Experiment 9

Develop a C program to implement the Round Robin CPU scheduling algorithm with a given time quantum. Analyze its performance with various process inputs.

Algorithm for Round Robin Scheduling:

1. **Input the number of processes**:

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

2. **Initialize process attributes**:

   For each process, initialize the following attributes:

   • *id*: Process ID.

   • *burstTime*: CPU burst time (the amount of time a process requires).

   • *remainingTime*: Initially set to the burstTime (indicating how much time is left for the process to complete).

   • *waitingTime*: Initially set to *0*.

   • *turnAroundTime*: Initially set to *0*.

3. **Input the time quantum**:

   Prompt the user to enter the time quantum, the fixed time slice each process will execute in a round.

4. **Call *findWaitingTime* function:**
    - Initialize *time* to 0.
    - Repeat the following until all processes are finished:
        - For each process in the list:
            - If the remaining burst time of the process is greater than 0:
            - If *remainingTime* > *quantum*, increment the current time (time += quantum) and decrease remainingTime by quantum.
            - If *remainingTime* <= *quantum*, increment the current time by the *remainingTime*, calculate the waiting time (*waitingTime = time - burstTime*), and set *remainingTime = 0* (indicating the process is completed).
            - Continue looping over the processes until all are finished.
5. **Call findTurnAroundTime function:**

    For each process, calculate the turnaround time (*turnAroundTime = burstTime + waitingTime*).

6. **Calculate and display results:**

- Print the process details: *id*, *burstTime*, *waitingTime*, and *turnAroundTime* for each process.
- Calculate and display the average waiting time (*totalWaitingTime / n*) and average turnaround time (*totalTurnAroundTime / n*).

```c
#include <stdio.h>

struct Process {
    int id;             // Process ID
    int burstTime;      // Burst Time (CPU time required)
    int remainingTime; // Remaining Burst Time
    int waitingTime;    // Waiting Time
    int turnAroundTime;// Turnaround Time
};
```

```c
void findWaitingTime(struct Process processes[], int n, int quantum) {
    int time = 0; // Current time

    while (1) {
        int done = 1; // Assume all processes are finished

        // Traverse all processes one by one
        for (int i = 0; i < n; i++) {
            if (processes[i].remainingTime > 0) {
                done = 0; // There is a pending process

                if (processes[i].remainingTime > quantum) {
                    time += quantum;
                    processes[i].remainingTime -= quantum;
                } else {
                    time += processes[i].remainingTime;
                    processes[i].waitingTime = time - processes[i].burstTime;
                    processes[i].remainingTime = 0;
                }
            }
        }

        // If all processes are done, break the loop
        if (done == 1)
            break;
    }
}
```

```c
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 quantum) {
    int totalWaitingTime = 0, totalTurnAroundTime = 0;

    findWaitingTime(processes, n, quantum);
    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, quantum;

    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);
        processes[i].remainingTime = processes[i].burstTime; // Initially, remaining time
    }

    printf("Enter the time quantum: ");
    scanf("%d", &quantum);

    printf("\nRound Robin Scheduling Algorithm:\n");
    findAvgTime(processes, n, quantum);

    return 0;
}
```