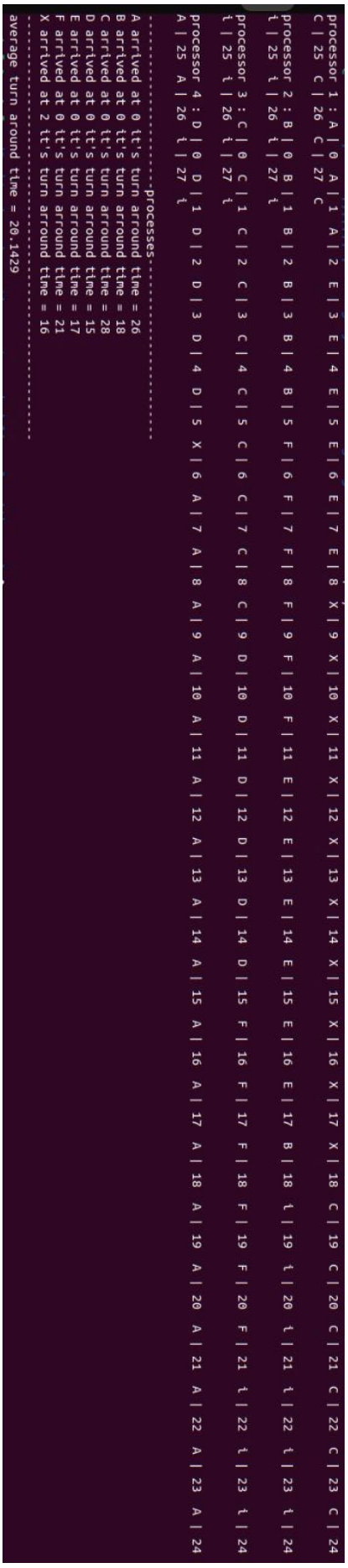
6 EXE

X 3 2

1 EXE

1 I/O

10 EXE



STCF

7

A 4 0

3 EXE

2 I/O

20 EXE

1 I/O

B 3 0

6 EXE

2 I/O

1 EXE

C 3 0

10 EXE

2 I/O

10 EXE

D 3 0

6 EXE

3 I/O

6 EXE

E 3 0

6 EXE

3 I/O

6 EXE

F 3 0

6 EXE

3 I/O

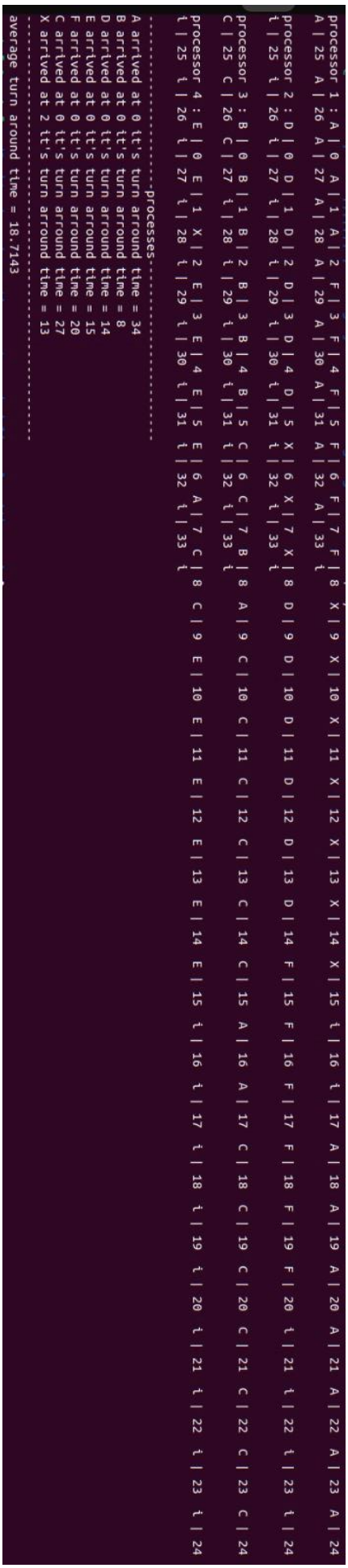
6 EXE

X 3 2

1 EXE

1 I/O

10 EXE



Priority

7

A 4 0

3 EXE

2 I/O

20 EXE

1 I/O

B 3 0

6 EXE

2 I/O

1 EXE

C 3 0

10 EXE

2 I/O

10 EXE

D 3 0

6 EXE

3 I/O

6 EXE

E 3 0

6 EXE

3 I/O

6 EXE

F 3 0

6 EXE

3 I/O

6 EXE

X 3 2

1 EXE

1 I/O

10 EXE