**­**

**Total Marks: 7.5**

**Marks Obtained:**

**Operating System**

**HUSJ CPU Scheduling**

**Project Report**

**Submitted To: sir Jawad Naseer**

**Date of Submission: May 16, 2025**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **Student Name** | **Reg.no** |
| **01** | **Muhammad Salman** | **2212400** |
| **02** | **Mushahid Hussain** | **2212408** |
| **03** | **Ubaid Bin Waris** | **2212416** |
| **04** | **Jehanzeb Khalid** | **2212391** |

Table of Contents

**Title page01**

**Table of Content02**

**Introduction03**

**Objective04**

**Methodology04**

Process Generation04

Process Reading05

Scheduling Logic05

Performance Calculation05

Output05

**Implementation06**

Process Generation Module06

Round Robin Schedular06

**Result08**

**Conclusion09**

**References09**

**Introduction**

In modern operating systems, efficient process scheduling plays a critical role in optimizing CPU utilization and improving system responsiveness. Among the various scheduling algorithms, **Round Robin (RR)** stands out due to its fairness and simplicity, especially in time-sharing systems. This project, titled **HUSJ**, presents a comprehensive implementation of the Round Robin scheduling algorithm, with an emphasis on **real-time process simulation** using input read from a file.

The system generates a set of processes with randomized **arrival times** and **burst (execution) times**, stored in a structured text file. The core of the simulation then reads this data, performs scheduling using a fixed **time quantum**, and computes critical performance metrics such as **waiting time**, **turnaround time**, **completion time**, and **throughput**. A **Gantt chart representation** of the process execution order is also displayed to enhance visual understanding of the CPU time allocation.

Designed with a **modular approach**, the project separates process generation, file handling, and scheduling logic into distinct components. This makes the code more maintainable, testable, and extensible for future enhancements such as adding priority-based scheduling or graphical visualization.

The **HUSJ** project not only demonstrates how theoretical concepts of CPU scheduling are implemented in practice, but also reinforces the importance of system-level programming, file I/O operations, and algorithmic thinking in operating systems coursework.

**Objective**

The primary objective of the **HUSJ** project is to simulate the **Round Robin CPU scheduling algorithm** using process data read from an external file. The goal is to model how an operating system handles process scheduling in a fair and time-efficient manner. The project aims to:

* Generate a set of processes with randomized arrival and execution times.
* Store and retrieve these processes from a file (processes.txt).
* Implement Round Robin scheduling using a fixed time quantum.
* Track and display key performance metrics such as:
  + Completion time
  + Waiting time
  + Turnaround time
  + Throughput
* Provide a visual execution trace in the form of a Gantt chart.
* Maintain code modularity for ease of maintenance and understanding.

**Methodology**

To achieve the objectives, the following modular approach was adopted:

1. **Process Generation**
   * A C program (generate\_process.c) generates 15–20 processes with random **arrival** and **burst times**.
   * These processes are saved to a file named processes.txt in a structured format.
2. **Process Reading**
   * A separate module reads the processes.txt file and stores the data in an array of Process structures for processing.
3. **Scheduling Logic**
   * The **Round Robin algorithm** is implemented with a fixed **time quantum** (e.g., 3 units).
   * The algorithm keeps track of process execution, manages a queue of ready processes, and updates remaining time, completion time, etc.
4. **Performance Calculation**
   * After all processes are executed, the program calculates:
     + **Waiting Time** = Turnaround Time - Burst Time
     + **Turnaround Time** = Completion Time - Arrival Time
     + **Average Waiting and Turnaround Times**
     + **Throughput** = Total Processes / Total Time
5. **Output**
   * A detailed table is printed showing process statistics.
   * The execution order is displayed as a Gantt chart.
   * Average metrics and throughput are calculated and printed.

**Implementation**

The implementation of the **HUSJ** project is divided into two major parts for clarity and modularity:

**1. Process Generation Module (generate\_process.c)**

This module is responsible for generating a set of 15–20 random processes and writing them to a file named processes.txt. Each process contains:

* **PID (Process ID)**
* **Arrival Time** (random between 0–9)
* **Burst Time** (random between 1–10)

**2. Round Robin Scheduler (main.c)**

This module reads processes from the file, stores them in a structure, and simulates the Round Robin Scheduling algorithm using a fixed **Time Quantum** (e.g., 3 units).

**Key Functions:**

* readProcesses()  
  Reads processes from processes.txt and loads them into memory.
* roundRobinScheduling()  
  Implements Round Robin logic and computes waiting time, turnaround time, and completion time.
* printResults()  
  Displays the Gantt chart, process table, average waiting time, turnaround time, and throughput.

**Core Structures and Logic:**

typedef struct {

int pid;

int arrival\_time;

int burst\_time;

int remaining\_time;

int completion\_time;

int waiting\_time;

int turnaround\_time;

} Process;

**Execution Flow:**

gcc generate\_process.c -o generate

./generate

1. Run the process generator:

gcc main.c -o scheduler

./scheduler

**Modularity Benefits:**

* **Code Reusability**: Functions can be reused in other scheduling algorithms.
* **Scalability**: Easy to change the number of processes or time quantum.
* **Maintenance**: Clear separation of logic makes debugging easier.

**Results**

After running the simulation, the following results were typically observed:

* All processes were scheduled fairly, each receiving equal CPU time slices in a cyclic order.
* The calculated **average waiting time** and **average turnaround time** reflected the effect of context switching.
* The **Gantt chart** effectively visualized how CPU time was allocated to different processes.
* The **throughput** demonstrated how efficiently the CPU handled the given workload.

A black screen with a black background

AI-generated content may be incorrect.

**Conclusion**

The **HUSJ** project successfully demonstrates the practical implementation of the Round Robin CPU scheduling algorithm using process data stored in a file. It highlights the balance between fairness and efficiency in time-sharing systems. By modularizing the code, the project becomes more maintainable and extensible for educational and research purposes.

This simulation provides valuable insights into process scheduling behavior, reinforces core operating system concepts, and develops skills in C programming, file I/O, and algorithm design. Future improvements could include GUI integration, support for multi-core simulation, or comparisons with other algorithms like SJF and Priority Scheduling.

**References**

1. Silberschatz, A., Galvin, P. B., & Gagne, G. (2020). *Operating System Concepts* (10th ed.). Wiley.
2. Tanenbaum, A. S., & Bos, H. (2015). *Modern Operating Systems* (4th ed.). Pearson.
3. Stallings, W. (2018). *Operating Systems: Internals and Design Principles* (9th ed.). Pearson.
4. Robbins, K. A., & Robbins, S. (2003). *UNIX Systems Programming: Communication, Concurrency, and Threads*. Prentice Hall.
5. GeeksforGeeks. (n.d.). Round Robin Scheduling. Retrieved May 2025.
6. TutorialsPoint. (n.d.). *Operating System - Round Robin Scheduling*. Retrieved from https://www.tutorialspoint.com/operating\_system/os\_process\_scheduling.htm
7. Baeldung on C. (2023). *Process Scheduling Algorithms in C*. Retrieved from https://www.baeldung.com/cs/process-scheduling-c
8. Linux Manual Pages. (n.d.). *man 7 sched*. Retrieved from https://man7.org/linux/man-pages/man7/sched.7.html