Akilesh prasad Roll
NO : 230701020

FIRST COME FIRST SERVE

Aim:

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

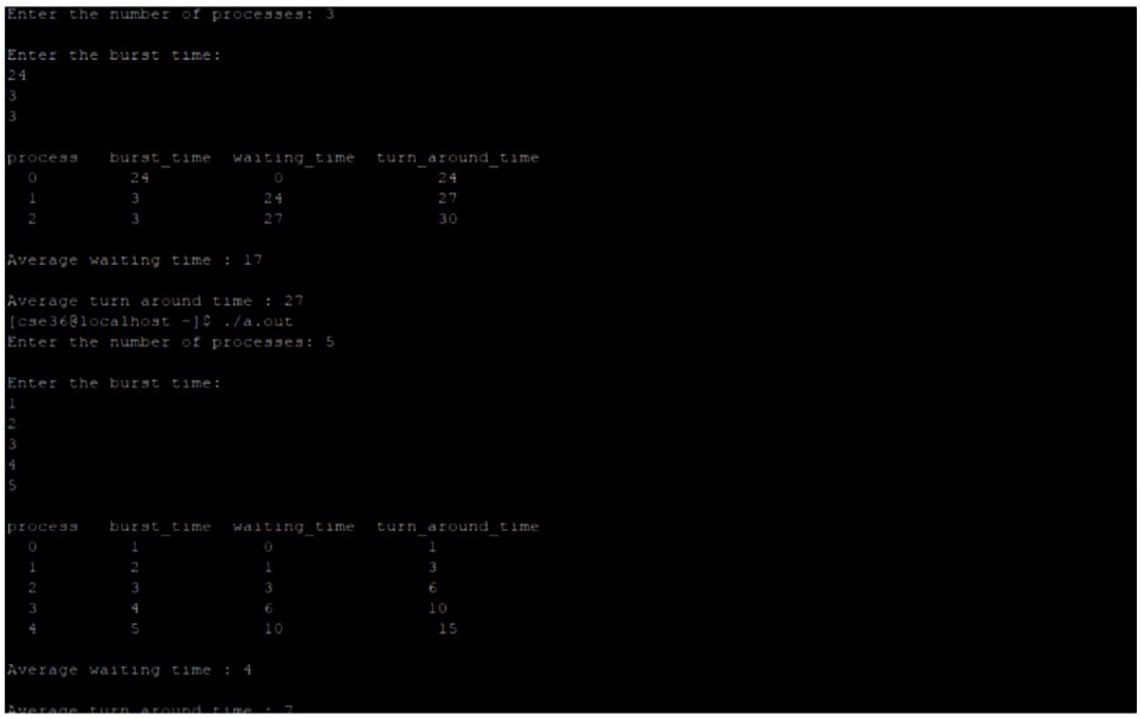
Algorithm:

1. Get the number of processes from the user.
2. Read the process name and burst Ɵme.
3. Calculate the total process Ɵme.
4. Calculate the total waiƟng Ɵme and total turnaround Ɵme for each process 5.

Display the process name & burst Ɵme for each process. 6. Display the total waiƟng Ɵme, average waiƟng Ɵme, turnaround Ɵme

Program Code:

OUTPUT :



Ex. no: 6a) Name :

Akilesh prasad Roll

NO : 230701020

SHORTEST JOB FIRST

Aim:

To implement the Shortest Job First (SJF) scheduling technique

Algorithm:

1. Declare the structure and its elements.

2. Get number of processes as input from the user.

3. Read the process name, arrival Ɵme and burst Ɵme

4. IniƟalize waiƟng Ɵme, turnaround Ɵme & flag of read processes to zero. 5. Sort based on burst Ɵme of all processes in ascending order 6. Calculate the waiƟng Ɵme and turnaround Ɵme for each process. 7. Calculate the average

waiƟng Ɵme and average turnaround Ɵme. 8. Display the results.

Program Code:

```
#include<stdio.h>
int main(){
  int n;
  printf("Enter the number of processes: ");
  scanf("%d",&n);
  int process[n], burst_time[n],arrival_time[n], waiting_time[n], turn_around_time[n];
  float  total_waiting_time=0,total_turn_around_time=0;
  printf("\nEnter the burst time: \n");
  for(int i=0;i<n;i++){
    process[i]=i;
    scanf("%d",&burst_time[i]);
  }
//sorting burst time
  for(int i=0;i<n;i++)
    for(int j=0;j<n-1;j++)
      if(burst_time[j]>burst_time[j+1])//swapping
      {
         burst_time[j]=burst_time[j]-burst_time[j+1];
         process[j]=process[j]-process[j+1];

         burst_time[j+1]=burst_time[j+1]+burst_time[j];
         process[j+1]=process[j+1]+process[j];

         burst_time[j]=burst_time[j+1]-burst_time[j];
         process[j]=process[j+1]-process[j];
      }



//finding waiting time
  waiting_time[0]=0;
  for(int i=1;i<n;i++)
   waiting_time[i]=waiting_time[i-1]+burst_time[i-1];

//finding turnaround time
  for(int i=0;i<n;i++){
   turn_around_time[i]=burst_time[i]+waiting_time[i];
   total_turn_around_time+=turn_around_time[i];
   total_waiting_time+=waiting_time[i];
  }

  printf("\nprocess   burst_time  waiting_time  turn_around_time\n");
  for(int i=0;i<n;i++){
  printf("  %d        %d          %d             %d\n", process[i],burst_time[i], waiting_time[i],turn_around_time[i]);
  }
//printf("%d %d",total_waiting_time/n,(total_turn_around_time/n));
  printf("\nAverage waiting time : %.2f\n",(total_waiting_time/n));
  printf("\nAverage turn around time : %.2f\n",(total_turn_around_time/n));
}
```

OUTPUT :

```
Enter the number of processes: 4

Enter the burst time:
8
4
9
5

process   burst_time  waiting_time  turn_around_time
  1          4             0              4
  3          5             4              9
  0          8             9              17
  2          9             17              26

Average waiting time : 7.50

Average turn around time : 14.00
```

Ex. no: 6a) Name :

Akilesh prasad Roll

NO : 230701020

PRIORITY SCHEDULING

Aim:

To implement priority scheduling technique

Algorithm:

1. Get the number of processes from the user.
2. Read the process name, burst Ɵme and priority of process.
3. Sort based on burst Ɵme of all processes in ascending order based priority 4.
Calculate the total waiƟng Ɵme and total turnaround Ɵme for each process 5.
Display the process name & burst Ɵme for each process.
6. Display the total waiƟng Ɵme, average waiƟng Ɵme, turnaround Ɵme

Program Code:

```c
#include<stdio.h>
int main(){
  int n;
  printf("Enter the number of processes: ");
  scanf("%d",&n);
  int process[n], burst_time[n],priority[n], waiting_time[n], turn_around_time[n];
  float  total_waiting_time=0,total_turn_around_time=0;
  printf("\nEnter the burst time with priority: \n");
  for(int i=0;i<n;i++){
    process[i]=i;
    printf("\nEnter burst_time[%d] with priority[%d]: \n",i+1,i+1);
    scanf("%d %d",&burst_time[i],&priority[i]);
  }
//sorting burst time
  for(int i=0;i<n;i++)
    for(int j=0;j<n-1;j++)
      if(priority[j]>priority[j+1])//swapping
      {
         burst_time[j]=burst_time[j]-burst_time[j+1];
         process[j]=process[j]-process[j+1];
         priority[j]=priority[j]-priority[j+1];

         burst_time[j+1]=burst_time[j+1]+burst_time[j];
         process[j+1]=process[j+1]+process[j];
         priority[j+1]=priority[j+1]+priority[j];

         burst_time[j]=burst_time[j+1]-burst_time[j];
         process[j]=process[j+1]-process[j];
         priority[j]=priority[j+1]-priority[j];
      }



//finding waiting time
  waiting_time[0]=0;
  for(int i=1;i<n;i++)
   waiting_time[i]=waiting_time[i-1]+burst_time[i-1];
```

```c
//finding turnaround time
  for(int i=0;i<n;i++){
   turn_around_time[i]=burst_time[i]+waiting_time[i];
   total_turn_around_time+=turn_around_time[i];
   total_waiting_time+=waiting_time[i];
  }

 printf("\nprocess   burst_time  waiting_time  turn_around_time\n");
 for(int i=0;i<n;i++){
  printf("  %d         %d           %d              %d\n", process[i],burst_time[i], waiting_time[i],turn_around_time[i]);
  }
//printf("%d %d",total_waiting_time/n,(total_turn_around_time/n));
  printf("\nAverage waiting time : %.2f\n",(total_waiting_time/n));
  printf("\nAverage turn around time : %.2f\n",(total_turn_around_time/n));
}
```

OUTPUT :

```
Enter the number of processes: 4

Enter the burst time with priority:

Enter burst_time[1] with priority[1]:
6
3

Enter burst_time[2] with priority[2]:
2
2

Enter burst_time[3] with priority[3]:
14
1

Enter burst_time[4] with priority[4]:
6
4

process   burst_time  waiting_time  turn_around_time
  2         14            0              14
  1         2            14              16
  0         6            16              22
  3         6            22              28

Average waiting time : 13.00

Average turn around time : 20.00
```

Ex. no: 6a) Name :

Akilesh prasad Roll

NO : 230701020

ROUND ROBIN SCHEDULING

Aim:

To implement the Round Robin (RR) scheduling technique

Algorithm:

1. Declare the structure and its elements. 2. Get number of processes and Time quantum as input from the user. 3. Read the process name, arrival Ɵme and burst Ɵme 4. Create an array rem_bt[] to keep track of remaining burst Ɵme of processes which is iniƟally copy of bt[] (burst Ɵmes array) 5. Create another array wt[] to store waiƟng Ɵmes of processes. IniƟalize this array as 0. 6. IniƟalize Ɵme : t = 0 7. Keep traversing the all processes while all processes are not done. Do following for i'th process if it is not done yet. a- If rem_bt[i] > quantum (i) t = t + quantum (ii) bt_rem[i] -= quantum; b- Else // Last cycle for this process (i) t = t + bt_rem[i]; (ii) wt[i] = t - bt[i] (iii) bt_rem[i] = 0; // This process is over 8. Calculate the waiƟng Ɵme and turnaround Ɵme for each process. 9. Calculate the average waiƟng Ɵme and average turnaround Ɵme. 10. Display the results.

Program Code:

```c
#include <stdio.h>
#include <stdbool.h>

struct Process {
    int id;
    int arrival_time;
    int burst_time;
    int remaining_time;
    int waiting_time;
    int turnaround_time;
};

void calculate_times(struct Process processes[], int n, int quantum) {
    int time = 0;
    bool done;

    do {
        done = true;

        for (int i = 0; i < n; i++) {
            if (processes[i].remaining_time > 0) {
                done = false;

                if (processes[i].remaining_time > quantum) {
                    time += quantum;
                    processes[i].remaining_time -= quantum;
                } else {
                    time += processes[i].remaining_time;
                    processes[i].waiting_time = time - processes[i].burst_time - processes[i].arrival_time;
                    processes[i].remaining_time = 0;
                }
            }
        }
    } while (!done);

    for (int i = 0; i < n; i++) {
        processes[i].turnaround_time = processes[i].burst_time + processes[i].waiting_time;
    }
}
```

```c
void print_results(struct Process processes[], int n) {
    float total_waiting_time = 0;
    float total_turnaround_time = 0;

    printf("\nProcess ID   Burst Time  Waiting Time   Turnaround Time\n");
    for (int i = 0; i < n; i++) {
        total_waiting_time += processes[i].waiting_time;
        total_turnaround_time += processes[i].turnaround_time;

        printf("Process[%d]    %d           %d             %d\n",
               processes[i].id,
                 processes[i].burst_time,
               processes[i].turnaround_time,
               processes[i].waiting_time);
    }


    printf("\nAverage Waiting Time: %.6f\n", total_waiting_time / n);

    printf("Average Turnaround Time: %.6f\n", total_turnaround_time / n);
}

int main() {
    int n, quantum;

    printf("Enter Total Number of Processes: ");
    scanf("%d", &n);

    struct Process processes[n];

    for (int i = 0; i < n; i++) {
        processes[i].id = i + 1;
        printf("\nEnter Details of Process[%d]\n", processes[i].id);
        printf("Arrival Time: ");
        scanf("%d", &processes[i].arrival_time);
        printf("Burst Time: ");
        scanf("%d", &processes[i].burst_time);
        processes[i].remaining_time = processes[i].burst_time;
        processes[i].waiting_time = 0;
        processes[i].turnaround_time = 0;
    }
```

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

    calculate_times(processes, n, quantum);
    print_results(processes, n);

    return 0;
}
```

OUTPUT :

```
Enter Total Number of Processes: 4

Enter Details of Process[1]
Arrival Time: 0
Burst Time: 4

Enter Details of Process[2]
Arrival Time: 1
Burst Time: 7

Enter Details of Process[3]
Arrival Time: 2
Burst Time: 5

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

Enter Time Quantum: 3

Process ID   Burst Time  Waiting Time   Turnaround 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
Average Turnaround Time: 17.000000
```