Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

Labreport - III

[Code No.: COMP 307]

Operating System

Submitted by

Shreyash Mahato

Roll No: 33

Submitted to

Ms. Rabina Shrestha

Department of Computer Science and Engineering

January 11, 2026

1. Introduction

An operating system must manage multiple processes efficiently, and process scheduling plays a vital role in this task. Scheduling determines how CPU time is distributed among running processes. Since different systems have different goals such as fairness, speed, or responsiveness, multiple scheduling algorithms exist. This lab work focuses on understanding and comparing popular CPU scheduling techniques through implementation and analysis.

2. Basic Concepts

2.1. Preemptive Scheduling

Preemptive scheduling allows the operating system to pause a currently running process and assign the CPU to another process when required. This usually happens when a process with higher priority or shorter execution time enters the system.

Advantages:

1. Improves system responsiveness
2. Suitable for time-sharing systems
3. Ensures better CPU utilization
4. Reduces long waiting times

Disadvantages:

1. Frequent context switching increases overhead
2. Implementation is more complex
3. Can lead to priority-related issues

Examples: SRTF, Round Robin

2.2. Non-Preemptive Scheduling

In non-preemptive scheduling, once a process starts executing, it continues until completion without interruption. Other processes must wait until the CPU is released.

Advantages:

1. Easy to design and implement
2. Minimal scheduling overhead
3. Efficient for batch processing

Disadvantages:

1. Poor response time
2. Convoy effect may occur
3. Starvation is possible

Examples: FCFS, SJF

3. CPU Scheduling Algorithms

3.1. Shortest Job First (SJF)

3.1.1. Description

Shortest Job First is a non-preemptive scheduling algorithm where the process with the smallest CPU burst time is executed first. This method reduces average waiting time when burst times are known in advance.

3.1.2. Execution Steps

1. All ready processes are examined
2. Process with minimum burst time is selected
3. FCFS is used if burst times are equal
4. Selected process runs until completion

3.1.3. Time Calculations

$$TAT = CT - AT \tag{1}$$

$$WT = TAT - BT \tag{2}$$

$$AWT = \frac{\sum WT}{n} \tag{3}$$

$$ATAT = \frac{\sum TAT}{n} \tag{4}$$

3.1.4. Program and Output

```c
#include <stdio.h>
#include <stdlib.h>

struct Process {
    int pid;
    int burst_time;
    int waiting_time;
    int turnaround_time;
};

int compare(const void *a, const void *b) {
    return ((struct Process *)a)->burst_time - ((struct Process *)b)->burst_time;
}

int main() {
    int n, i;
    struct Process p[20];
    int total_wt = 0, total_tat = 0;

    printf("Enter number of processes: ");
    scanf("%d", &n);

    // Input burst times
    for (i = 0; i < n; i++) {
        p[i].pid = i + 1;
        printf("Enter Burst Time for Process %d: ", i + 1);
        scanf("%d", &p[i].burst_time);
    }

    // Sort by burst time
    qsort(p, n, sizeof(struct Process), compare);

    // Calculate waiting time
    p[0].waiting_time = 0;
    for (i = 1; i < n; i++) {
        p[i].waiting_time = p[i-1].waiting_time + p[i-1].burst_time;
    }

```

Figure 1: SJF Program

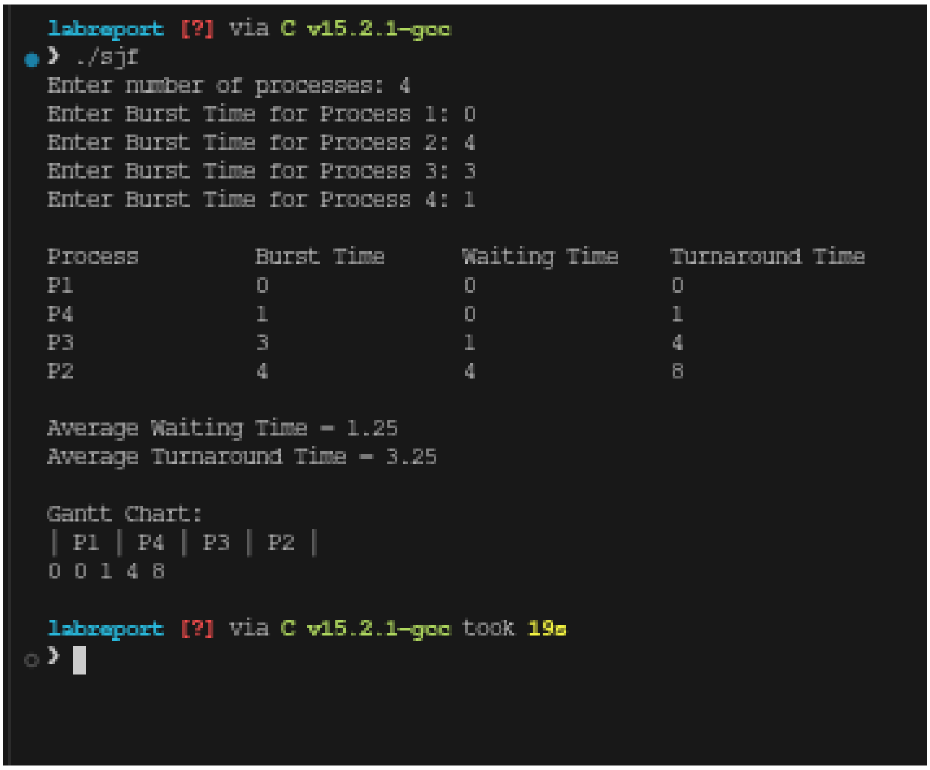


Figure 2: SJF Output

3.2. Shortest Remaining Time First (SRTF)

3.2.1. Description

SRTF is the preemptive form of SJF. At any moment, the CPU is assigned to the process with the least remaining execution time. New processes can interrupt running ones if they require less time.

3.2.2. Execution Steps

1. CPU starts with the first arriving process
2. New arrivals are compared with remaining time
3. CPU switches if a shorter job arrives
4. Process completes when remaining time reaches zero

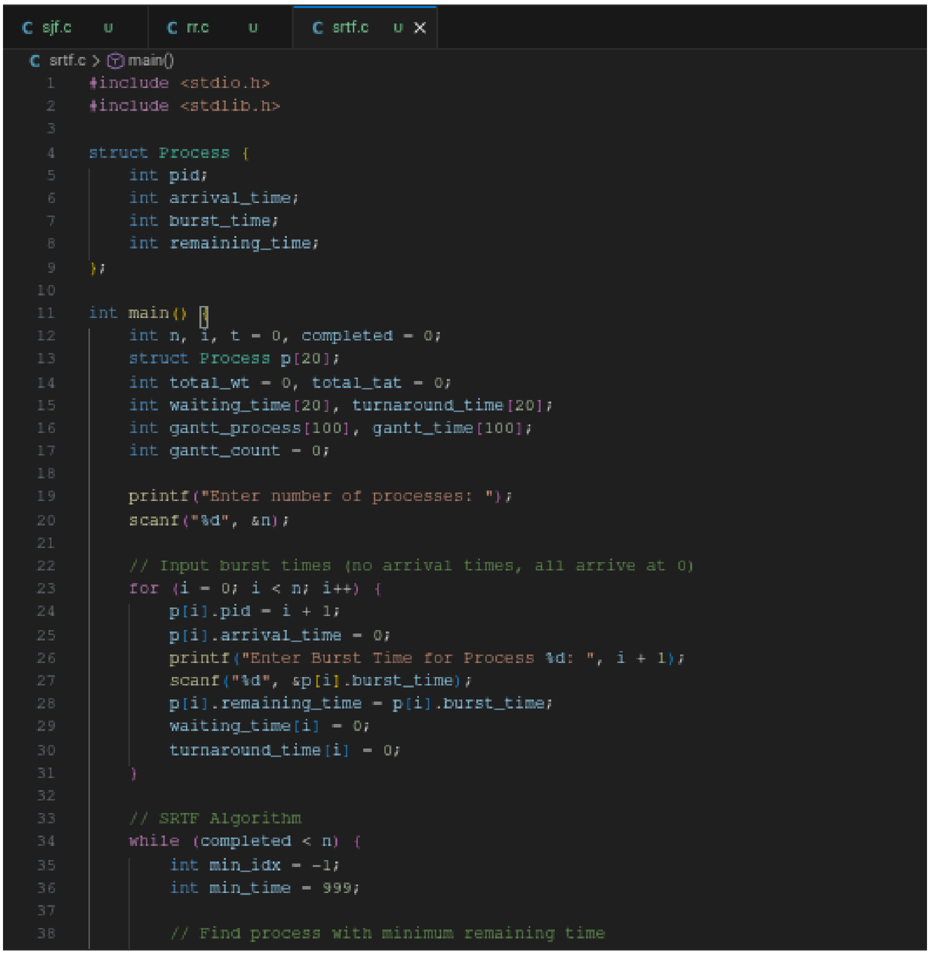
3.2.3. Time Calculations

$$RT = BT - ExecutedTime \tag{5}$$

$$TAT = CT - AT \tag{6}$$

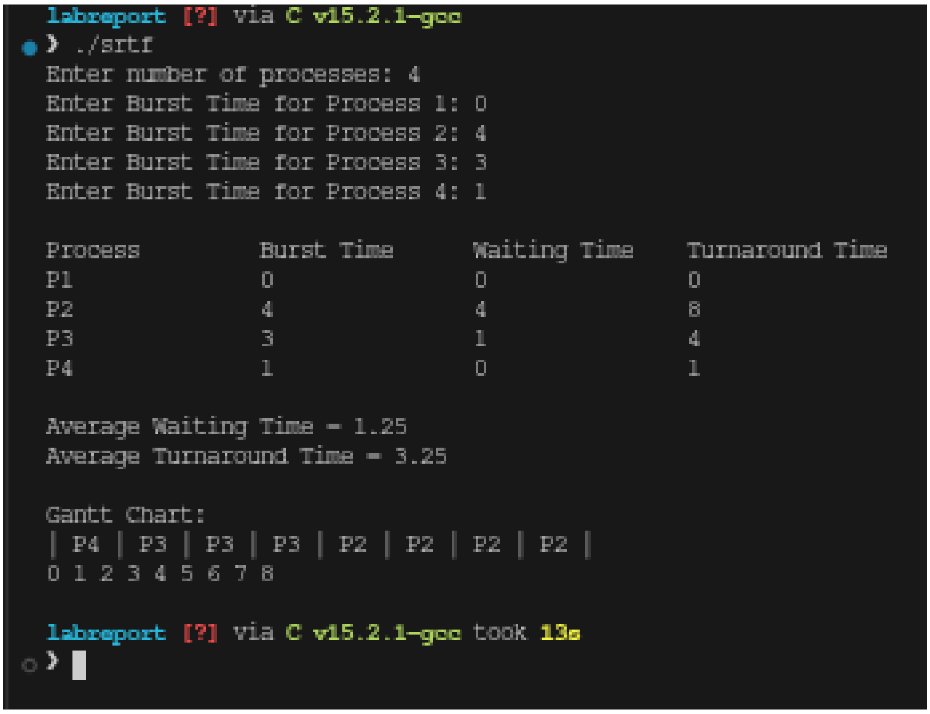
$$WT = TAT - BT \tag{7}$$

3.2.4. Program and Output




```c
#include <stdio.h>
#include <stdlib.h>

struct Process {
    int pid;
    int arrival_time;
    int burst_time;
    int remaining_time;
};

int main() {
    int n, i, t = 0, completed = 0;
    struct Process p[20];
    int total_wt = 0, total_tat = 0;
    int waiting_time[20], turnaround_time[20];
    int gantt_process[100], gantt_time[100];
    int gantt_count = 0;

    printf("Enter number of processes: ");
    scanf("%d", &n);

    // Input burst times (no arrival times, all arrive at 0)
    for (i = 0; i < n; i++) {
        p[i].pid = i + 1;
        p[i].arrival_time = 0;
        printf("Enter Burst Time for Process %d: ", i + 1);
        scanf("%d", &p[i].burst_time);
        p[i].remaining_time = p[i].burst_time;
        waiting_time[i] = 0;
        turnaround_time[i] = 0;
    }

    // SRTF Algorithm
    while (completed < n) {
        int min_idx = -1;
        int min_time = 999;

        // Find process with minimum remaining time
```

Figure 3: SRTF Program



```
labreport [?] via C v15.2.1-gcc
❯ ./srtf
Enter number of processes: 4
Enter Burst Time for Process 1: 0
Enter Burst Time for Process 2: 4
Enter Burst Time for Process 3: 3
Enter Burst Time for Process 4: 1

Process         Burst Time      Waiting Time    Turnaround Time
P1              0               0               0
P2              4               4               8
P3              3               1               4
P4              1               0               1

Average Waiting Time = 1.25
Average Turnaround Time = 3.25

Gantt Chart:
| P4 | P3 | P3 | P3 | P2 | P2 | P2 | P2 |
0 1 2 3 4 5 6 7 8

labreport [?] via C v15.2.1-gcc took 13s
❯
```

Figure 4: SRTF Output

3.3. Round Robin Scheduling

3.3.1. Description

Round Robin scheduling is designed for multi-user systems. Each process is given a fixed time slice called the time quantum. This ensures equal CPU access and prevents starvation.

3.3.2. Execution Steps

1. Processes are stored in a circular queue
2. Each process executes for one time quantum
3. Incomplete processes rejoin the queue
4. Execution continues until all processes finish

3.3.3. Important Relations

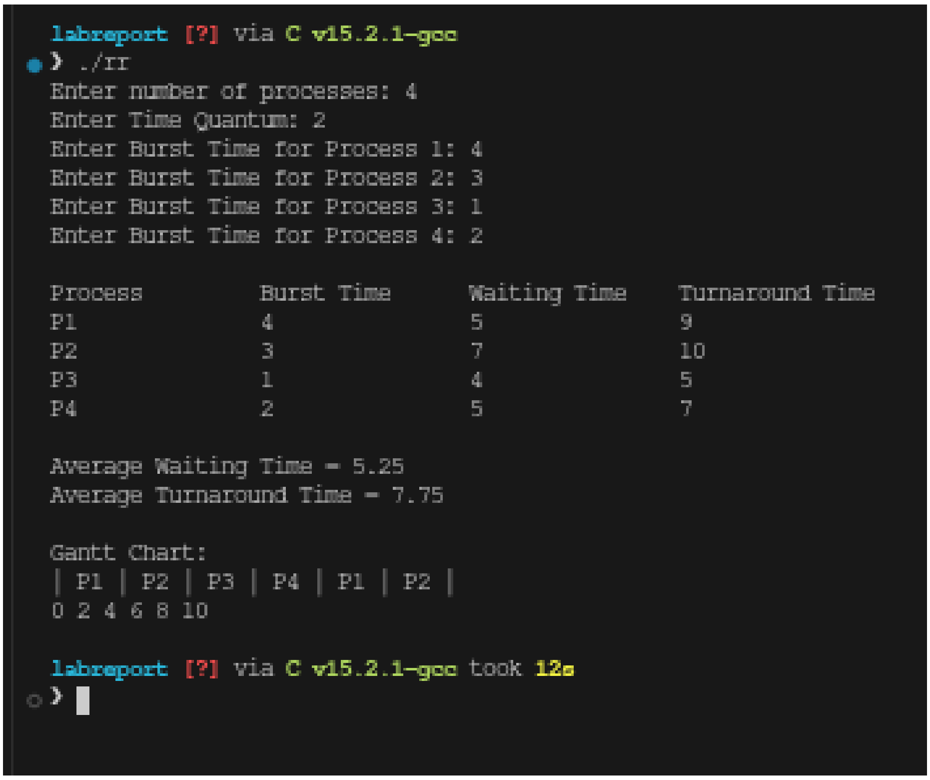
$$ET = \min(TQ, RT) \tag{8}$$

$$WT = TAT - BT \tag{9}$$

3.3.4. Program and Output

```
C sjf.c U    C rr.c U ×    C srtf.c U
C rr.c > Process
#include <stdio.h>
#include <stdlib.h>

struct Process {
    int pid;
    int burst_time;
    int remaining_time;
};

int main() {
    int n, i, t = 0, completed = 0;
    int quantum;
    struct Process p[20];
    int total_wt = 0, total_tat = 0;
    int waiting_time[20], turnaround_time[20];
    int gantt_process[100], gantt_time[100];
    int gantt_count = 0;

    printf("Enter number of processes: ");
    scanf("%d", &n);

    printf("Enter Time Quantum: ");
    scanf("%d", &quantum);

    // Input burst times
    for (i = 0; i < n; i++) {
        p[i].pid = i + 1;
        printf("Enter Burst Time for Process %d: ", i + 1);
        scanf("%d", &p[i].burst_time);
        p[i].remaining_time = p[i].burst_time;
        waiting_time[i] = 0;
        turnaround_time[i] = 0;
    }

    // Round Robin Algorithm
    int head = 0;
    while (completed < n) {
        if (p[head].remaining_time > 0) {
```

Figure 5: Round Robin Program



```
labreport [?] via C v15.2.1-gcc
❯ ./rr
Enter number of processes: 4
Enter Time Quantum: 2
Enter Burst Time for Process 1: 4
Enter Burst Time for Process 2: 3
Enter Burst Time for Process 3: 1
Enter Burst Time for Process 4: 2

Process         Burst Time      Waiting Time    Turnaround Time
P1              4               5               9
P2              3               7               10
P3              1               4               5
P4              2               5               7

Average Waiting Time = 5.25
Average Turnaround Time = 7.75

Gantt Chart:
| P1 | P2 | P3 | P4 | P1 | P2 |
0 2 4 6 8 10

labreport [?] via C v15.2.1-gcc took 12s
❯
```

Figure 6: Round Robin Output

4. Algorithm Comparison

Aspect	SJF	SRTF	RR
Preemption	No	Yes	Yes
Waiting Time	Low	Very Low	Moderate
Fairness	No	No	Yes
Context Switching	Low	High	High
Starvation	Possible	Possible	No

5. Conclusion

From this experiment, it is clear that no single scheduling algorithm is perfect. SJF performs well for minimizing waiting time but lacks flexibility. SRTF improves response time at the cost of overhead. Round Robin ensures fairness but increases context switching. Therefore, the choice of scheduling algorithm depends on system requirements and workload type.