Ex. No.: 6a) **Roll No. 230701091**

FIRST COME FIRST SERVE

Aim:

To implement First-come First- serve (FCFS) scheduling

technique. PROGRAM :

```
#include <stdio.h>
   int
   main() {
   int n, i;
      int bt[20], wt[20], tat[20];
      float avg_wt = 0, avg_tat = 0;
   printf("Enter the number of
   process:\n"); scanf("%d", &n);
      printf("Enter the burst time of the
   processes:\n"); for (i = 0; i < n; i++) {
            scanf("%d", &bt[i]);
      }
      wt[0] = 0;
      tat[0] = bt[0];
      for (i = 1; i < n; i++) {
            wt[i] = wt[i - 1] + bt[i - 1];
            tat[i] = wt[i] + bt[i];
      }
      printf("Process Burst Time Waiting Time Turn Around
   Time\n"); for (i = 0; i < n; i++) {
            printf("%d\t%d\t\t%d\t\t%d\n", i, bt[i],
      wt[i], tat[i]); avg_wt += wt[i];
            avg_tat += tat[i];
      }
      avg_wt /= n;
   avg_tat /= n;
       printf("Average waiting time is: %.1f\n", avg_wt);
    printf("Average Turn around Time is: %.1f\n", avg_tat);

      return 0;
}
```

OUTPUT :

```
Enter the number of process:
3
Enter the burst time of the processes:
24 3 3
Process Burst Time Waiting Time Turn Around Time
0       24              0               24
1       3               24              27
2       3               27              30
Average waiting time is: 17.0
Average Turn around Time is: 27.0
```

Ex. No.: 6b) **Roll No. 230701091**

SHORTEST JOB FIRST

Aim:

To implement the Shortest Job First (SJF) scheduling technique.

Program Code:

```
#include
<stdio.h> struct
Process {
    int pid; // Process ID
    int burst_time; // Burst Time
    int waiting_time; // Waiting Time
    int turn_around_time; // Turnaround Time
};

void calculate_times(struct Process proc[], int n) {
    int total_waiting_time = 0, total_turn_around_time = 0;

    // Calculating Waiting Time and
  Turnaround Time proc[0].waiting_time = 0;
  proc[0].turn_around_time =
  proc[0].burst_time;

    for (int i = 1; i < n; i++) {
        proc[i].waiting_time = proc[i - 1].waiting_time + proc[i -
    1].burst_time; proc[i].turn_around_time = proc[i].waiting_time +
    proc[i].burst_time;
    }

    // Calculate total waiting time and total
  turnaround time for (int i = 0; i < n; i++) {
    total_waiting_time += proc[i].waiting_time;
    total_turn_around_time += proc[i].turn_around_time;
    }

    // Calculate average waiting time and average
  turnaround time float avg_waiting_time =
  (float)total_waiting_time / n;
    float avg_turn_around_time = (float)total_turn_around_time / n;
    // Displaying the results
    printf("Process\tBurst Time\tWaiting Time\tTurn Around
  Time\n"); for (int i = 0; i < n; i++) {
    printf("%d\t%d\t\t%d\t\t%d\n", proc[i].pid, proc[i].burst_time,
proc[i].waiting_time, proc[i].turn_around_time);
    }
    printf("Average waiting time is: %.2f\n", avg_waiting_time);
```

```
    printf("Average Turn Around Time is: %.2f\n",
    avg_turn_around_time);
}
```

```
int
main() {
int n;

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

    struct Process proc[n];

    // Taking burst time as input for each
process printf("Enter the burst time of the
processes:\n"); for (int i = 0; i < n; i++) {
    proc[i].pid = i + 1; //
    Process ID printf("Process %d:
    ", proc[i].pid); scanf("%d",
    &proc[i].burst_time);
    }
    // Sorting the processes based on burst time in

ascending order for (int i = 0; i < n - 1; i++) {
        for (int j = i + 1; j < n; j++) {
            if (proc[i].burst_time > proc[j].burst_time) {
                // Swap the processes
                struct Process temp
        = proc[i]; proc[i] = proc[j];
                proc[j] = temp;
            }
        }
    }

    // Calculate and display the
results calculate_times(proc, n);

    return 0;

}
```

Output :

```
Process 4: 5
Process Burst Time      Waiting Time    Turn Around Time
2       4               0               4
4       5               4               9
1       8               9               17
3       9               17              26
Average waiting time is: 7.50
Average Turn Around Time is: 14.00
```

Ex. No.: 6c) **Roll No. 230701091**

PRIORITY SCHEDULING

Aim:

To implement priority scheduling technique.

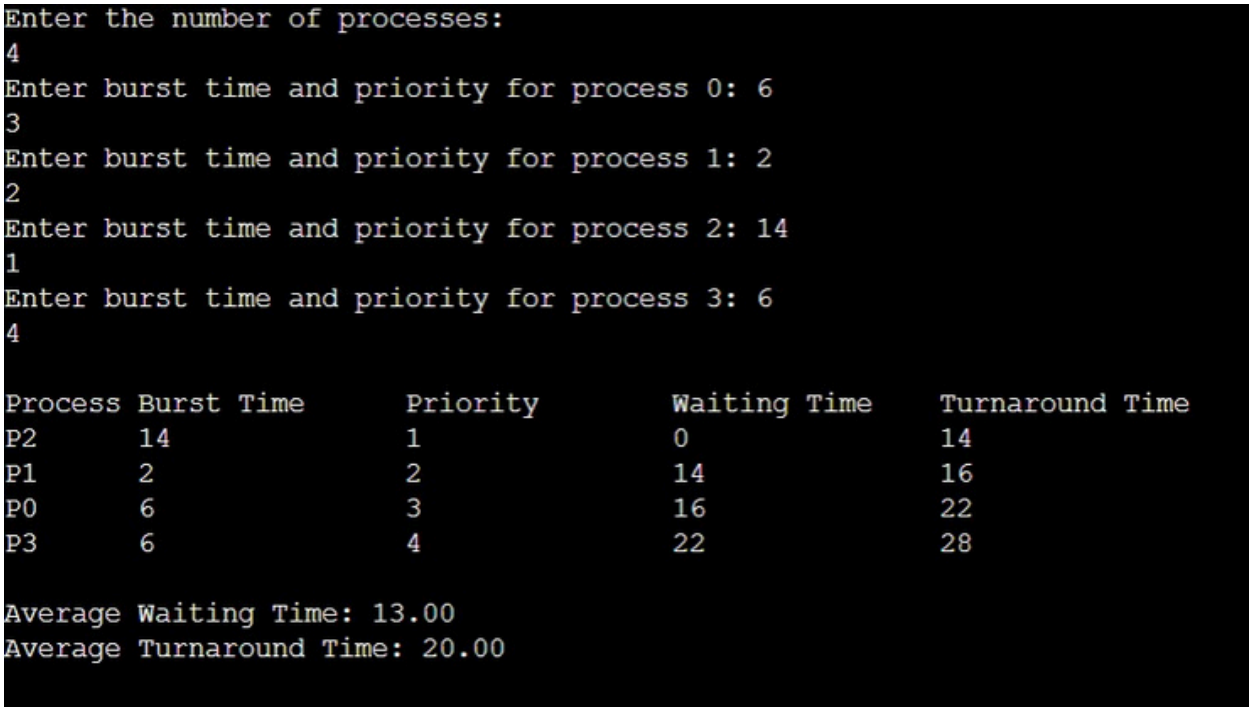
PROGRAM:

```
#include

<stdio.h> struct
Process {
      int
   id; int
   bt;
   int
   priority;
   int wt;
    int tat;
int
main() {
      int n, i, j;
      struct Process p[20];
      float total_wt = 0, total_tat = 0;
   printf("Enter the number of
   processes:\n"); scanf("%d", &n);

         for (i = 0; i < n;
      i++) { p[i].id = i;
            printf("Enter burst time and priority for process
      %d: ", i); scanf("%d %d", &p[i].bt, &p[i].priority);
      }
      struct Process
   temp; for (i = 0; i <
   n - 1; i++) {
            for (j = i + 1; j < n; j++) {
                  if (p[i].priority >
            p[j].priority) { temp =
            p[i];
                        p
            [i] = p[j];
            p[j] =
            temp;
                  }}}
      p[0].wt = 0;
   p[0].tat =
   p[0].bt;
      for (i = 1; i < n; i++) {
            p[i].wt = p[i - 1].wt + p[i - 1].bt;
```

```
        p[i].tat = p[i].wt + p[i].bt;
    }
    printf("\nProcess\tBurst Time\tPriority\tWaiting Time\tTurnaround
Time\n"); for (i = 0; i < n; i++) {
        printf("P%d\t%d\t\t%d\t\t%d\t\t%d\n", p[i].id, p[i].bt, p[i].priority,
    p[i].wt, p[i].tat); total_wt += p[i].wt;
```

```
            total_tat += p[i].tat;
        }
        printf("\nAverage Waiting Time: %.2f", total_wt / n);
    printf("\nAverage Turnaround Time: %.2f\n", total_tat / n);

        return 0;
}
```

OUTPUT :



```
Enter the number of processes:
4
Enter burst time and priority for process 0: 6
3
Enter burst time and priority for process 1: 2
2
Enter burst time and priority for process 2: 14
1
Enter burst time and priority for process 3: 6
4

Process Burst Time      Priority        Waiting Time    Turnaround Time
P2      14              1               0               14
P1      2               2               14              16
P0      6               3               16              22
P3      6               4               22              28

Average Waiting Time: 13.00
Average Turnaround Time: 20.00
```

Ex. No.: 6d) **Roll No. 230701091**

ROUND ROBIN SCHEDULING

Aim:

To implement the Round Robin (RR) scheduling technique. PROGRAM:

```
#include
<stdio.h> struct
Process {
      int id;
      int
   arrivalTime;
   int burstTime;
      int
   remainingTime;
   int waitingTime;
      int turnaroundTime;
};
int main() {

      int n, timeQuantum, time = 0, done;
   printf("Enter Total Number of Processes:
   "); scanf("%d", &n);
      struct Process p[n];
         for (int i = 0; i <
      n; i++) { p[i].id = i
      + 1;
            printf("\nEnter Details of Process[%d]\n", i
      + 1); printf("Arrival Time: ");
            scanf("%d",
      &p[i].arrivalTime);
      printf("Burst Time: ");
      scanf("%d",
      &p[i].burstTime);
            p[i].remainingTime =
      p[i].burstTime; p[i].waitingTime = 0;
            p[i].turnaroundTime = 0;
      }

      printf("\nEnter Time Quantum:
   "); scanf("%d", &timeQuantum);

      int completed =
   0; while (completed
   != n) {
            done = 1;
```

```
for (int i = 0; i < n; i++) {
        if (p[i].arrivalTime <= time &&
p[i].remainingTime > 0) { done = 0;
            if (p[i].remainingTime
  > timeQuantum) { time +=
  timeQuantum;
```

```
                    p[i].remainingTime -= timeQuantum;
                } else {
                    time += p[i].remainingTime;
                    p[i].waitingTime = time - p[i].burstTime -
            p[i].arrivalTime; p[i].turnaroundTime = time -
            p[i].arrivalTime; p[i].remainingTime = 0;
                    completed++;
                }}}
    if (done) {time++; } }
    float totalWT = 0, totalTAT = 0;
    printf("\nProcess ID\tBurst Time\tTurnaround Time\tWaiting
  Time\n"); for (int i = 0; i < n; i++) {
    printf("Process[%d]\t%d\t\t%d\t\t%d\n", p[i].id, p[i].burstTime,
p[i].turnaroundTime, p[i].waitingTime);
        totalWT +=
    p[i].waitingTime; totalTAT +=
    p[i].turnaroundTime;
    }
    printf("\nAverage Waiting Time: %.6f", totalWT / n);
  printf("\nAvg Turnaround Time: %.6f\n", totalTAT / n);

    return 0;}
```

OUTPUT:

```
Enter Details of Process[4]
Arrival Time: 3
Burst Time: 6

Enter Time Quantum: 3

Process ID      Burst Time      Turnaround Time Waiting Time
Process[1]      4               13              9
Process[2]      7               21              14
Process[3]      5               16              11
Process[4]      6               18              12

Average Waiting Time: 11.500000
Avg Turnaround Time: 17.000000
```