CPU Scheduling Algorithms

Objectives:

- SRTF reduces waiting time by prioritizing processes with the shortest remaining time. This helps to minimize the overall system delay, especially for short tasks.

- RR ensures fairness by allocating a fixed time slice to each process in a cyclic manner, preventing any process from starving or monopolizing the CPU.

- SRTF ensures that the CPU is used efficiently by executing processes with the shortest remaining time.

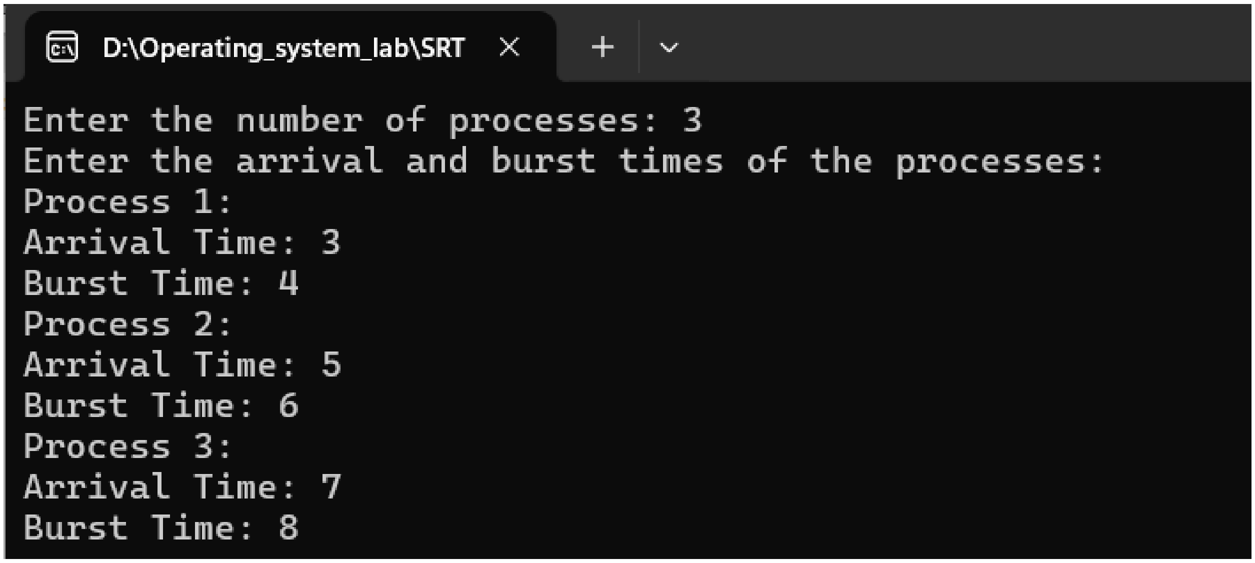
- RR maximizes CPU utilization by ensuring each process gets a fair share of CPU time and no process is left waiting indefinitely.

01.SRTF (Shortest Remaining Time First) Scheduling Algorithm.

Code:

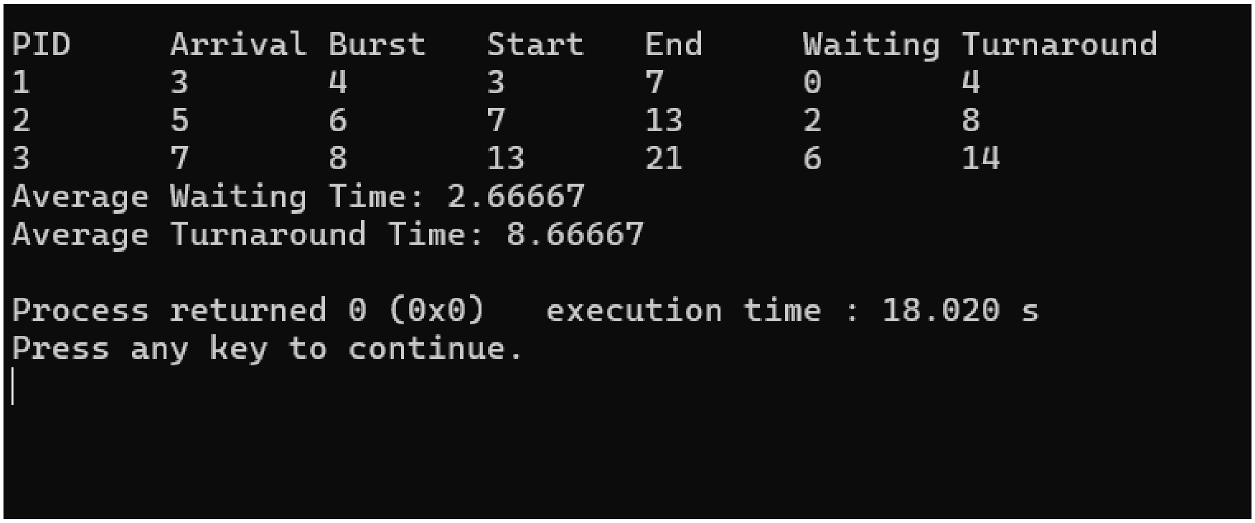
```
#include <iostream>
#include <climits>
using namespace std;

void SRT(int burstTimes[], int arrivalTimes[], int n) {
    int remainingTime[n], startTime[n], endTime[n], completionTime[n];
    int waitingTime[n], turnaroundTime[n];
    int currentTime = 0, completed = 0, minProcess = -1;
    bool isFirstExecution[n] = {false};
    int totalWaitingTime = 0, totalTurnaroundTime = 0;

    // Initialize remaining times and start times
    for (int i = 0; i < n; i++) {
        remainingTime[i] = burstTimes[i];
        startTime[i] = -1;
    }

    // Process until all processes are completed
    while (completed != n) {
        // Find the process with the shortest remaining time at current time
        int minTime = INT_MAX;
        for (int i = 0; i < n; i++) {
            if (arrivalTimes[i] <= currentTime && remainingTime[i] > 0 && remainingTime[i] < minTime) {
                minTime = remainingTime[i];
                minProcess = i;
            }
        }

        if (minProcess == -1) {
            currentTime++; // Idle time
            continue;
        }

        // Start time for the process (only set once)
        if (!isFirstExecution[minProcess]) {
            startTime[minProcess] = currentTime;
            isFirstExecution[minProcess] = true;
        }

        // Process execution for 1 unit of time
        remainingTime[minProcess]--;
        currentTime++;

        // If the process finishes
        if (remainingTime[minProcess] == 0) {
            completed++;
            completionTime[minProcess] = currentTime;
            endTime[minProcess] = currentTime;

            // Calculate waiting and turnaround times
            turnaroundTime[minProcess] = completionTime[minProcess] - arrivalTimes[minProcess];
            waitingTime[minProcess] = turnaroundTime[minProcess] - burstTimes[minProcess];
            totalWaitingTime += waitingTime[minProcess];
            totalTurnaroundTime += turnaroundTime[minProcess];
        }
    }

    // Output results
    cout << "\nPID\tArrival\tBurst\tStart\tEnd\tWaiting\tTurnaround\n";
    for (int i = 0; i < n; i++) {
        cout << i + 1 << "\t" << arrivalTimes[i] << "\t" << burstTimes[i] << "\t"
             << startTime[i] << "\t" << endTime[i] << "\t" << waitingTime[i] << "\t" <<
turnaroundTime[i] << "\n";
    }

    cout << "Average Waiting Time: " << (float)totalWaitingTime / n << "\n";
    cout << "Average Turnaround Time: " << (float)totalTurnaroundTime / n << "\n";
}

int main() {
    int n;

    cout << "Enter the number of processes: ";
    cin >> n;

    int burstTimes[n], arrivalTimes[n];
    cout << "Enter the arrival and burst times of the processes:\n";
    for (int i = 0; i < n; i++) {
        cout << "Process " << i + 1 << ":\n";
        cout << "Arrival Time: ";
        cin >> arrivalTimes[i];
        cout << "Burst Time: ";
        cin >> burstTimes[i];
    }

    SRT(burstTimes, arrivalTimes, n);

    return 0;
}
```

Input:



```
Enter the number of processes: 3
Enter the arrival and burst times of the processes:
Process 1:
Arrival Time: 3
Burst Time: 4
Process 2:
Arrival Time: 5
Burst Time: 6
Process 3:
Arrival Time: 7
Burst Time: 8
```

Output:



```
PID     Arrival Burst   Start   End     Waiting Turnaround
1       3       4       3       7       0       4
2       5       6       7       13      2       8
3       7       8       13      21      6       14
Average Waiting Time: 2.66667
Average Turnaround Time: 8.66667

Process returned 0 (0x0)   execution time : 18.020 s
Press any key to continue.
```

02. RR (Round Robin) Scheduling Algorithm.

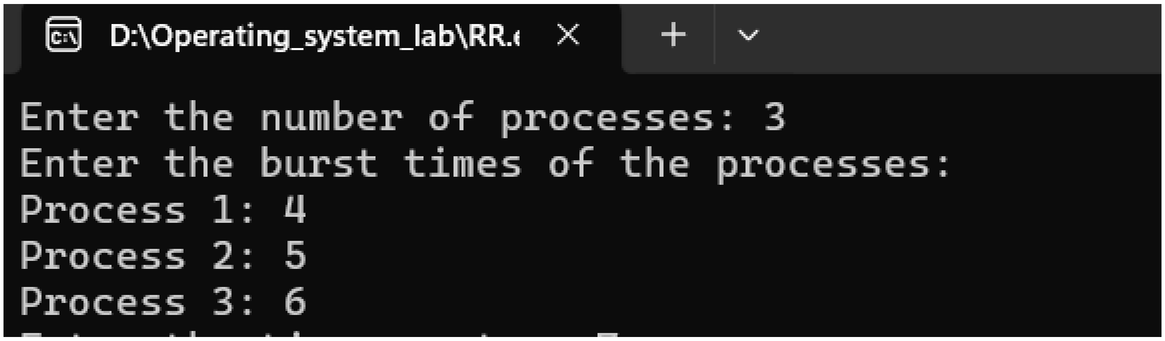
Code:

```
#include <iostream>
#include <queue>
using namespace std;

void RoundRobin(int burstTimes[], int n, int quantum) {
    int remainingTime[n], startTime[n], endTime[n], waitingTime[n], turnaroundTime[n];
    queue<int> q;
    int currentTime = 0;
    bool isFirstExecution[n] = {false};
    int totalWaitingTime = 0, totalTurnaroundTime = 0;

    // Initialize remaining times
    for (int i = 0; i < n; i++) {
        remainingTime[i] = burstTimes[i];
        startTime[i] = -1;
        q.push(i); // Add all processes to the queue
    }

    // Process execution in round-robin fashion
    while (!q.empty()) {
        int i = q.front();
        q.pop();

        // If it's the first execution of a process, record the start time
        if (!isFirstExecution[i]) {
```

```
        startTime[i] = currentTime;
        isFirstExecution[i] = true;
    }

    // Execute the process for the time quantum or until completion
    int executionTime = min(remainingTime[i], quantum);
    remainingTime[i] -= executionTime;
    currentTime += executionTime;

    // If the process has finished, calculate its end time, waiting, and turnaround times
    if (remainingTime[i] == 0) {
        endTime[i] = currentTime;
        turnaroundTime[i] = endTime[i] - startTime[i];
        waitingTime[i] = turnaroundTime[i] - burstTimes[i];
        totalWaitingTime += waitingTime[i];
        totalTurnaroundTime += turnaroundTime[i];
    } else {
        q.push(i); // If the process is not finished, add it back to the queue
    }
}

// Output results
cout << "\nPID\tBurst\tStart\tEnd\tWaiting\tTurnaround\n";
for (int i = 0; i < n; i++) {
    cout << i + 1 << "\t" << burstTimes[i] << "\t"
         << startTime[i] << "\t" << endTime[i] << "\t"
         << waitingTime[i] << "\t" << turnaroundTime[i] << "\n";
}
```

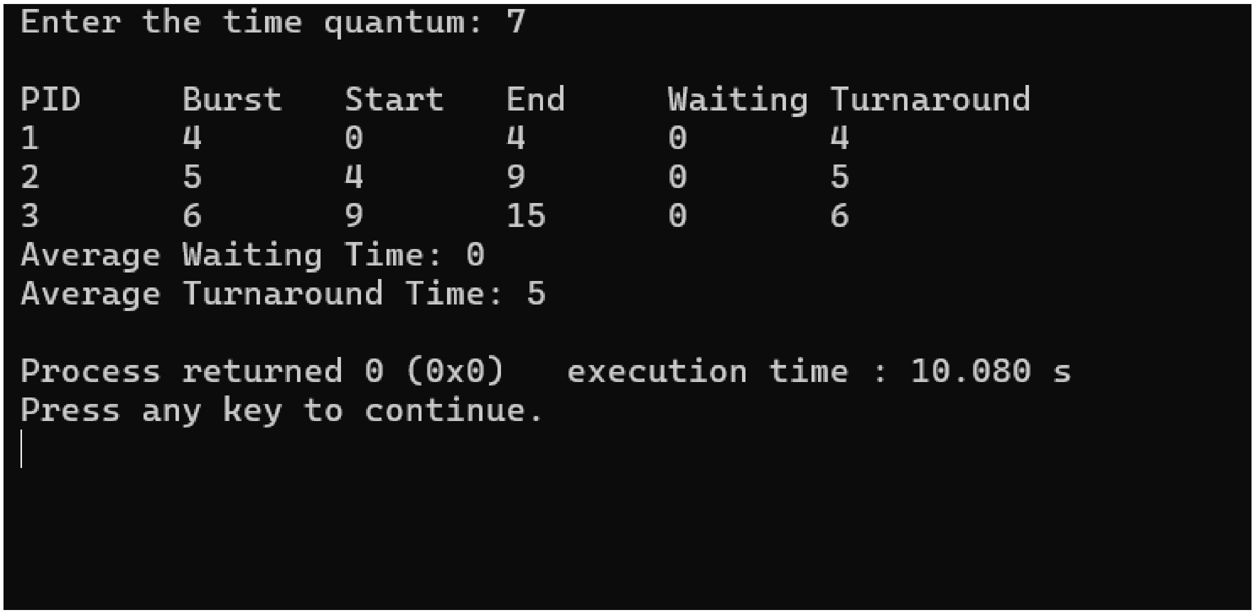
```
    cout << "Average Waiting Time: " << (float)totalWaitingTime / n << "\n";
    cout << "Average Turnaround Time: " << (float)totalTurnaroundTime / n << "\n";
}

int main() {
    int n, quantum;

    cout << "Enter the number of processes: ";
    cin >> n;

    int burstTimes[n];
    cout << "Enter the burst times of the processes:\n";
    for (int i = 0; i < n; i++) {
        cout << "Process " << i + 1 << ": ";
        cin >> burstTimes[i];
    }

    cout << "Enter the time quantum: ";
    cin >> quantum;

    RoundRobin(burstTimes, n, quantum);

    return 0;
}
```

Input:



```
Enter the number of processes: 3
Enter the burst times of the processes:
Process 1: 4
Process 2: 5
Process 3: 6
```

Output:



```
Enter the time quantum: 7

PID     Burst   Start   End     Waiting Turnaround
1       4       0       4       0       4
2       5       4       9       0       5
3       6       9       15      0       6
Average Waiting Time: 0
Average Turnaround Time: 5

Process returned 0 (0x0)   execution time : 10.080 s
Press any key to continue.
```