Assignment No. 3

Implement the C program for CPU Scheduling Algorithms: Shortest Job First (Preemptive) and Round Robin with different arrival time

A.Shortest Job First (Preemptive)

3_sjf.c

```
#include <stdio.h>
#include <limits.h> // For INT_MAX to represent a large number
#define MAX_PROCESSES 10 // Define the maximum number of processes supported
int main() {
// Arrays to store arrival times, burst times, and temporary burst times
int arrival_time[MAX_PROCESSES], burst_time[MAX_PROCESSES],
temp[MAX_PROCESSES];
int remaining_time[MAX_PROCESSES]; // Array to track the remaining burst times for each
process
int completion_time[MAX_PROCESSES], waiting_time[MAX_PROCESSES],
turnaround_time[MAX_PROCESSES];
int i, smallest, count = 0, time, limit;
int total_wait_time = 0, total_turnaround_time = 0;
float average_waiting_time, average_turnaround_time;
// Prompt the user to enter the number of processes
printf("\nEnter the Total Number of Processes (max %d):\t", MAX_PROCESSES);
scanf("%d", &limit);
// Check if the number of processes exceeds the maximum allowed
if (limit > MAX_PROCESSES) {
printf("Number of processes cannot exceed %d\n", MAX_PROCESSES);
return 1;
}
// Read the arrival and burst times for each process
printf("\nEnter Details of %d Processes\n", limit);
for(i = 0; i < limit; i++) {
printf("\nProcess %d:\n", i + 1);
printf("Enter Arrival Time:\t");
scanf("%d", &arrival_time[i]);
printf("Enter Burst Time:\t");
scanf("%d", &burst_time[i]);
remaining_time[i] = burst_time[i]; // Initialize remaining time with the burst time
temp[i] = burst_time[i]; // Store the original burst time for later calculations
}
int completed[MAX_PROCESSES] = {0}; // Array to keep track of which processes have been
completed// Main loop to simulate time and schedule processes
for(time = 0; count != limit; time++) {
smallest = -1; // Initialize smallest as -1 to indicate no process selected yet
// Find the process with the smallest remaining burst time that has arrived and is not yet
completed
for(i = 0; i < limit; i++) {
if (arrival_time[i] <= time && !completed[i]) {
if (smallest == -1 || remaining_time[i] < remaining_time[smallest]) {
```

```
smallest = i; // Update smallest to the current process with the shortest remaining time
}
}
}
if (smallest != -1) {
remaining_time[smallest]--; // Execute the selected process for one unit of time
if (remaining_time[smallest] == 0) {
// Process is completed
count++;
completed[smallest] = 1; // Mark the process as completed
completion_time[smallest] = time + 1; // Calculate completion time of the process
waiting_time[smallest] = completion_time[smallest] - arrival_time[smallest] -
temp[smallest];
turnaround_time[smallest] = completion_time[smallest] - arrival_time[smallest];
total_wait_time += waiting_time[smallest]; // Accumulate total waiting time
total_turnaround_time += turnaround_time[smallest]; // Accumulate total turnaround
time
}
}
}
// Calculate average waiting time and turnaround time
average_waiting_time = (float)total_wait_time / limit;
average_turnaround_time = (float)total_turnaround_time / limit;
// Print the process details including arrival time, burst time, completion time, waiting time, and
turnaround time
printf("\nProcess No\tAT\tBT\tCT\tTAT\tWT\n");
for(i = 0; i < limit; i++) {
printf("%d\t\t%d\t%d\t%d\t%d\t%d\n", i + 1, arrival_time[i], temp[i], completion_time[i],
turnaround_time[i], waiting_time[i]);
}
// Print the average waiting time and average turnaround time
printf("\nAverage Waiting Time:\t%f", average_waiting_time);
printf("\nAverage Turnaround Time:\t%f\n", average_turnaround_time);
}
return 0;
```

OUTPUT:

```
pl-17@pl17-OptiPlex-3020:~/07IT$ gcc 3_sjf.c
pl-17@pl17-OptiPlex-3020:~/07IT$ ./a.out

Enter the Total Number of Processes (max 10):      4

Enter Details of 4 Processes

Process 1:
Enter Arrival Time:    0
Enter Burst Time:      5

Process 2:
Enter Arrival Time:    1
Enter Burst Time:      3

Process 3:
```

```
Enter Arrival Time:    2
Enter Burst Time:      4

Process 4:
Enter Arrival Time:    4
Enter Burst Time:      1

Process No    AT    BT    CT    TAT   WT
1             0     5     9     9     4
2             1     3     4     3     0
3             2     4     13    11    7
4             4     1     5     1     0

Average Waiting Time:        2.750000
Average Turnaround Time:     6.000000
```

B. Round Robin

3_RR.c

```
#include <stdio.h>

#define MAX_PROCESSES 10  // Define a constant for the maximum number of processes

int main() {
    int i, limit, total_time = 0, time_quantum;
    int wait_time = 0, turnaround_time = 0;
    int completion_time[MAX_PROCESSES];
    int remaining_time[MAX_PROCESSES];
    int completed[MAX_PROCESSES] = {0};
    int arrival_time[MAX_PROCESSES], burst_time[MAX_PROCESSES];
    int remaining_processes = 0;
    float average_wait_time, average_turnaround_time;

    // Input number of processes
    printf("Enter Total Number of Processes (max %d):\n\t", MAX_PROCESSES);
    scanf("%d", &limit);

    // Input arrival and burst times for each process
    for (i = 0; i < limit; i++) {
        printf("Enter Details of Process[%d]\n", i + 1);

        printf("Arrival Time:\t");
        scanf("%d", &arrival_time[i]);

        printf("Burst Time:\t");
        scanf("%d", &burst_time[i]);

        remaining_time[i] = burst_time[i];  // Initialize remaining times
    }
    // Input time quantum
    printf("Enter Time Quantum:\n\t");
    scanf("%d", &time_quantum);

    printf("\nProcess ID\tArrival Time\tBurst Time\tCompletion Time\tTurnaround Time\tWaiting Time\n");

    // Main Round Robin scheduling loop
    int process_index = 0;
    for (remaining_processes = limit; remaining_processes > 0;) {
        if (remaining_time[process_index] > 0) {
            if (remaining_time[process_index] <= time_quantum) {
                total_time += remaining_time[process_index];
                remaining_time[process_index] = 0;
            } else {
                remaining_time[process_index] -= time_quantum;
                total_time += time_quantum;
            }

            // Check if the process is complete
            if (remaining_time[process_index] == 0) {
                remaining_processes--;
                completion_time[process_index] = total_time;
                int turnaround = completion_time[process_index] - arrival_time[process_index];
                int wait = turnaround - burst_time[process_index];
                wait_time += wait;
                turnaround_time += turnaround;

                // Print process details
                printf("Process[%d]\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n", process_index + 1, arrival_time[process_index], burst_time[process_index], completion_time[process_index], turnaround, wait);
            }
        }

        // Move to the next process
        process_index = (process_index + 1) % limit;

        // Ensure that we don't process the process which hasn't arrived yet
        while (arrival_time[process_index] > total_time) {
            process_index = (process_index + 1) % limit;
        }
    }

    // Calculate average waiting time and turnaround time
    average_wait_time = (float)wait_time / limit;
    average_turnaround_time = (float)turnaround_time / limit;

    // Print average times
    printf("\nAverage Waiting Time:\t%f", average_wait_time);
    printf("\nAverage Turnaround Time:\t%f", average_turnaround_time);
    return 0;
}
```

OUTPUT:

```
pl-17@pl17-OptiPlex-3020:~/07IT$ gcc 3_RR.c
pl-17@pl17-OptiPlex-3020:~/07IT$ ./a.out
Enter Total Number of Processes (max 10):
        4
Enter Details of Process[1]
Arrival Time:  0
Burst Time:    5
Enter Details of Process[2]
Arrival Time:  1
Burst Time:    4
Enter Details of Process[3]
Arrival Time:  2
Burst Time:    2
Enter Details of Process[4]
Arrival Time:  4
Burst Time:    1
Enter Time Quantum:
        2

Process ID  Arrival Time Burst Time Completion Time Turnaround Time Waiting Time
Process[3]     2             2           6              4              2
Process[4]     4             1           7              3              2
Process[2]     1             4           11             10             6
Process[1]     0             5           12             12             7
Average Waiting Time:        4.250000
Average Turnaround Time:     7.250000
```