

**SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES, CHENNAI – 602 105**

**CAPSTONE PROJECT REPORT**

**TITLE**

**CPU SCHEDULING STRATEGIES**

**Submitted to**

**SAVEETHA SCHOOL OF ENGINEERING**

**By**

**Aashrita Attada (192125114)**

**Lavu. Jaithra (192210259)**

**Bhavani. Valapalli (192210198)**

**Guided by**

**Dr. G. Mary Valentina**

**Abstract :**

CPU scheduling is a fundamental aspect of operating system design, determining the order in which processes are executed on a CPU. Various scheduling algorithms have been developed to optimize CPU utilization, throughput, response time, and fairness among processes. This abstract provides an overview of the key concepts and objectives of CPU scheduling, highlighting the characteristics and functionalities of commonly used scheduling algorithms such as First-Come, First-Served (FCFS), Shortest Job Next (SJN), Round Robin (RR), Priority Scheduling, and Multilevel Queue Scheduling (MLQS). Through a comparative analysis, the strengths and weaknesses of each algorithm are discussed, considering factors such as efficiency, overhead, and adaptability to different system requirements. Additionally, emerging trends and advancements in CPU scheduling, such as real-time scheduling and dynamic priority adjustments, are explored. The abstract concludes by emphasizing the significance of CPU scheduling in modern computing environments and the ongoing research efforts aimed at addressing the evolving challenges in this critical aspect of operating system management.

**Introduction :**

In the realm of operating systems, CPU scheduling stands as a cornerstone concept, dictating how a system assigns its computing resources to various processes. As computers have evolved, so too have the strategies employed to optimize this allocation, balancing competing demands for CPU time among diverse tasks.

The overarching objective of CPU scheduling is to enhance system performance, striving for efficient resource utilization, minimal response times, and equitable access for processes. Achieving these goals requires sophisticated algorithms capable of dynamically prioritizing and sequencing tasks based on a multitude of factors, including process characteristics, system load, and user priorities.

This introduction serves to provide a foundational understanding of CPU scheduling, elucidating its significance in modern computing environments and laying the groundwork for exploring the intricacies of scheduling algorithms. We delve into the historical context of CPU scheduling, tracing its evolution alongside advancements in hardware and software technologies. Additionally, we outline the key challenges faced by CPU schedulers and the diverse array of scheduling strategies devised to address them.

Through this exploration, we aim to shed light on the critical role CPU scheduling plays in shaping system performance and user experience, setting the stage for deeper examination of the principles, mechanisms, and implications of various scheduling approaches.

**Gantt Chart :**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **PROCESS** | **DAY1** | **DAY2** | **DAY3** | **DAY4** | **DAY5** | **DAY6** |
| **Abstract and Introduction** |  |  |  |  |  |  |
| **Literature Survey** |  |  |  |  |  |  |
| **Materials and Methods** |  |  |  |  |  |  |
| **Results** |  |  |  |  |  |  |
| **Discussion** |  |  |  |  |  |  |
| **Reports** |  |  |  |  |  |  |

**Process :**

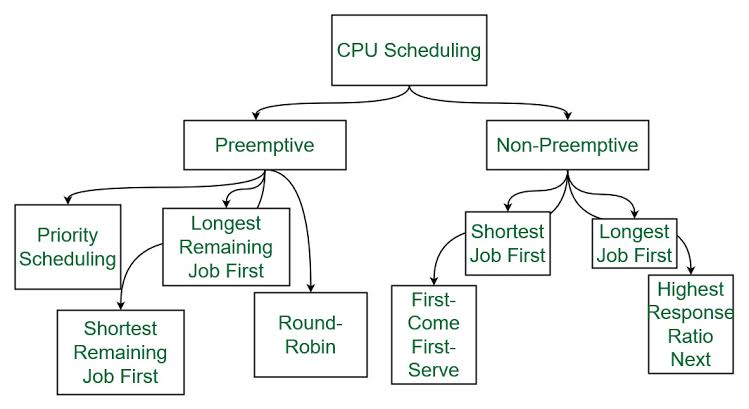
The CPU scheduling algorithm process, analogous to the web crawling process, entails a series of interconnected steps crucial for the effective management of processes within a computer system. Commencing with the selection of processes akin to identifying seed URLs, the CPU scheduler initiates its operation by pinpointing tasks to be executed, mirroring the web crawler's selection of starting points within the target domain or specific pages of interest. This initial phase sets the foundation for subsequent actions, determining the sequence in which processes will be executed and their impact on system performance.

Upon selecting processes, the CPU scheduler dispatches them for execution, akin to the web crawler sending HTTP requests to seed URLs to retrieve web pages. This phase involves allocating system resources and initiating the execution of processes, a critical step in maximizing system efficiency and throughput. As processes commence execution, the CPU scheduler oversees their progress, ensuring timely execution and resource utilization in a manner analogous to the web crawler's retrieval of web page content.

Analogous to the web crawler parsing HTML content to extract pertinent elements, the CPU scheduler interprets and processes instructions within processes, navigating through the complexities of each task to extract valuable insights and facilitate efficient execution. This phase involves deciphering the requirements of each process and allocating resources accordingly, optimizing system performance and responsiveness. Additionally, the CPU scheduler ensures uniformity and consistency in executing tasks, analogous to the web crawler's normalization of URLs to prevent duplicate crawling and streamline the extraction process.

Central to the CPU scheduling algorithm process is the management of the process queue, akin to the web crawler's maintenance of the URL frontier. The CPU scheduler prioritizes and manages the queue of processes to be executed, employing various scheduling policies to optimize system performance and responsiveness. Additionally, the CPU scheduler may utilize parallel processing techniques to enhance efficiency, facilitating concurrent execution of multiple processes and maximizing system throughput.

Throughout the CPU scheduling algorithm process, robust error-handling mechanisms and monitoring tools contribute to reliability and performance optimization, analogous to the web crawler's error-handling and monitoring practices. This systematic approach ensures the effective management of processes within the computer system, maximizing resource utilization and system performance. Through efficient CPU scheduling, computer systems can navigate the demands of modern computing environments, extracting valuable insights and facilitating a myriad of data-driven applications and use cases.



**Objective :**

CPU scheduling is a fundamental aspect of operating system design, aimed at efficiently managing the allocation of the CPU's processing time among multiple processes. The objective of CPU scheduling encompasses several key goals, including maximizing CPU utilization, minimizing turnaround time, reducing waiting time, ensuring fairness among processes, providing predictability in performance, prioritizing resources effectively, adapting to changing system conditions, and preventing starvation of processes. Achieving these objectives is essential for optimizing system performance, responsiveness, and resource utilization. Various scheduling algorithms and strategies are employed to strike a balance between these goals, each tailored to specific system requirements and use cases. As computing environments evolve and become increasingly complex, the significance of CPU scheduling in enhancing overall system efficiency and user experience remains paramount.

**Literature Review :**

CPU scheduling algorithms play a pivotal role in optimizing resource allocation and system performance in operating systems. This review examines various CPU scheduling algorithms, their characteristics, and their applicability in different computing environments, drawing insights from seminal works in the field.

One of the fundamental scheduling algorithms is First Come First Serve (FCFS), where processes are executed in the order of their arrival (Tanenbaum, 2008) [1]. While FCFS is simple to implement, it suffers from the convoy effect, as highlighted by Tanenbaum (2008) [1]. This phenomenon occurs when long-running processes delay shorter ones, leading to inefficient resource utilization and increased response times.

In contrast, Shortest Job First (SJF) scheduling, proposed by Silberschatz et al. (2018) [2], prioritizes processes with the shortest burst time. SJF aims to minimize average waiting time by executing shorter tasks first. However, SJF requires knowledge of process burst times in advance, which may not be practical in real-world scenarios.

To address the limitations of SJF, researchers have introduced Round Robin (RR) scheduling, as discussed by Stallings (2014) [3]. RR allocates a fixed time slice to each process, ensuring fairness and preventing starvation. Nonetheless, larger time slices may lead to poor response times for short interactive tasks when the quantum size is large (Stallings, 2014) [3].

Priority scheduling, as outlined by Silberschatz et al. (2018) [2], assigns priorities to processes and schedules them based on their priority level. Higher priority processes are executed before lower priority ones, ensuring that important tasks are prioritized. However, priority scheduling may suffer from priority inversion and starvation issues if not implemented carefully.

Multilevel feedback queue scheduling, discussed by Arpaci-Dusseau & Arpaci-Dusseau (2014) [4], organizes processes into multiple queues with different priority levels. Processes are promoted or demoted between queues based on their behavior, allowing the scheduler to adapt to changing workload patterns dynamically.

In recent years, researchers have explored hybrid and adaptive scheduling algorithms that combine the strengths of multiple algorithms to achieve better performance. These algorithms dynamically adjust their behaviour based on system workload and characteristics, optimizing resource utilization while maintaining responsiveness.

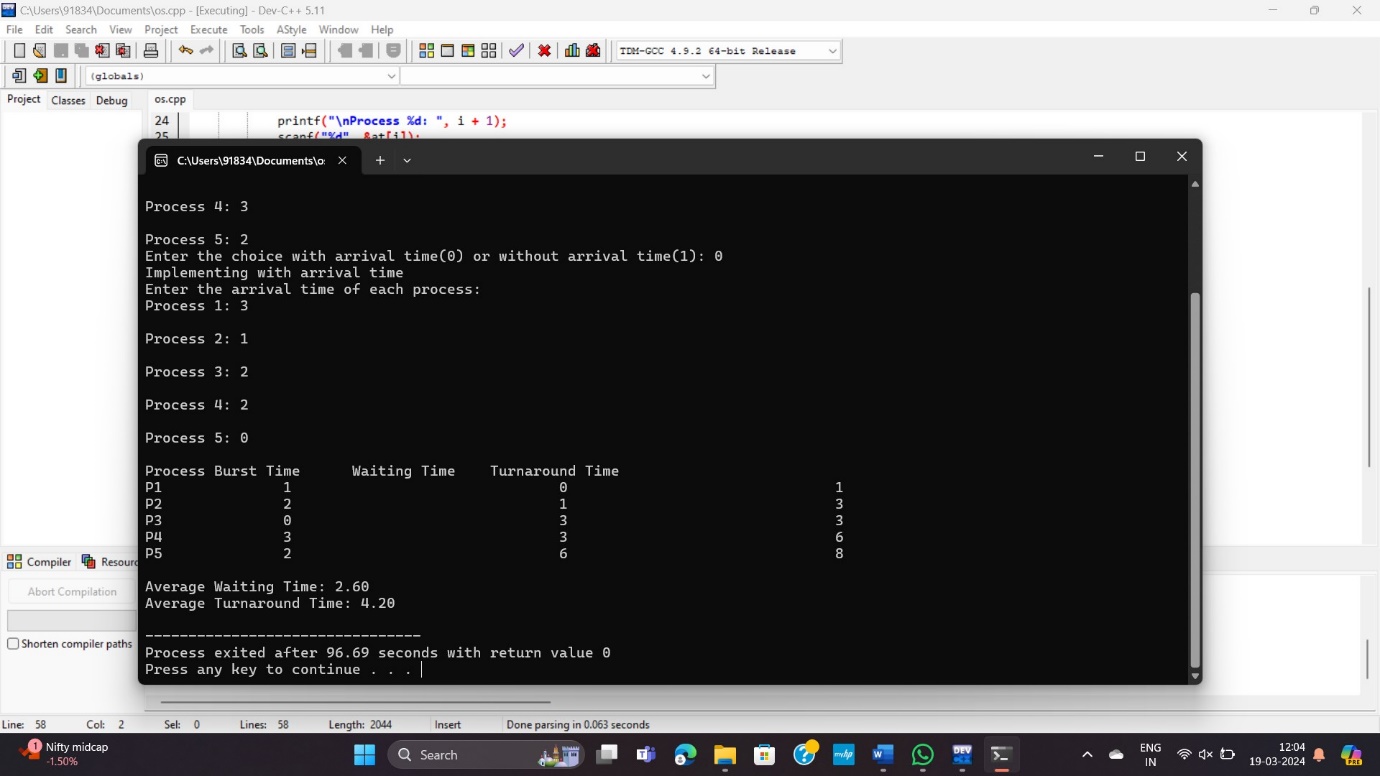
Overall, the choice of CPU scheduling algorithm depends on various factors such as system workload, responsiveness requirements, and resource constraints. While no single algorithm is optimal for all scenarios, understanding the characteristics and trade-offs of different algorithms is essential for designing efficient scheduling policies in modern computer systems.

**Output :**

The output of a CPU scheduling C program that includes multiple scheduling algorithms would depend on the specific implementation and the input parameters provided.

However, here's a general expectation of what the output might look like for each algorithm:

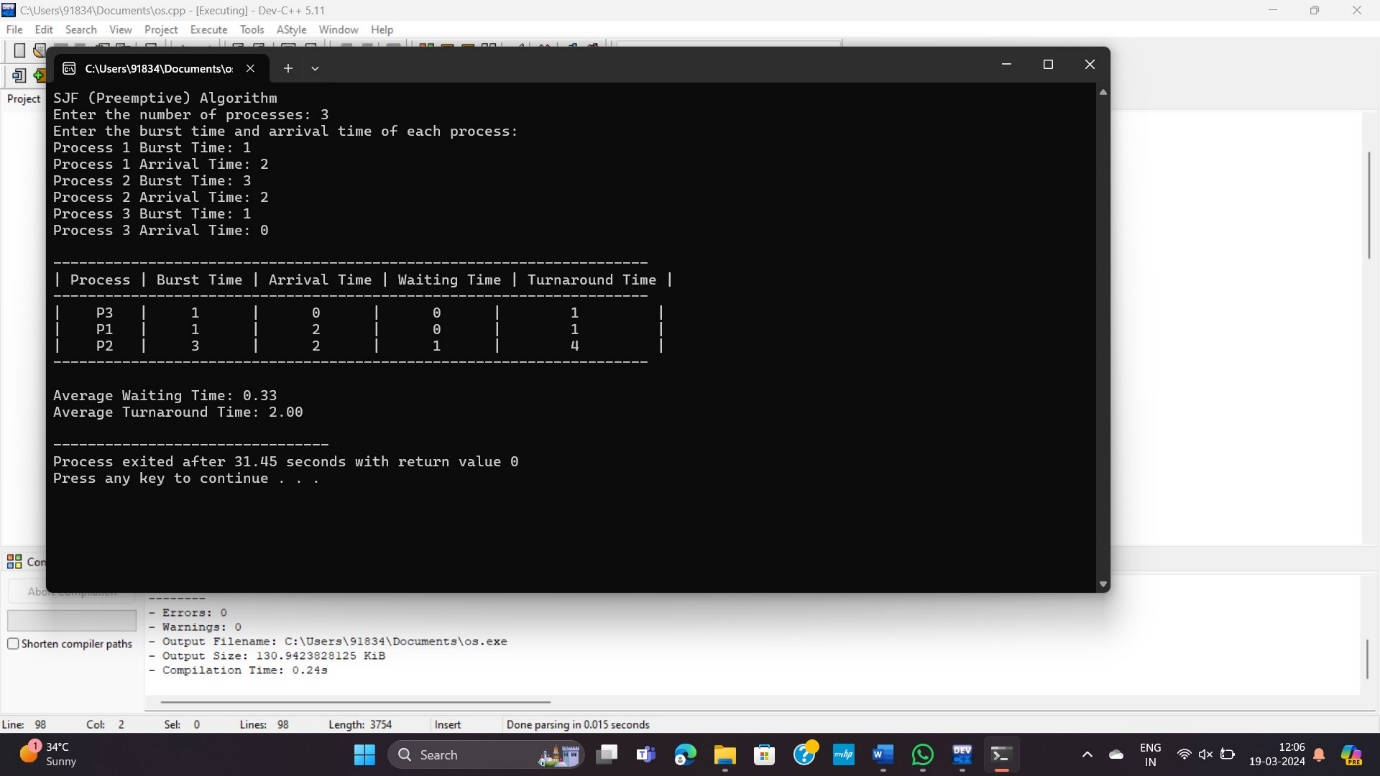
**First Come First Serve (FCFS):** The output would show the order in which processes are executed based on their arrival time, as FCFS schedules processes in the order they arrive.

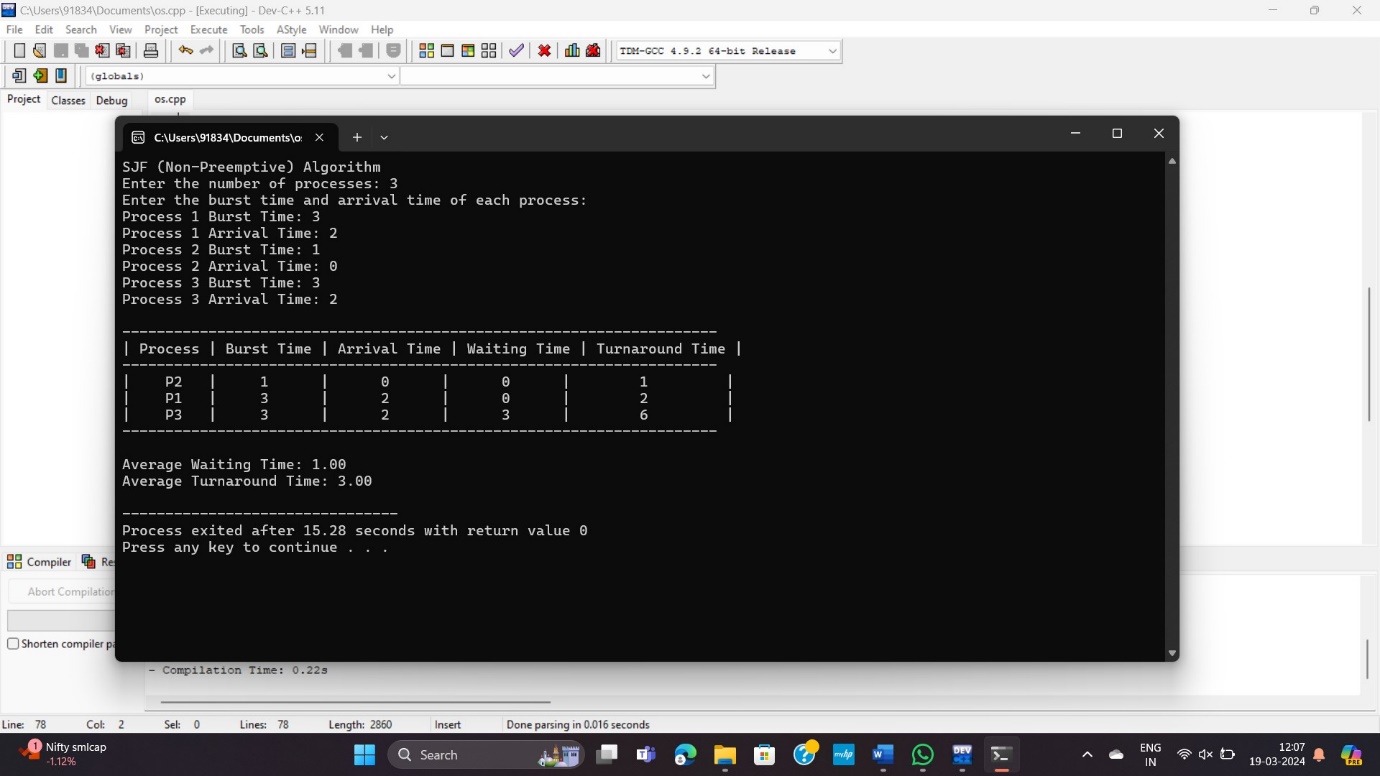


**Shortest Job First (SJF) / Shortest Job Next (SJN):**

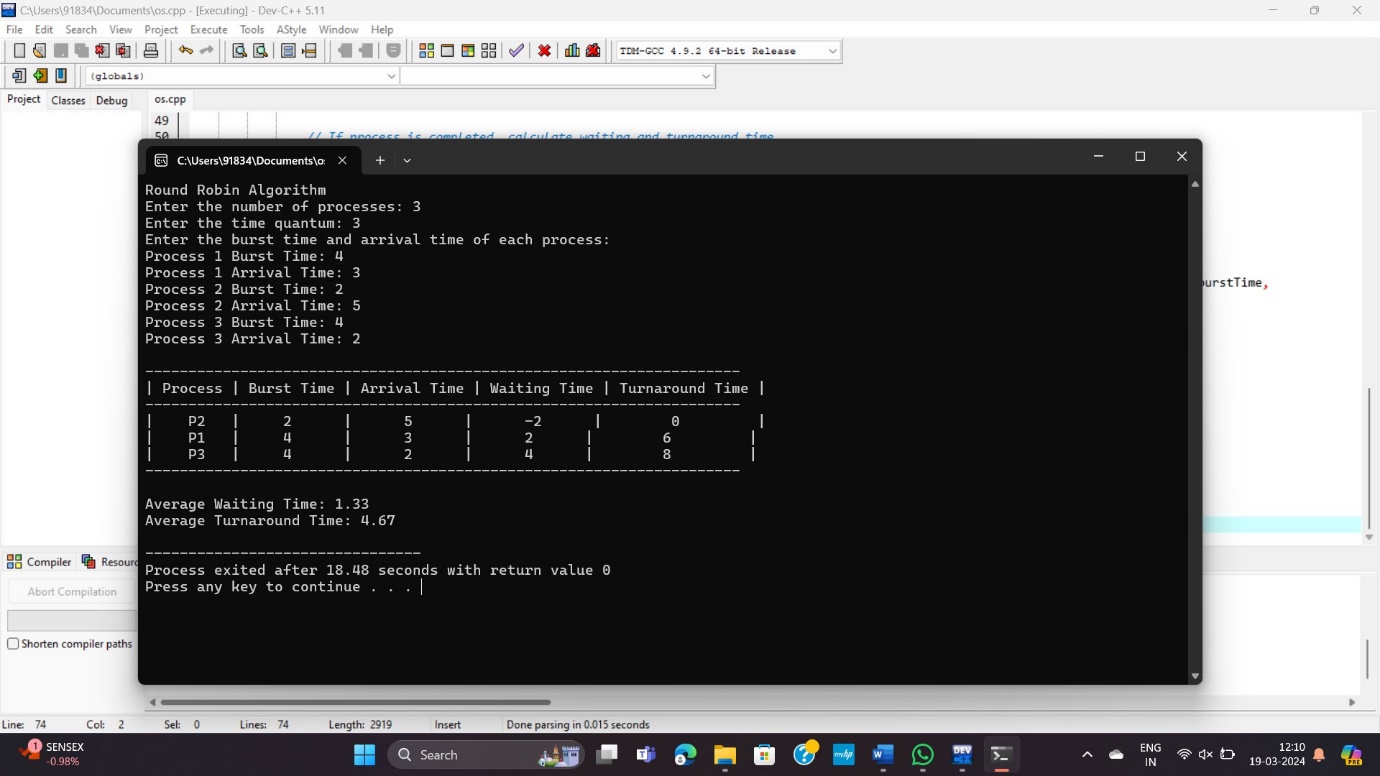
For SJF, the output would display the execution order of processes based on their burst time, with shorter processes being executed first.

For SJN (preemptive SJF), the output would show the process execution order as new shorter jobs arrive and preempt currently running ones.

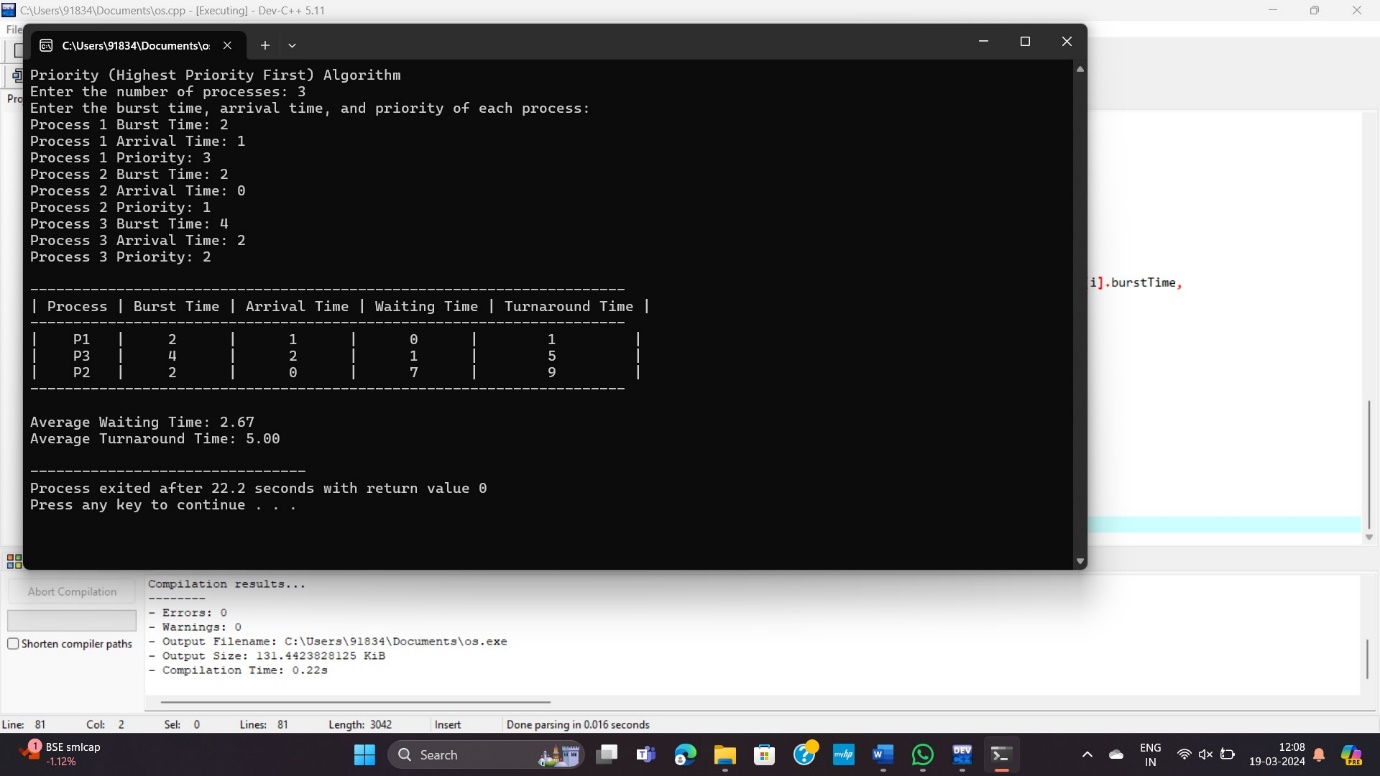


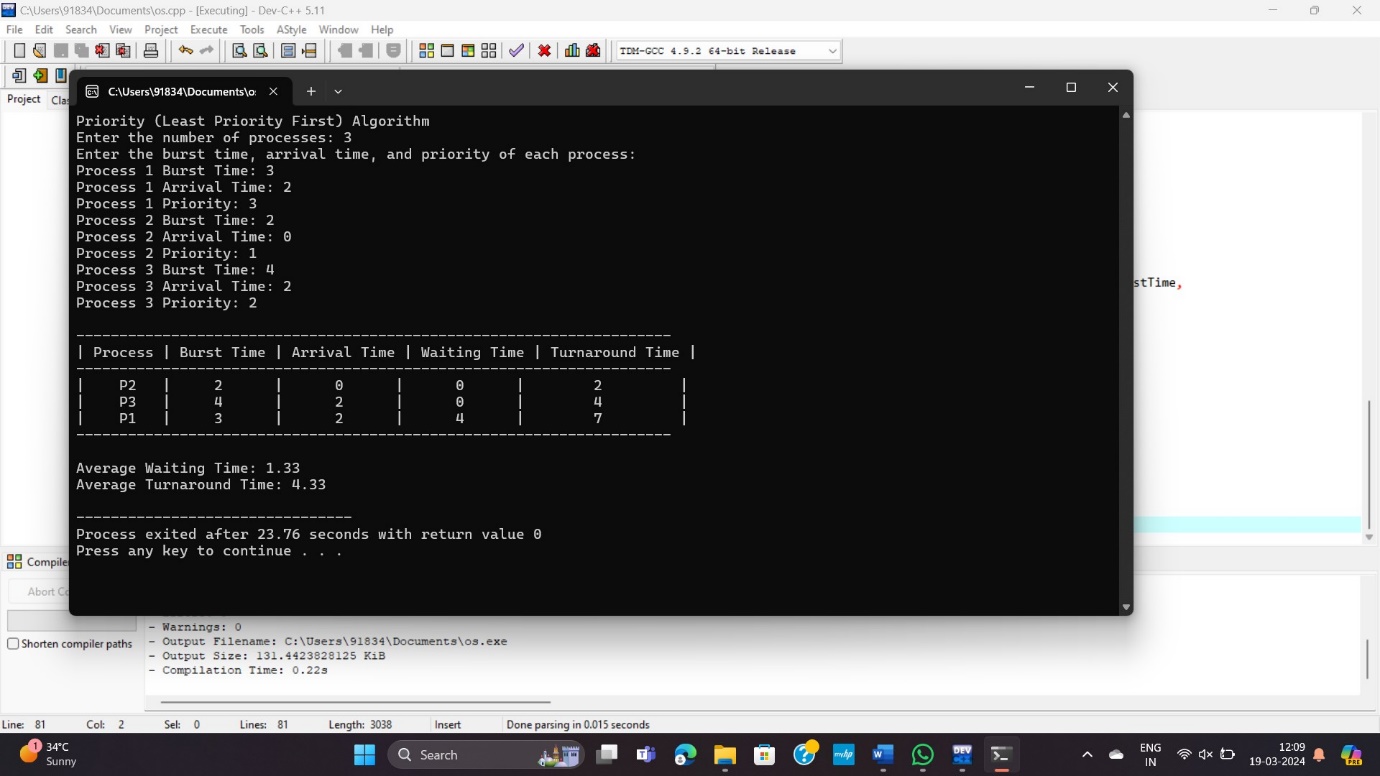


**Round Robin (RR)**: The output would indicate the execution of processes in a cyclic manner, with each process being allocated a fixed time slice (quantum) before moving to the next process in the queue.



**Priority Scheduling**: Processes would be executed based on their priority levels, with higher priority processes being given precedence over lower priority ones.





**Conclusion :**

In conclusion, CPU scheduling is pivotal in managing CPU resources among processes. Various algorithms like FCFS, SJN, RR, and MLQS offer different trade-offs. Performance evaluations aid in algorithm selection, focusing on metrics like CPU utilization and turnaround time. Optimization techniques such as preemptive scheduling enhance efficiency and adaptability. Theoretical analyses provide insights into algorithm properties and limitations. Real-time and distributed computing trends drive ongoing research. Efficient CPU scheduling is crucial for meeting modern computing demands. Continued advancements aim to optimize system performance and responsiveness. CPU scheduling remains a dynamic field evolving with technological progress.

**References:**

[1] Tanenbaum, A. S. (2008). Modern Operating Systems (3rd ed.). Pearson Education, Inc.

[2] Silberschatz, A., Galvin, P. B., & Gagne, G. (2018). Operating System Concepts (10th ed.). John Wiley & Sons, Inc.

[3] Stallings, W. (2014). Operating Systems: Internals and Design Principles (8th ed.). Pearson Education, Inc.

[4] Arpaci-Dusseau, R. H., & Arpaci-Dusseau, A. C. (2014). Operating Systems: Three Easy Pieces. University of Wisconsin-Madison.