VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Name: Apurba Koirala

Reg no: 22BCE3799

Subject Code: BCSE303P

Course Title: Operating Systems Lab

Lab Slot: L39 + L40

Guided by: Dr. Anto S

Lab Assessment 2

Reg No: 22BCE3799
Name: Apurba Koirala
Slot: L39+L40

Exercise 2 – CPU Scheduling Algorithms

Write a C program to simulate the following non-preemptive CPU scheduling algorithms to find turnaround time and waiting time.

a) FCFS

b) SJF

c) Round Robin (pre-emptive)

d) Priority

e) Shortest Remaining Time First (Pre-emptive)

Get the input from the user and print the output.

AIM:

To develop a C program that simulates various CPU scheduling algorithms to compute the turnaround time and waiting time for a set of processes. The scheduling algorithms to be implemented are:

a) First-Come, First-Served (FCFS)

b) Shortest Job First (SJF)

c) Round Robin (Pre-emptive)

d) Priority Scheduling

e) Shortest Remaining Time First (SRTF) (Pre-emptive)

Reg No: 22BCE3799
Name: Apurba Koirala
Slot: L39+L40

ALGORITHM:

a) FCFS (First-Come, First-Served):

1. Arrange the processes in the sequence of their arrival times.
2. Compute the waiting time for each process by summing the burst times of the processes that came before it.
3. Calculate the turnaround time as the sum of the burst time and the waiting time for each process.

b) SJF (Shortest Job First):

1. Arrange the processes in ascending order based on their burst times.
2. Determine the waiting time for each process similarly to the FCFS method.
3. Calculate the turnaround time for each process.

c) Round Robin (Pre-emptive):

1. Define the time quantum.
2. Initialize the remaining burst time for each process.
3. Iterate through the processes, decreasing their remaining burst times by the time quantum or by the remaining time if it's less than the quantum.
4. If a process completes, compute its waiting time.
5. Continue until all processes have completed.
6. Compute the turnaround time for each process.

d) Priority Scheduling:

1. Arrange the processes according to their priorities, with lower numbers indicating higher priority.
2. Compute the waiting time for each process as done in FCFS.
3. Determine the turnaround time for each process.

e) SRTF (Shortest Remaining Time First) (Pre-emptive):

1. At each time unit, choose the process with the shortest remaining burst time that has already arrived.
2. Decrease the burst time of the chosen process.
3. When a process completes, calculate its waiting time.
4. Repeat until all processes are completed.
5. Compute the turnaround time for each process.

Reg No: 22BCE3799
Name: Apurba Koirala
Slot: L39+L40

SOURCE CODE :

```
#include <stdio.h>


struct Task {

    int id;

    int burst_time;

    int wait_time;

    int turnaround_time;

    int priority_level;

    int arrival_time;

};


void sortByBurstTime(struct Task tasks[], int count) {

    for (int i = 0; i < count-1; i++) {

        for (int j = 0; j < count-i-1; j++) {

            if (tasks[j].burst_time > tasks[j+1].burst_time) {

                struct Task temp = tasks[j];

                tasks[j] = tasks[j+1];

                tasks[j+1] = temp;

            }

        }

    }

}


void sortByPriority(struct Task tasks[], int count) {
```

```
for (int i = 0; i < count-1; i++) {
    for (int j = 0; j < count-i-1; j++) {
      if (tasks[j].priority_level > tasks[j+1].priority_level) {
        struct Task temp = tasks[j];
        tasks[j] = tasks[j+1];
        tasks[j+1] = temp;
      }
    }
  }
}

void calculateWaitTime(struct Task tasks[], int count) {
  tasks[0].wait_time = 0;
  for (int i = 1; i < count; i++) {
    tasks[i].wait_time = tasks[i-1].wait_time + tasks[i-1].burst_time;
  }
}

void calculateTurnaroundTime(struct Task tasks[], int count) {
  for (int i = 0; i < count; i++) {
    tasks[i].turnaround_time = tasks[i].burst_time + tasks[i].wait_time;
  }
}

void firstComeFirstServe(struct Task tasks[], int count) {
  calculateWaitTime(tasks, count);

    calculateTurnaroundTime(tasks, count);

}


void shortestJobFirst(struct Task tasks[], int count) {

    sortByBurstTime(tasks, count);

    calculateWaitTime(tasks, count);

    calculateTurnaroundTime(tasks, count);

}


void priorityScheduling(struct Task tasks[], int count) {

    sortByPriority(tasks, count);

    calculateWaitTime(tasks, count);

    calculateTurnaroundTime(tasks, count);

}


void roundRobin(struct Task tasks[], int count, int time_quantum) {

    int remaining_burst_time[count];

    for (int i = 0; i < count; i++) {

        remaining_burst_time[i] = tasks[i].burst_time;

    }


    int current_time = 0;

    while (1) {

        int all_done = 1;
  for (int i = 0; i < count; i++) {
            if (remaining_burst_time[i] > 0) {
                all_done = 0;
                if (remaining_burst_time[i] > time_quantum) {
                    current_time += time_quantum;
                    remaining_burst_time[i] -= time_quantum;
                } else {
                    current_time += remaining_burst_time[i];
                    tasks[i].wait_time = current_time - tasks[i].burst_time;
                    remaining_burst_time[i] = 0;
                }
            }
        }
        if (all_done == 1)
            break;
    }
    calculateTurnaroundTime(tasks, count);
}


void shortestRemainingTimeFirst(struct Task tasks[], int count) {
    int remaining_burst_time[count];
    for (int i = 0; i < count; i++) {
        remaining_burst_time[i] = tasks[i].burst_time;
    }
```

Reg No: 22BCE3799
Name: Apurba Koirala
Slot: L39+L40

```
int completed_tasks = 0, current_time = 0, minimum_time = 1e9;

  int shortest_task = 0, finish_time;

  int task_found = 0;


  while (completed_tasks != count) {

      for (int j = 0; j < count; j++) {

          if ((tasks[j].arrival_time <= current_time) && (remaining_burst_time[j] <
minimum_time) && remaining_burst_time[j] > 0) {

              minimum_time = remaining_burst_time[j];

              shortest_task = j;

              task_found = 1;

          }

      }

      if (task_found == 0) {

          current_time++;

          continue;

      }

      remaining_burst_time[shortest_task]--;

      minimum_time = remaining_burst_time[shortest_task];

      if (minimum_time == 0) minimum_time = 1e9;


      if (remaining_burst_time[shortest_task] == 0) {

          completed_tasks++;

          finish_time = current_time + 1;

          tasks[shortest_task].wait_time = finish_time - tasks[shortest_task].burst_time -
tasks[shortest_task].arrival_time;
```

```
    if (tasks[shortest_task].wait_time < 0) tasks[shortest_task].wait_time = 0;
        }
        current_time++;
    }
    calculateTurnaroundTime(tasks, count);
}


void displayResults(struct Task tasks[], int count) {
    printf("\nTasks\tBurst Time\tWaiting Time\tTurnaround Time\n");
    for (int i = 0; i < count; i++) {
        printf("%d\t%d\t\t%d\t\t%d\n", tasks[i].id, tasks[i].burst_time, tasks[i].wait_time, tasks[i].turnaround_time);
    }
}


int main() {
    int count, time_quantum, choice;
    printf("Enter the number of tasks: ");
    scanf("%d", &count);


    struct Task tasks[count];
    for (int i = 0; i < count; i++) {
        tasks[i].id = i+1;
        printf("Enter burst time for task %d: ", i+1);
        scanf("%d", &tasks[i].burst_time);
    }
```

Reg No: 22BCE3799
Name: Apurba Koirala
Slot: L39+L40

```
printf("Choose Scheduling Algorithm:\n");
  printf("1. FCFS\n2. SJF\n3. Priority Scheduling\n4. Round Robin\n5. SRTF\n");
  scanf("%d", &choice);


  switch(choice) {
     case 1:
        firstComeFirstServe(tasks, count);
        break;
     case 2:
        shortestJobFirst(tasks, count);
        break;
     case 3:
        for (int i = 0; i < count; i++) {
           printf("Enter priority level for task %d: ", i+1);
           scanf("%d", &tasks[i].priority_level);
        }
        priorityScheduling(tasks, count);
        break;
     case 4:
        printf("Enter time quantum: ");
        scanf("%d", &time_quantum);
        roundRobin(tasks, count, time_quantum);
        break;
     case 5:
```

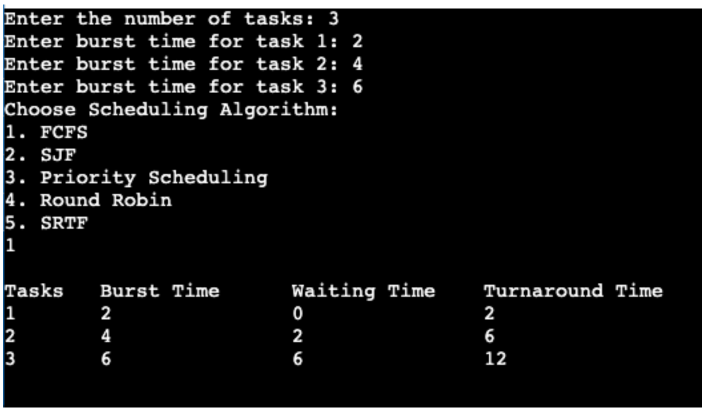
Reg No : 22BCE3799
Name: Apurba Koirala
Slot: L39+L40

```
 for (int i = 0; i < count; i++) {

                printf("Enter arrival time for task %d: ", i+1);

                scanf("%d", &tasks[i].arrival_time);

            }

            shortestRemainingTimeFirst(tasks, count);

            break;

        default:

            printf("Invalid choice!\n");

            return 0;

    }

    displayResults(tasks, count);

    return 0;

}
```

Reg No: 22BCE3799
Name: Apurba Koirala
Slot: L39+L40

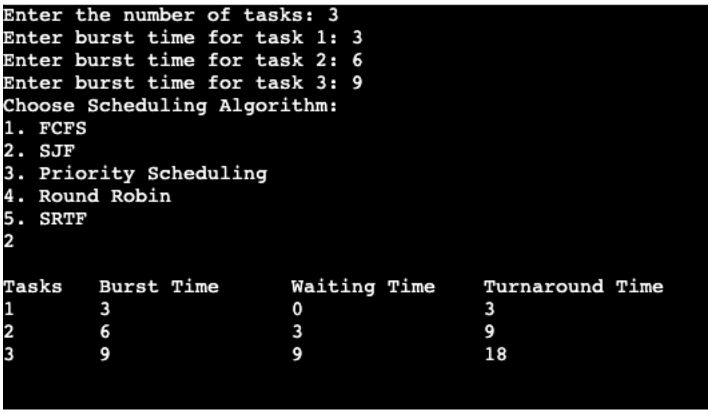
OUTPUT SCREEN SHOT:

FCFS:



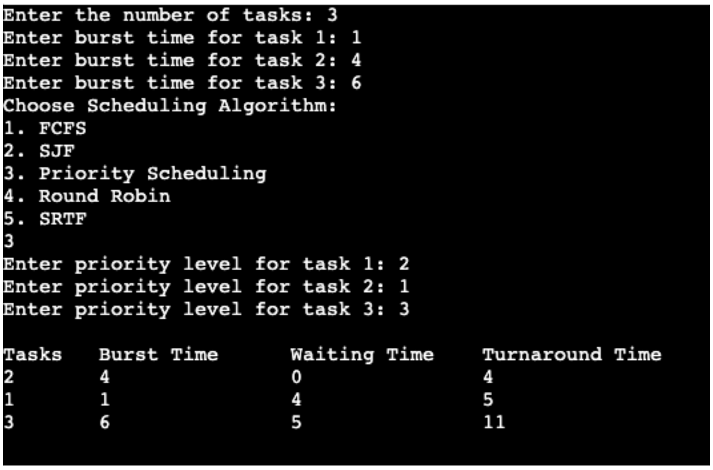
```
Enter the number of tasks: 3
Enter burst time for task 1: 2
Enter burst time for task 2: 4
Enter burst time for task 3: 6
Choose Scheduling Algorithm:
1. FCFS
2. SJF
3. Priority Scheduling
4. Round Robin
5. SRTF
1

Tasks   Burst Time      Waiting Time    Turnaround Time
1       2               0               2
2       4               2               6
3       6               6               12
```

SJF:



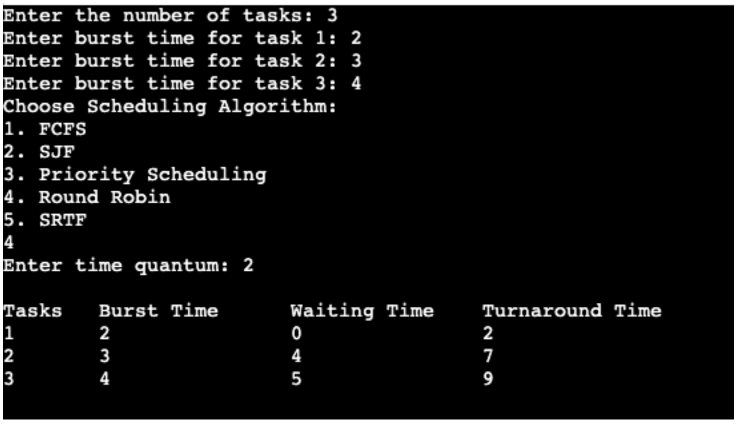
```
Enter the number of tasks: 3
Enter burst time for task 1: 3
Enter burst time for task 2: 6
Enter burst time for task 3: 9
Choose Scheduling Algorithm:
1. FCFS
2. SJF
3. Priority Scheduling
4. Round Robin
5. SRTF
2

Tasks   Burst Time      Waiting Time    Turnaround Time
1       3               0               3
2       6               3               9
3       9               9               18
```

Priority Scheduling:



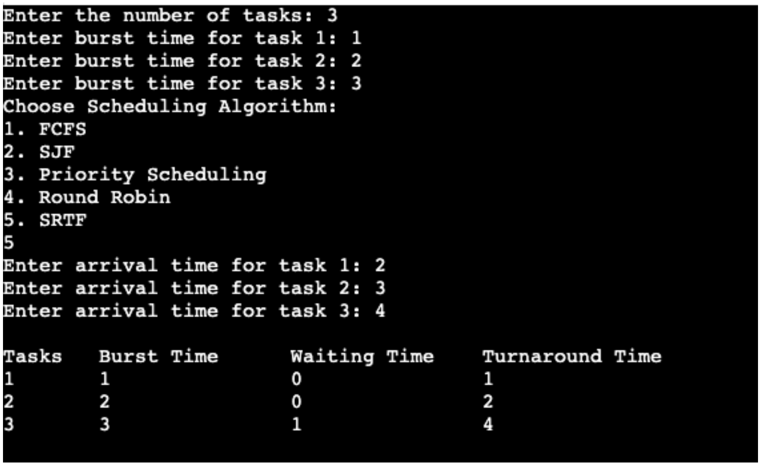
```
Enter the number of tasks: 3
Enter burst time for task 1: 1
Enter burst time for task 2: 4
Enter burst time for task 3: 6
Choose Scheduling Algorithm:
1. FCFS
2. SJF
3. Priority Scheduling
4. Round Robin
5. SRTF
3
Enter priority level for task 1: 2
Enter priority level for task 2: 1
Enter priority level for task 3: 3

Tasks   Burst Time      Waiting Time    Turnaround Time
2       4               0               4
1       1               4               5
3       6               5               11
```

Round Robin:



```
Enter the number of tasks: 3
Enter burst time for task 1: 2
Enter burst time for task 2: 3
Enter burst time for task 3: 4
Choose Scheduling Algorithm:
1. FCFS
2. SJF
3. Priority Scheduling
4. Round Robin
5. SRTF
4
Enter time quantum: 2

Tasks   Burst Time      Waiting Time    Turnaround Time
1       2               0               2
2       3               4               7
3       4               5               9
```

SRTF:



```
Enter the number of tasks: 3
Enter burst time for task 1: 1
Enter burst time for task 2: 2
Enter burst time for task 3: 3
Choose Scheduling Algorithm:
1. FCFS
2. SJF
3. Priority Scheduling
4. Round Robin
5. SRTF
5
Enter arrival time for task 1: 2
Enter arrival time for task 2: 3
Enter arrival time for task 3: 4

Tasks   Burst Time      Waiting Time    Turnaround Time
1       1               0               1
2       2               0               2
3       3               1               4
```

RESULTS:

Burst Time, Waiting Time and Turnaround Time successfully computed for each of the scheduling algorithms.