Experiment - 8

Write C programs to implement the First-Come, First-Served (FCFS) and Shortest Job First (SJF) CPU scheduling algorithms. Test and analyze their performance with different sets of processes.

CPU scheduling algorithms

Definition: Technique in which a process is executed by using resources of the CPU.

Basic Terminologies:

- **Process ID**
- **Arrival Time (AT):** The time which is required for the Process to enter the ready queue or the time when the Process is ready to be executed by the CPU.
- **Burst Time (BT):** The Time Slot which the Process requires to complete the Process is known as the Burst Time.
- **Completion Time (CT):** The Total Time required by the CPU to complete the process is known as Completion Time.

- **Turn Around Time (TAT):** The time taken by the CPU since the Process has been ready to execute or since the process is in Ready Queue is known as Turn Around Time.

$$TAT = CT - AT$$

- **Waiting Time (WT):** The time the Process has been waiting to complete its process since the assignment of process for completion is known as Waiting Time.

$$WT = TAT - BT$$

- **Gantt Chart:** It is the place where all the already executed processes are stored. This is very useful for calculating Waiting Time, Completion Time, Turn Around Time.

Modes in CPU Scheduling Algorithms:

- Pre-emptive Approach
- Non Pre-emptive Approach

Types of CPU Scheduling Algorithms:

1. First Come First Serve (FCFS)
2. Shortest Job First (SJF)
3. Round Robin Scheduling (RR)

Algorithm for First-Come, First-Served (FCFS) Scheduling

1. **Input the Number of Processes:**

   Prompt the user to enter the total number of processes n.

2. **Declare Process Structure:**

   - Define a structure Process with the following attributes:
     - id: A unique identifier for each process.
     - burstTime: The execution time required by the process.
     - waitingTime: The time the process spends waiting before it starts execution.
     - turnAroundTime: The total time from process arrival to completion.

3. **Input Burst Time for Each Process:**

   For each process, take input for its burst time and assign it an id.

4. Calculate Waiting Time for Each Process:

- The first process in the list starts immediately, so its waiting time is 0.
- For every subsequent process:

  Its waiting time is the sum of the waiting time and burst time of the previous process.

5. Calculate Turnaround Time for Each Process:

For each process, calculate its turnaround time as the sum of its burst time and waiting time.

6. Print Process Details:

- Display a table showing:
  - Process ID.
  - Burst Time.
  - Waiting Time.
  - Turnaround Time.

7. Calculate and Print Average Waiting and Turnaround Times:

- Calculate the total waiting time and total turnaround time by summing the respective times for all processes.
- Compute the average waiting time and average turnaround time by dividing the total times by the number of processes.
- Print the average waiting time and average turnaround time.

```c
#include <stdio.h>

struct Process {
    int id;
    int burstTime;
    int waitingTime;
    int turnAroundTime;
};

void findWaitingTime(struct Process processes[], int n) {
    processes[0].waitingTime = 0;  // First process has no waiting time

    for (int i = 1; i < n; i++) {
        processes[i].waitingTime = processes[i - 1].waitingTime + processes[i - 1].burstTime;
    }
}

void findTurnAroundTime(struct Process processes[], int n) {
    for (int i = 0; i < n; i++) {
        processes[i].turnAroundTime = processes[i].burstTime + processes[i].waitingTime;
    }
}
```

```c
void findAvgTime(struct Process processes[], int n) {
    int totalWaitingTime = 0, totalTurnAroundTime = 0;

    findWaitingTime(processes, n);
    findTurnAroundTime(processes, n);

    printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");

    for (int i = 0; i < n; i++) {
        totalWaitingTime += processes[i].waitingTime;
        totalTurnAroundTime += processes[i].turnAroundTime;
        printf("%d\t\t%d\t\t%d\t\t%d\n", processes[i].id, processes[i].burstTime,
               processes[i].waitingTime, processes[i].turnAroundTime);
    }

    printf("\nAverage Waiting Time = %.2f\n", (float) totalWaitingTime / n);
    printf("Average Turnaround Time = %.2f\n", (float) totalTurnAroundTime / n);
}
```

```c
int main() {
    int n;

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

    struct Process processes[n];
    for (int i = 0; i < n; i++) {
        processes[i].id = i + 1;
        printf("Enter burst time for process %d: ", i + 1);
        scanf("%d", &processes[i].burstTime);
    }

    printf("\nFirst-Come, First-Served (FCFS) Scheduling Algorithm:\n");
    findAvgTime(processes, n);

    return 0;
}
```

```
Enter the number of processes: 3
Enter burst time for process 1: 4
Enter burst time for process 2: 5
Enter burst time for process 3: 2

First-Come, First-Served (FCFS) Scheduling Algorithm:
Process Burst Time      Waiting Time    Turnaround Time
1               4               0               4
2               5               4               9
3               2               9               11

Average Waiting Time = 4.33
Average Turnaround Time = 8.00
```

Algorithm for the Shortest Job First (SJF) Scheduling Program

1. **Input the Number of Processes:**

   Prompt the user to enter the total number of processes n.

2. **Declare Process Structure:**

   - Define a structure Process with the following attributes:
     - id: A unique identifier for each process.
     - burstTime: The execution time required by the process.
     - waitingTime: The time the process spends waiting before it starts execution.
     - turnAroundTime: The total time from process arrival to completion.

3. **Input Burst Time for Each Process:**

   For each process, take input for its burst time and assign it an id.

4. Sort Processes by Burst Time (Non-preemptive):

- Sort the processes in ascending order based on their burst times:
    - For each pair of processes, compare their burst times.
    - If the burst time of the current process is greater than the next, swap them.
    - Repeat this process until the entire list is sorted.

5. **Calculate Waiting Time for Each Process:**

- The first process in the list starts immediately, so its waiting time is 0.
- For every subsequent process:

    Its waiting time is the sum of the waiting time and burst time of the previous process.

6. Calculate Turnaround Time for Each Process:

For each process, calculate its turnaround time as the sum of its burst time and waiting time.

6. Print Process Details:

- Display a table showing:
  - Process ID.
  - Burst Time.
  - Waiting Time.
  - Turnaround Time

7. Calculate and Print Average Waiting and Turnaround Times:

- Calculate the total waiting time and total turnaround time by summing the respective times for all processes.
- Compute the average waiting time and average turnaround time by dividing the total times by the number of processes.
- Print the average waiting time and average turnaround time.

```c
#include <stdio.h>

struct Process {
    int id;
    int burstTime;
    int waitingTime;
    int turnAroundTime;
};

// Sort processes by burst time (non-preemptive)
void sortProcesses(struct Process processes[], int n) {
    struct Process temp;
    for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (processes[i].burstTime > processes[j].burstTime) {
                temp = processes[i];
                processes[i] = processes[j];
                processes[j] = temp;
            }
        }
    }
}

void findWaitingTime(struct Process processes[], int n) {
    processes[0].waitingTime = 0;

    for (int i = 1; i < n; i++) {
        processes[i].waitingTime = processes[i - 1].waitingTime + processes[i - 1].burstTime;
    }
}
```

```
void findTurnAroundTime(struct Process processes[], int n) {
    for (int i = 0; i < n; i++) {
        processes[i].turnAroundTime = processes[i].burstTime + processes[i].waitingTime;
    }
}

void findAvgTime(struct Process processes[], int n) {
    int totalWaitingTime = 0, totalTurnAroundTime = 0;

    findWaitingTime(processes, n);
    findTurnAroundTime(processes, n);

    printf("Process\tBurst Time\tWaiting Time\tTurnaround Time\n");

    for (int i = 0; i < n; i++) {
        totalWaitingTime += processes[i].waitingTime;
        totalTurnAroundTime += processes[i].turnAroundTime;
        printf("%d\t\t%d\t\t%d\t\t%d\n", processes[i].id, processes[i].burstTime,
               processes[i].waitingTime, processes[i].turnAroundTime);
    }

    printf("\nAverage Waiting Time = %.2f\n", (float) totalWaitingTime / n);
    printf("Average Turnaround Time = %.2f\n", (float) totalTurnAroundTime / n);
}
```

```c
int main() {
    int n;

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

    struct Process processes[n];
    for (int i = 0; i < n; i++) {
        processes[i].id = i + 1;
        printf("Enter burst time for process %d: ", i + 1);
        scanf("%d", &processes[i].burstTime);
    }

    sortProcesses(processes, n);

    printf("\nShortest Job First (SJF) Scheduling Algorithm:\n");
    findAvgTime(processes, n);

    return 0;
}
```

```
Enter the number of processes: 3
Enter burst time for process 1: 4
Enter burst time for process 2: 5
Enter burst time for process 3: 2

Shortest Job First (SJF) Scheduling Algorithm:
Process Burst Time      Waiting Time    Turnaround Time
3               2               0               2
1               4               2               6
2               5               6               11

Average Waiting Time = 2.67
Average Turnaround Time = 6.33
```