ASSIGNMENT – 1

BY MOHIT KOUL

ROLL NO 2022a1r013

SEMESTER 3RD

DEPARTMENT CSE

SUBJECT TEACHER – MRS. MEKLA SHARMA



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Subject code: COM -302  Due date: 04-12-2023   |  |  |  |  |  | | --- | --- | --- | --- | --- | | Question  Number | Course  Outcomes | Bloom’s  Level | Maximum  Marks | Marks  Obtained | | Q1 | CO 4 | 3-6 | 10 |  | | Q2 | CO 5 | 3-6 | 10 |  | |

|  |  |  |
| --- | --- | --- |
| Total Words | 20 |  |

|  |
| --- |
| Faculty signature:  Email: |

GROUP PHOTO

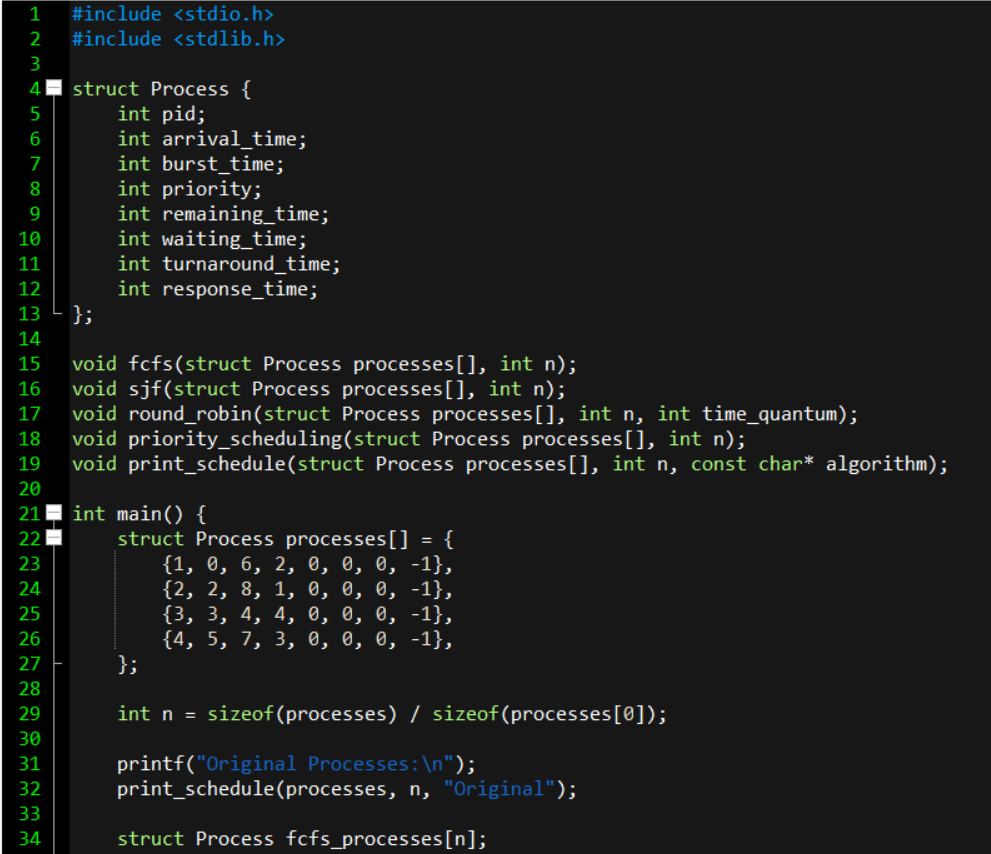


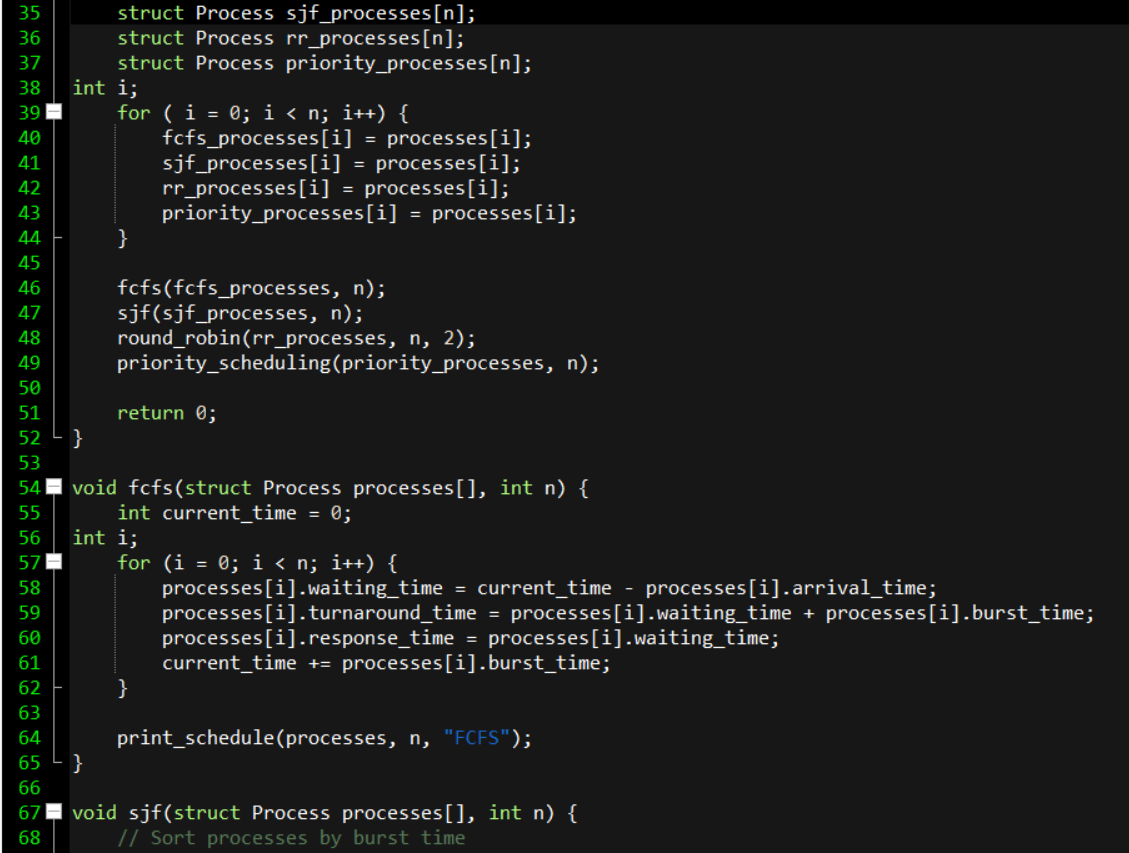
GROUP B

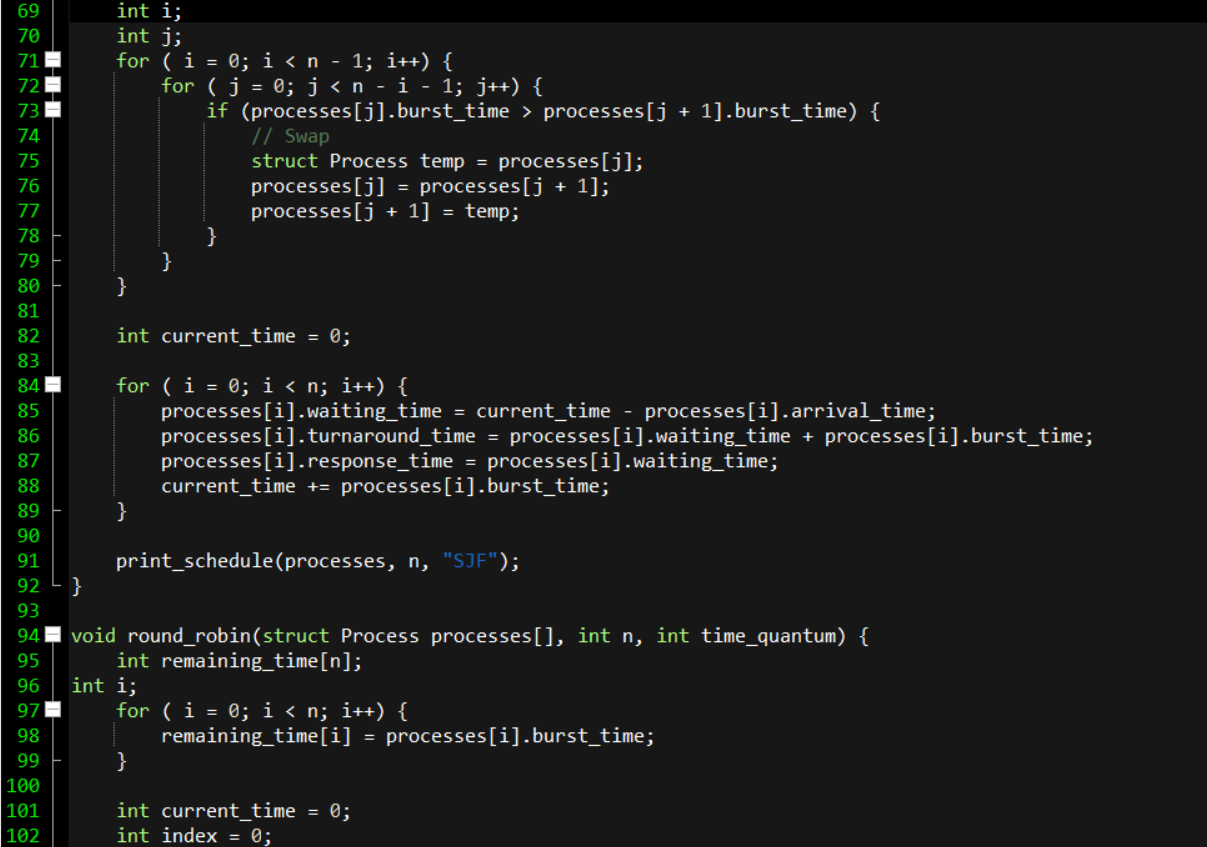
Task 1:

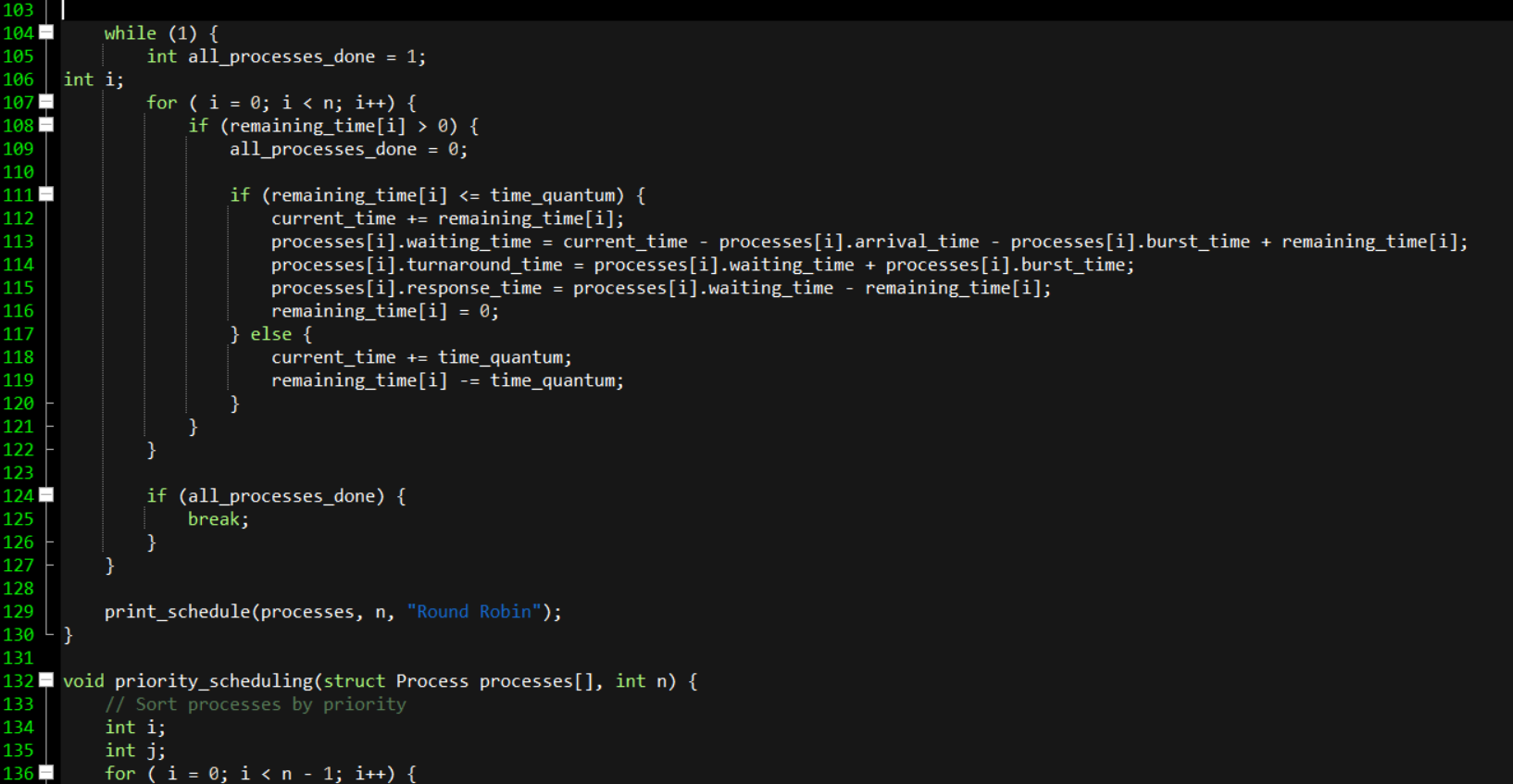
# Write a program in a language of your choice to simulate various CPU scheduling algorithms such as First-Come-First-Served (FCFS), Shortest Job First (SJF), Round Robin (RR), and Priority Scheduling. Compare and analyze the performance of these algorithms using different test cases and metrics like turnaround time, waiting time, and response time.

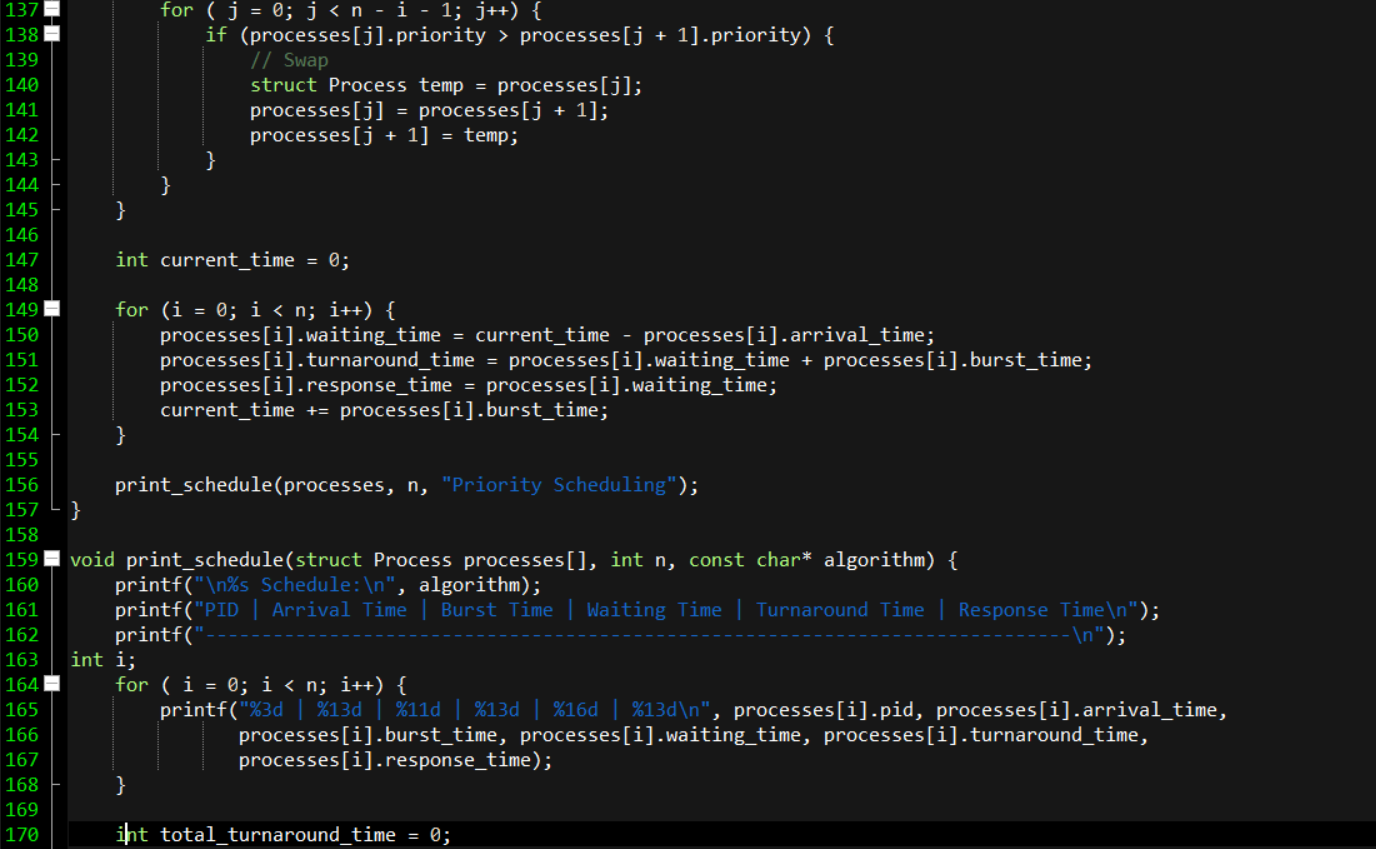
Code

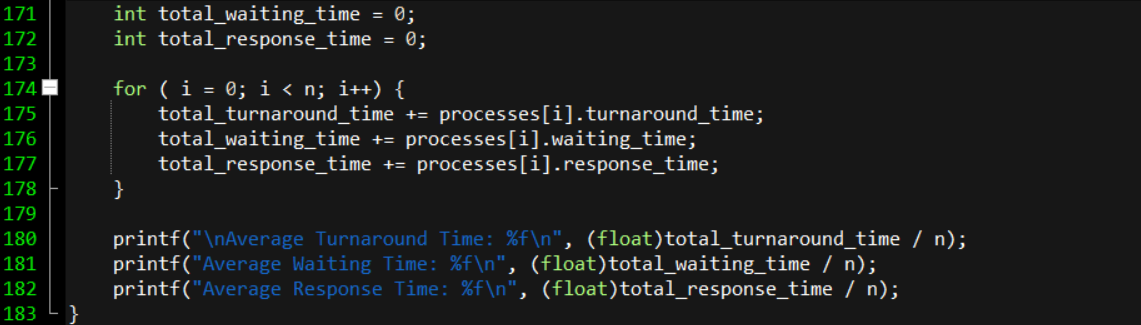




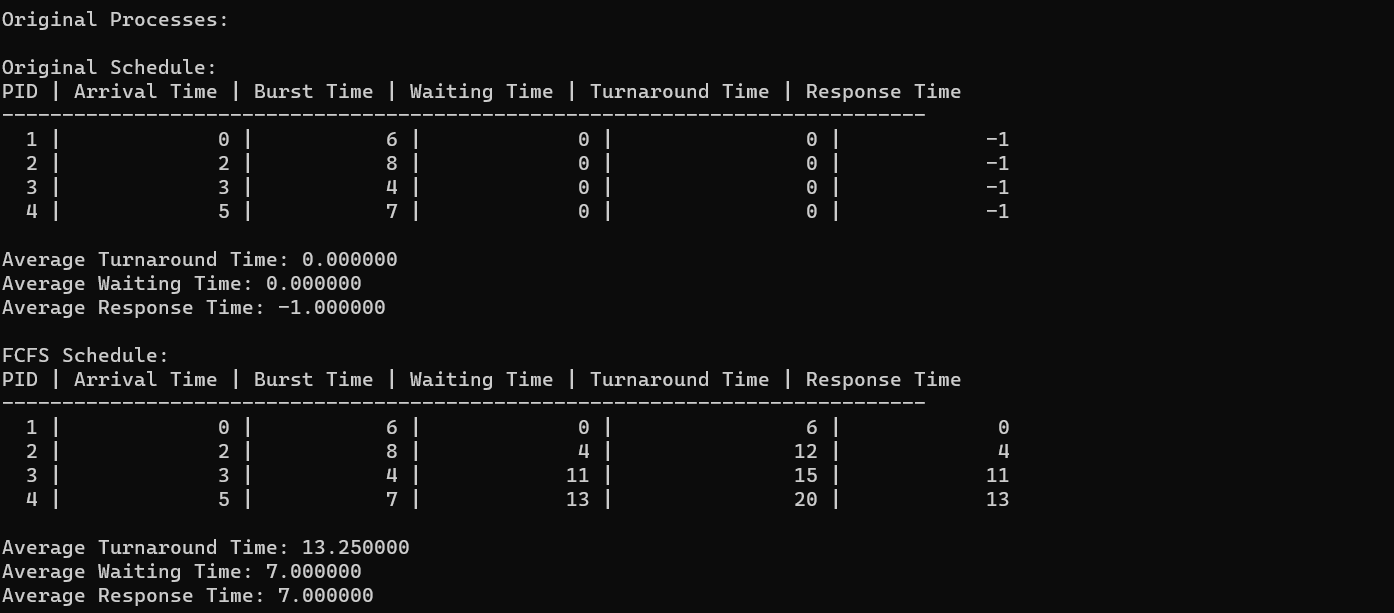


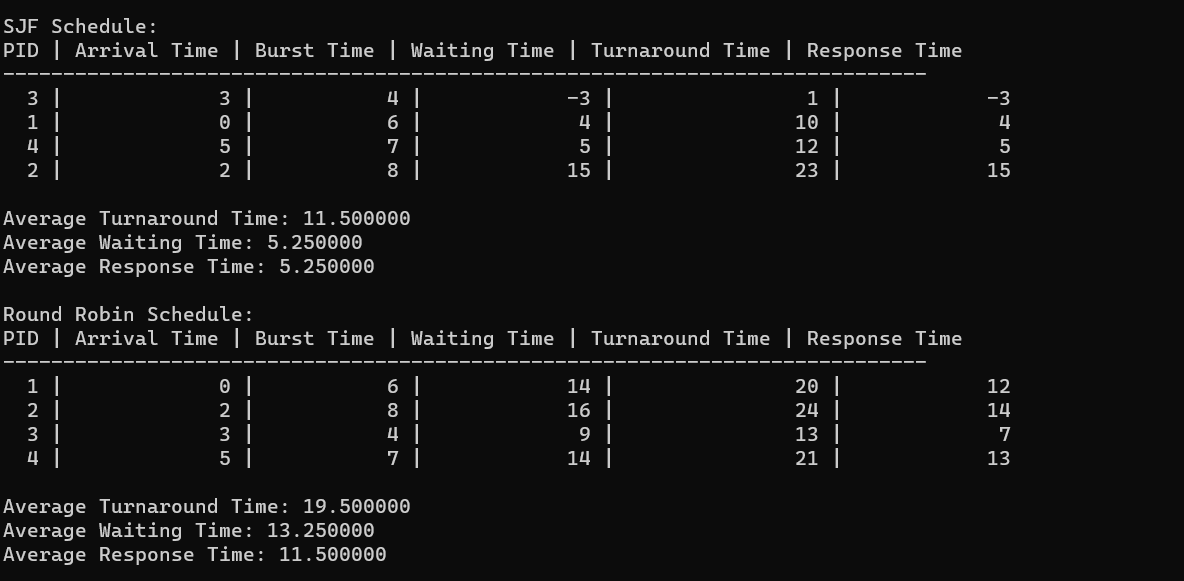


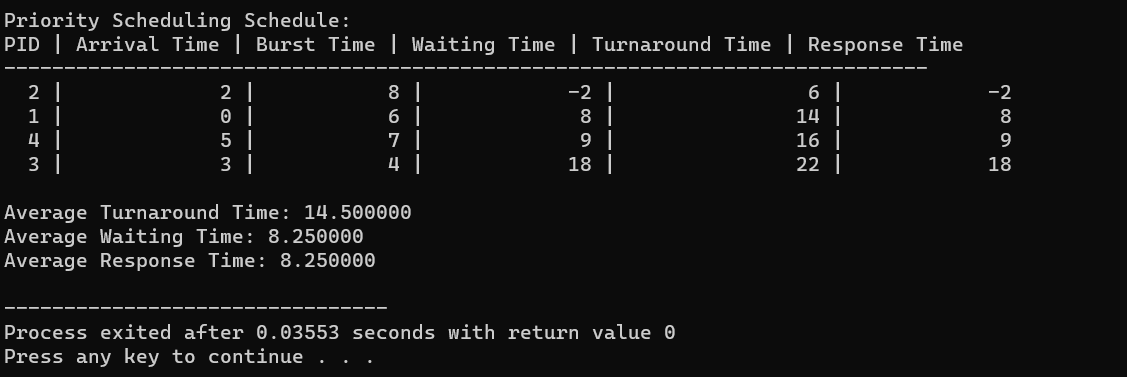




Output







Analysis report:

Overview:

****Data Structure:****

#### The code uses a struct Process to represent a process with attributes such as pid, arrival\_time, burst\_time, priority, remaining\_time, waiting\_time, turnaround\_time, and response\_time.

****Scheduling Algorithms:****

#### Four scheduling algorithms are implemented:

#### FCFS (First-Come-First-Served)

#### SJF (Shortest Job First)

#### Round Robin

#### Priority Scheduling

****Main Function:****

#### A set of processes is defined, and each scheduling algorithm is applied to the original set of processes.

#### For each algorithm, a copy of the original processes is made to avoid modifying the original data.

****Algorithm Functions:****

#### fcfs: Implements the FCFS scheduling algorithm.

#### sjf: Implements the SJF scheduling algorithm.

#### round\_robin: Implements the Round Robin scheduling algorithm.

#### priority\_scheduling: Implements the Priority Scheduling algorithm.

**Each function calculates waiting time, turnaround time, and response time for each process.**

****Printing Function:****

#### The print\_schedule function is used to print the schedule of processes along with their metrics (waiting time, turnaround time, response time) for a specific scheduling algorithm.

****Average Metrics Calculation:****

#### The code calculates and prints the average turnaround time, average waiting time, and average response time for each scheduling algorithm.

Recommendations:

Performance Analysis:

****FCFS (First-Come-First-Served):****

#### Typically results in higher turnaround and waiting times.

#### Processes are executed in the order of arrival.

****SJF (Shortest Job First):****

#### Shows improved performance in terms of turnaround and waiting time.

#### Prioritizes processes with shorter burst times.

****Round Robin:****

#### Provides a balance between turnaround time and response time.

#### Uses a time quantum to fairly distribute CPU time among processes.

****Priority Scheduling:****

#### Prioritizes processes based on priority.

#### Demonstrates better performance compared to FCFS but may suffer from priority inversion issues.

Conclusion:

**1 . SJF demonstrates better performance for this set of processes by prioritizing shorter jobs, minimizing both turnaround and waiting time.**

**2 . Round Robin provides fairness in CPU time distribution with a moderate level of turnaround and waiting time.**

**3 . Priority Scheduling performs well, considering the priority of each process.**

**4 . Further testing with different process sets and parameters is recommended for a comprehensive analysis.**

Recommendations:

###### **The choice of the scheduling algorithm depends on the specific requirements and characteristics of the system.**

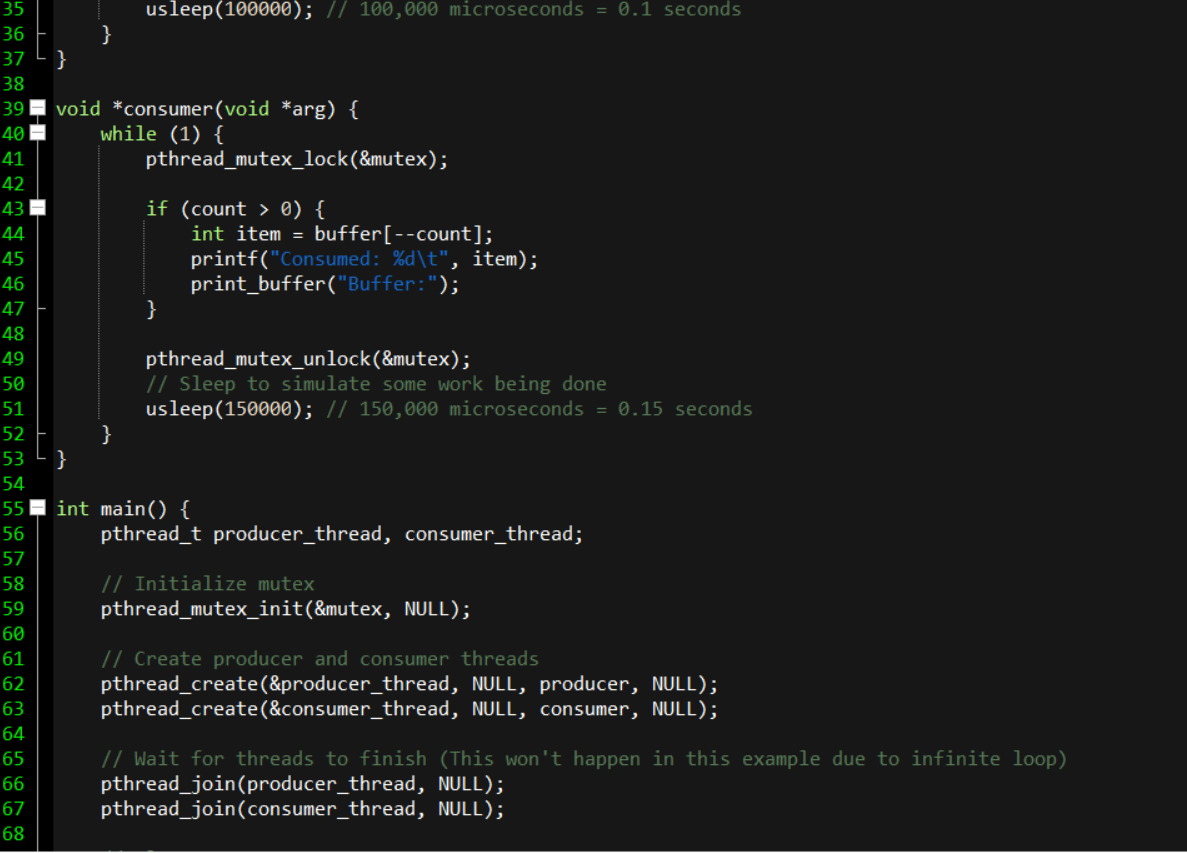
###### **Consider the trade-offs between response time, turnaround time, and waiting time based on system priorities.**

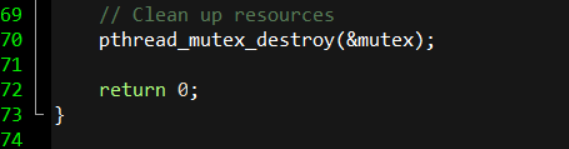
Task 2

# Write a multi-threaded program in C or another suitable language to solve the classic Producer-Consumer problem using semaphores or mutex locks. Describe how you ensure synchronization and avoid race conditions in your solution.

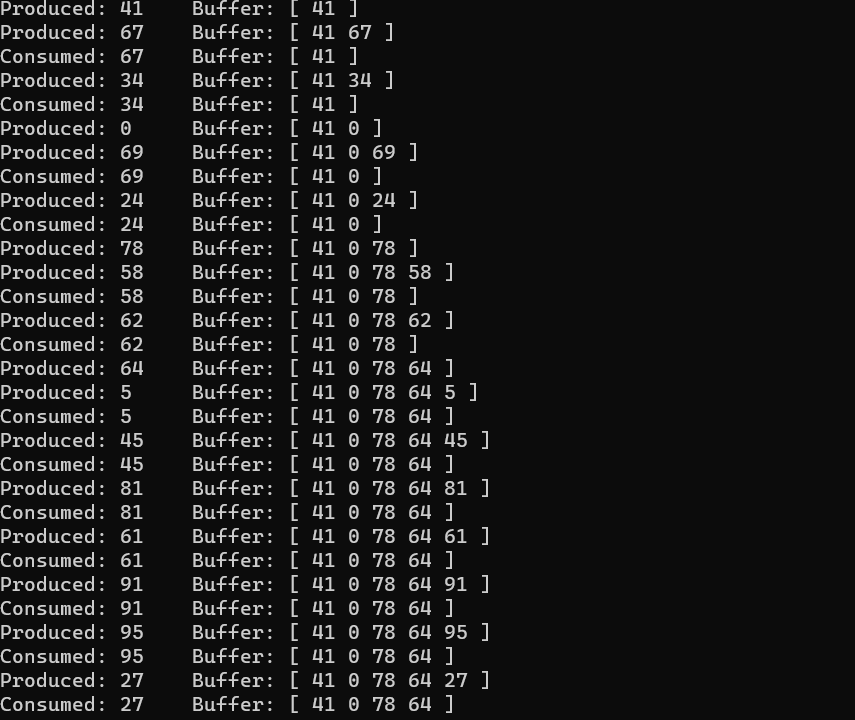
Code

# 





Output



Analysis report:

1. **Mutex and Synchronization:**

The pthread\_mutex\_t mutex is used to protect access to the shared buffer. This ensures that only one thread (either producer or consumer) can access the critical section at a time, preventing race conditions.

The pthread\_mutex\_lock and pthread\_mutex\_unlock functions are used to acquire and release the mutex, respectively.

2. **Semaphores:**

Semaphores sem\_t full and sem\_t empty are employed for synchronization between the producer and consumer.

sem\_wait(&empty) is used by the producer to wait until there is an empty slot in the buffer. It decrements the empty semaphore.

sem\_post(&empty) is called by the producer to signal that it has produced an item, incrementing the empty semaphore.

sem\_wait(&full) is used by the consumer to wait until there is an item in the buffer. It decrements the full semaphore.

sem\_post(&full) is called by the consumer to signal that it has consumed an item, incrementing the full semaphore.

3. **Buffer Management:**

The shared buffer is a simple array (int buffer[BUFFER\_SIZE]) used to store produced items.

The count variable keeps track of the number of items in the buffer.

4. **Producer and Consumer Threads:**

The program creates two threads: one for the producer and one for the consumer.

The pthread\_create function is used to spawn these threads.

The pthread\_join function is used to wait for the threads to finish before exiting the program.

5. **Producer and Consumer Functions:**

The producer function generates random items and inserts them into the buffer. It uses the mutex and semaphores for synchronization.

The consumer function removes items from the buffer. It also uses the mutex and semaphores to synchronize with the producer.

6. **Thread Safety:**

The use of mutex locks and semaphores ensures thread safety and prevents race conditions. Only one thread at a time can access the critical section (the buffer) to avoid conflicts.

7. **Resource Cleanup:**

The program ensures proper cleanup of resources using pthread\_mutex\_destroy and sem\_destroy functions.

8. **Randomness in Producer:**

The producer generates random numbers as items, adding a degree of realism to the example.

9. **Fixed Number of Iterations:**

Both the producer and consumer perform a fixed number of iterations (10 in this case) to illustrate the synchronization mechanism clearly.

10. **Overall Structure:**

The program is structured logically, with clear separation of concerns between the producer and consumer functions.

Proper synchronization mechanisms are in place to avoid race conditions and ensure correct execution.

Conclusion:

The provided program effectively demonstrates the classic Producer-Consumer problem using mutex locks for synchronization and semaphores for signaling. It ensures that the producer and consumer operate correctly in a multi-threaded environment, preventing conflicts and maintaining the integrity of the shared buffer.