Department of Computer Science & Engineering

Course Title: Operating System Lab

Course Code: CSE 406

Lab Report No: Mid Q1

Lab Report: Shortest Job First (SJF) Scheduling

Submission Date: 21-04-2025

Submitted To:	**Submitted By:**
Atia Rahman Orthi	**Name: Amirul Islam Papon**
Lecturer,	**Reg No: 21201076**
Department of CSE, UAP	**Sec: B**

Objective:

To implement the Shortest Job First (SJF) CPU scheduling algorithm and evaluate its performance based on waiting time, turnaround time, and completion time.

Theory:

Shortest Job First (SJF) is a non-preemptive scheduling algorithm where the CPU is assigned to the process with the shortest burst time among the available processes.

- It minimizes the average waiting time.
- It is optimal for batch systems with known burst times.
- If two processes have the same burst time, the one that arrives first is scheduled first.

Key Terms:

- **Arrival Time**: Time when a process enters the ready queue.
- **Burst Time**: Time required by the process to execute.
- **Completion Time**: Time at which the process finishes execution.
- **Turnaround Time** = Completion Time – Arrival Time
- **Waiting Time** = Turnaround Time – Burst Time

Algorithm (Steps):

1. Sort the processes based on arrival time.
2. Select the process with the shortest burst time from the available processes at the current time.
3. Execute the selected process to completion.
4. Repeat steps 2–3 until all processes are completed.
5. Calculate and display waiting time, turnaround time, and completion time for each process.

Solve:

Q1:-

Pid	AT	BT	CT	TAT	WT
1	2	4	11	9	5
2	10	1	12	2	1
3	3	2	7	4	2
4	2	5	17	15	10
5	0	5	5	5	0

P5	P3	P1	P2	P4

0 5 7 11 12 17

Code:

sjf_mid_test.py > ...

```python
def sjf_scheduling(processes):
    n = len(processes)
    completed = 0
    current_time = 0
    is_done = [False] * n

    waiting = [0] * n
    turnaround = [0] * n
    completion = [0] * n
    start = [0] * n

    while completed < n:
        idx = -1
        shortest = float('inf')
        for i in range(n):
            pid, arrival, burst = processes[i]
            if arrival <= current_time and not is_done[i]:
                if burst < shortest:
                    shortest = burst
                    idx = i

        if idx == -1:
            current_time += 1
        else:
            pid, arrival, burst = processes[idx]
            start[idx] = current_time
            completion[idx] = current_time + burst
            turnaround[idx] = completion[idx] - arrival
            waiting[idx] = turnaround[idx] - burst
            current_time = completion[idx]
            is_done[idx] = True
            completed += 1

    print("PID\tArrival\tBurst\tCompletion\tWaiting\tTurnaround")
    for i in range(n):
        pid, arrival, burst = processes[i]
        print(f"{pid}\t{arrival}\t\t{burst}\t\t{completion[i]}\t\t\t{waiting[i]}\t\t{turnaround[i]}")

process_list = [(1, 2, 4), (2, 10, 1), (3, 3, 2), (4, 2, 5), (5, 0, 5)]
sjf_scheduling(process_list)
```

Output:

```
[Running] python -u "d:\Operating-System\sjf_mid_test.py"
PID Arrival Burst   Completion  Waiting Turnaround
1   2       4       11          5       9
2   10      1       12          1       2
3   3       2       7           2       4
4   2       5       17          10      15
5   0       5       5           0       5

[Done] exited with code=0 in 0.048 seconds
```

Discussion and Conclusion:

The SJF scheduling algorithm efficiently reduces the average waiting time and turnaround time, making it an optimal scheduling technique for batch processing systems. However, SJF has limitations:

- Starvation Issue: Longer processes may suffer from indefinite waiting if shorter jobs keep arriving.
- Non-Preemptive Limitation: Once a process starts execution, it cannot be interrupted, which may not be ideal for real-time systems.

GithubLink: https://github.com/Amirul-Islam-Papon/Operating-System/blob/main/sjf_mid_test.py