LOVELY PROFESSIONAL UNIVERSITY

Academic Task-3 (Operating System)

Course Code:-CSE-316

NAME:- Maan Pratap Singh

ROLL No. A-06

Reg. No. 11807726

SECTION:- K18JE

Submitted To:- Ms.Milanjit Kaur

Name : Maan Pratap Singh

Roll No : **A**-06

Reg .No : 11807726

Email : maanpratapsingh29@gmail.com

Github Link : https://github.com/Maan2911/CA3-CSE316-K18JE-11807726

Question No. Design a scheduling program that is capable of scheduling many processes that comes in at some time interval and are allocated the CPU not more that 10 time units. CPU must schedule processes having short execution time first. CPU is idle for 3 time units and does not entertain any process prior this time. Scheduler must maintain a queue that keeps the order of execution of all the processes. Compute average waiting and turnaround time.

Description: The Given problem of scheduling program designing an operating system, a programmer must consider which scheduling algorithm will perform best for the use the system is going to see. There is no universal "best" scheduling algorithm, and many operating systems use extended or combinations of the scheduling algorithms above. First come first serve scheduling algorithm is simple to understand and suitable only for batch system where waiting time is large. The shortest job first scheduling algorithm deals with different approach. In this algorithm, the major benefit is that it gives the minimum average waiting time. The priority scheduling algorithm is based on the priority in which the highest priority job can run first and the lowest priority job need to wait though it will create a problem of starvation. The round robin scheduling algorithm is preemptive which is based on FCFS policy and time quantum. This algorithm is suitable for the time sharing systems. In multilevel queue scheduling, processes are permanently assigned to a queue depending upon its nature and no process in the lower priority queue could run unless the higher priority queues were empty. Also, it is pre-emptive in nature. Multilevel feedback queue scheduling is also pre-emptive in nature and it allows the processes to move between the queues depending upon the given time quantum.

Explanation:

1. For solving this problem the processes are scheduled considering their arrival and Wait time.
2. I have made 5 functions: readData(); Init(); getNextProcess(); dispTime(); computeSRT();
3. User will get to enter process burst time and arrival time.
4. Depending on that ComputeSRT function will calculate the shortest run time of the processes and it will be entered in gantt chart.

5. After every unit time when a process will entered by user ComputeSRT will compare the remaining run time of currently running process and if less than new process then it will be preumpted.

6. After completion of all the processes the program will prepare a gantt chart and program will end

Code Snippet:

```
#include<stdio.h>

int main()

{

  int count,j,n;

  int time,remaining;

  int flag=0,time_quantum=10;

  int waiting_time=0,turn_around_time=0,arrival_time[10],burst_time[10],rt[10];

  printf("\n\nEnter the Total number of Process:\t ");

  scanf("%d",&n);

  remaining=n;

  for(count=0;count<n;count++)

  {

    printf("Enter Arrival Time and Burst Time for Process Process Number %d :",count+1);

    scanf("%d",&arrival_time[count]);

    scanf("%d",&burst_time[count]);

    rt[count]=burst_time[count];

  }

  printf("Enter Time Quantum:%d\t",time_quantum);


  printf("\n\nProcess\t|Turnaround Time|Waiting Time\n\n");

  for(time=0,count=0;remaining!=0;)

  {

    if(rt[count]<=time_quantum && rt[count]>0)

    {
      time+=rt[count];
      rt[count]=0;
      flag=1;
    }
    else if(rt[count]>0)
    {
      rt[count]-=time_quantum;
      time+=time_quantum;
    }
    if(rt[count]==0 && flag==1)
    {
      remaining--;
      printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-arrival_time[count],time-arrival_time[count]-
burst_time[count]);
      waiting_time+=time-arrival_time[count]-burst_time[count];
      turn_around_time+=time-arrival_time[count];
      flag=0;
    }
    if(count==n-1)
      count=0;
    else if(arrival_time[count+1]<=time)
      count++;
    else
      count=0;
  }
  printf("\nAverage Waiting Time= %f\n",waiting_time*1.0/n);
  printf("Avg Turnaround Time = %f",turn_around_time*1.0/n);

  return 0;
}
```