**OPERATING SYSTEM**

**CSE 316**

PROJECT REPORT

***By***

**Name : Viven Kumar Jangra**

**Registration No. : 11805470**

**Roll No. : 25**

**Submitted To : Shivali Chopra Ma’am**

**A screenshot of a cell phone

Description automatically generated**

**School of Computer Science Engineering**

**Lovely Professional University, Jalandhar**

April – 2020

**Student Declaration**

This is to declare that this report has been written by me/us. No part of the report is copied from other sources. All information included from other sources have been duly acknowledged. I/We aver that if any part of the report is found to be copied, I/we are shall take full responsibility for it.

Email : vivenkumarjangra@gmail.com

GitHub Link :

**Question 1**

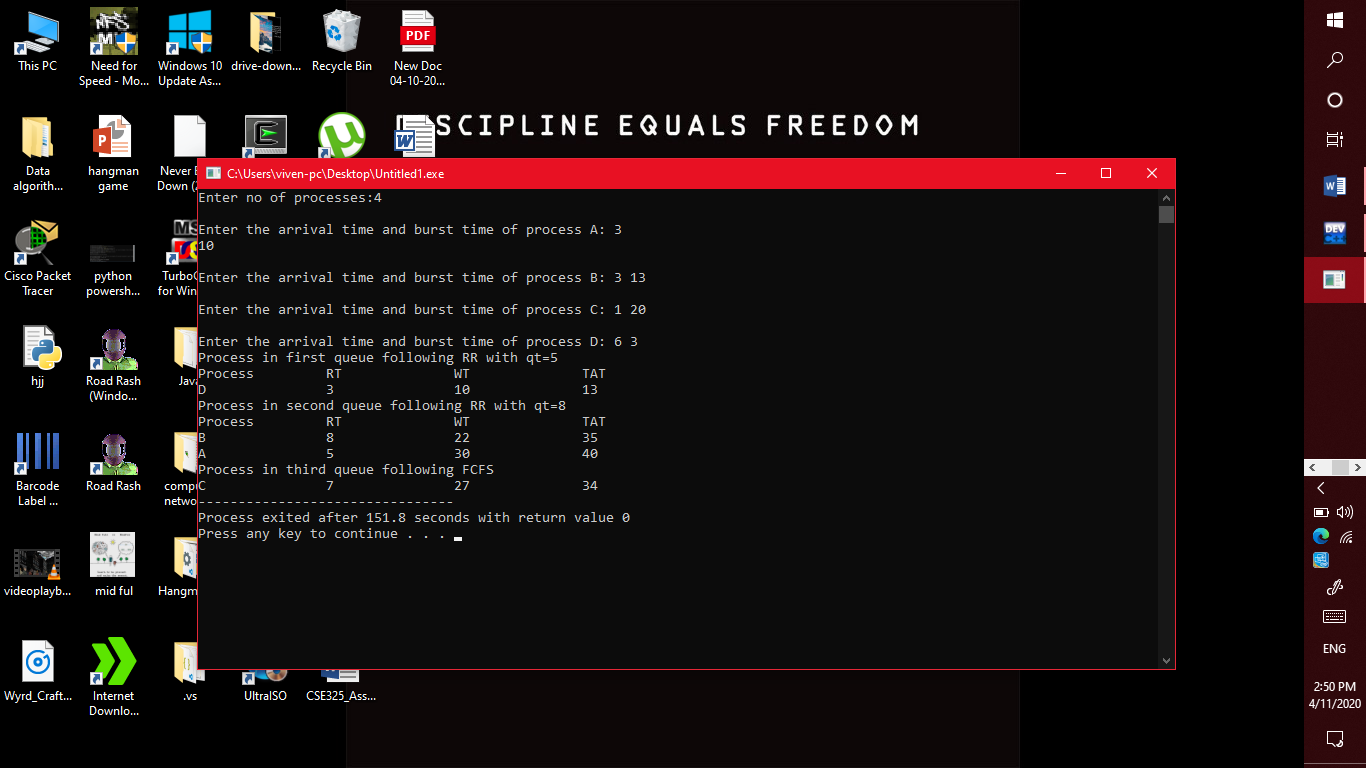
**12. Implement the multi-level feedback queue scheduling algorithm by considering the following diagram: You can use the code of others to implement Roud-Robin, and FCFS but implement aging by your own self.**

****

**Code:**

|  |
| --- |
|  |
| #include<stdio.h> |
| struct process |  |
| { |
| char name; |
| int AT,BT,WT,TAT,RT,CT; |
| }A1[10],A2[10],A3[10]; |
|  |
| int n; |
| void sortByArrival() |
| { |
| struct process temp; |
| int i,j; |
| for(i=0;i<n;i++) |
| { |
| for(j=i+1;j<n;j++) |
| { |
| if(A1[i].AT>A1[j].AT) |
| { |
| temp=A1[i]; |
| A1[i]=A1[j]; |
| A1[j]=temp; |
| } |
| } |
| } |
| } |
|  |
| int main() |
| { |
| int i,j,k=0,r=0,time=0,tq1=5,tq2=8,flag=0; |
| char c; |
| printf("Enter no of processes:"); |
| scanf("%d",&n); |
| for(i=0,c='A';i<n;i++,c++) |
| { |
| A1[i].name=c; |
| printf("\nEnter the arrival time and burst time of process %c: ",A1[i].name); |
| scanf("%d%d",&A1[i].AT,&A1[i].BT); |
| A1[i].RT=A1[i].BT;/\*save burst time in remaining time for each process\*/ |
|  |
| } |
| sortByArrival(); |
| time=A1[0].AT; |
| printf("Process in first queue following RR with qt=5"); |
| printf("\nProcess\t\tRT\t\tWT\t\tTAT\t\t"); |
| for(i=0;i<n;i++) |
| { |
|  |
| if(A1[i].RT<=tq1) |
| { |
|  |
| time+=A1[i].RT; /\*from arrival time of first process to completion of second process\*/ |
| A1[i].RT=0; |
| A1[i].WT=time-A1[i].AT-A1[i].BT;/\*amount of time process has been waiting in the first queue\*/ |
| A1[i].TAT=time-A1[i].AT;/\*amount of time to execute the process\*/ |
| printf("\n%c\t\t%d\t\t%d\t\t%d",A1[i].name,A1[i].BT,A1[i].WT,A1[i].TAT); |
|  |
| } |
| else /\*process moves to queue 2 with qt=8\*/ |
| { |
| A2[k].WT=time; |
| time+=tq1; |
| A1[i].RT-=tq1; |
| A2[k].BT=A1[i].RT; |
| A2[k].RT=A2[k].BT; |
| A2[k].name=A1[i].name; |
| k=k+1; |
| flag=1; |
| } |
| } |
| if(flag==1) |
| {printf("\nProcess in second queue following RR with qt=8"); |
| printf("\nProcess\t\tRT\t\tWT\t\tTAT\t\t"); |
| }for(i=0;i<k;i++) |
| { |
| if(A2[i].RT<=tq2) |
| { |
| time+=A2[i].RT;/\*from arrival time of first process +BT of this process\*/ |
| A2[i].RT=0; |
| A2[i].WT=time-tq1-A2[i].BT;/\*amount of time process has been waiting in the ready queue\*/ |
| A2[i].TAT=time-A2[i].AT;/\*amount of time to execute the process\*/ |
| printf("\n%c\t\t%d\t\t%d\t\t%d",A2[i].name,A2[i].BT,A2[i].WT,A2[i].TAT); |
|  |
| } |
| else /\*process moves to queue 3 with FCFS\*/ |
| { |
| A3[r].AT=time; |
| time+=tq2; |
| A2[i].RT-=tq2; |
| A3[r].BT=A2[i].RT; |
| A3[r].RT=A3[r].BT; |
| A3[r].name=A2[i].name; |
| r=r+1; |
| flag=2; |
| } |
| } |
|  |
| {if(flag==2) |
| printf("\nProcess in 3 queue following FCFS "); |
| } |
| for(i=0;i<r;i++) |
| { |
| if(i==0) |
| A3[i].CT=A3[i].BT+time-tq1-tq2; |
| else |
| A3[i].CT=A3[i-1].CT+A3[i].BT; |
|  |
| } |
|  |
| for(i=0;i<r;i++) |
| { |
| A3[i].TAT=A3[i].CT; |
| A3[i].WT=A3[i].TAT-A3[i].BT; |
| printf("\n%c\t\t%d\t\t%d\t\t%d\t\t",A3[i].name,A3[i].BT,A3[i].WT,A3[i].TAT); |
|  |
| } |
|  |
| } |
|  |
|  |  |

**Output:**



**Description:**

Multi-level feedback queue scheduling algorithm :

In a multilevel queue-scheduling algorithm, processes are permanently assigned to a queue on entry to the system. Processes do not move between queues. This setup has the advantage of low scheduling overhead, but the disadvantage of being inflexible.

Multilevel feedback queue scheduling, however, allows a process to move between queues. The idea is to separate processes with different CPU-burst characteristics. If a process uses too much CPU time, it will be moved to a lower-priority queue. Similarly, a process that waits too long in a lower-priority queue may be moved to a higher-priority queue. This form of aging prevents starvation.

**Algorithm:**

* A1 is round robin with time quantum 5 (RR5),
* A2 is round robin with time quantum 8 (RR8), and
* A3 follows first come first serve (FCFS)
* The process cannot be executed in the lower queue if there are any jobs in all higher queues. For example, A1 has 5 processes, A2 has 1 process, and A3 has 1 process. Then, first the process in A1 should be executed (and completed), and then a process in A2 is executed. Finally, A3 will get CPU resource.
* A new process enters queue A1 which is served RR5. • When it gains CPU, a process receives 5 milliseconds. • If it does not finish in 5 milliseconds, the process is moved to queue A2. • At A2 process is again served RR8 and receives 8 additional milliseconds. • If it still does not complete, it is preempted and moved to queue A3. • At A3 process is executed by first come first serve. • If it still does not complete, it is processed at A2 until completed.

OUTPUT: The remaining time of processes in each queue level, total waiting time and total turnaround time are displayed.

**Question 2**

**15. CPU schedules N processes which arrive at different time intervals and each process is allocated the CPU for a specific user input time unit, processes are scheduled using a preemptive round robin scheduling algorithm. Each process must be assigned a numerical priority, with a higher number indicating a higher relative priority. In addition to the processes one task has priority 0. The length of a time quantum is T units, where T is the custom time considered as time quantum for processing. If a process is preempted by a higher priority process, the preempted process is placed at the end of the queue. Design a scheduler so that the task with priority 0 does not starve for resources and gets the CPU at some time unit to execute. Also compute waiting time, turn around.**

**Code:**

#include<stdio.h>

int main()

{

  int count,i,n,time,remain,flag=0,timequantum;

  int wait\_time=0,turnaround\_time=0,at[10],bt[10],rt[10];

  printf("Enter Total Process:\t ");

  scanf("%d",&n);

  remain=n;

  for(count=0;count<n;count++)

  {

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

    scanf("%d",&at[count]);

    scanf("%d",&bt[count]);

    rt[count]=bt[count];

  }

  printf("Enter Time Quantum:\t");

  scanf("%d",&timequantum);

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

  for(time=0,count=0;remain!=0;)

  {

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

    {

      time+=rt[count];

      rt[count]=0;

      flag=1;

    }

    else if(rt[count]>0)

    {

      rt[count]-=timequantum;

      time+=timequantum;

    }

    if(rt[count]==0 && flag==1)

    {

      remain--;

      printf("P[%d]\t|\t%d\t|\t%d\n",count+1,time-at[count],time-at[count]-bt[count]);

      wait\_time+=time-at[count]-bt[count];

      turnaround\_time+=time-at[count];

      flag=0;

    }

    if(count==n-1)

      count=0;

    else if(at[count+1]<=time)

      count++;

    else

      count=0;

  }

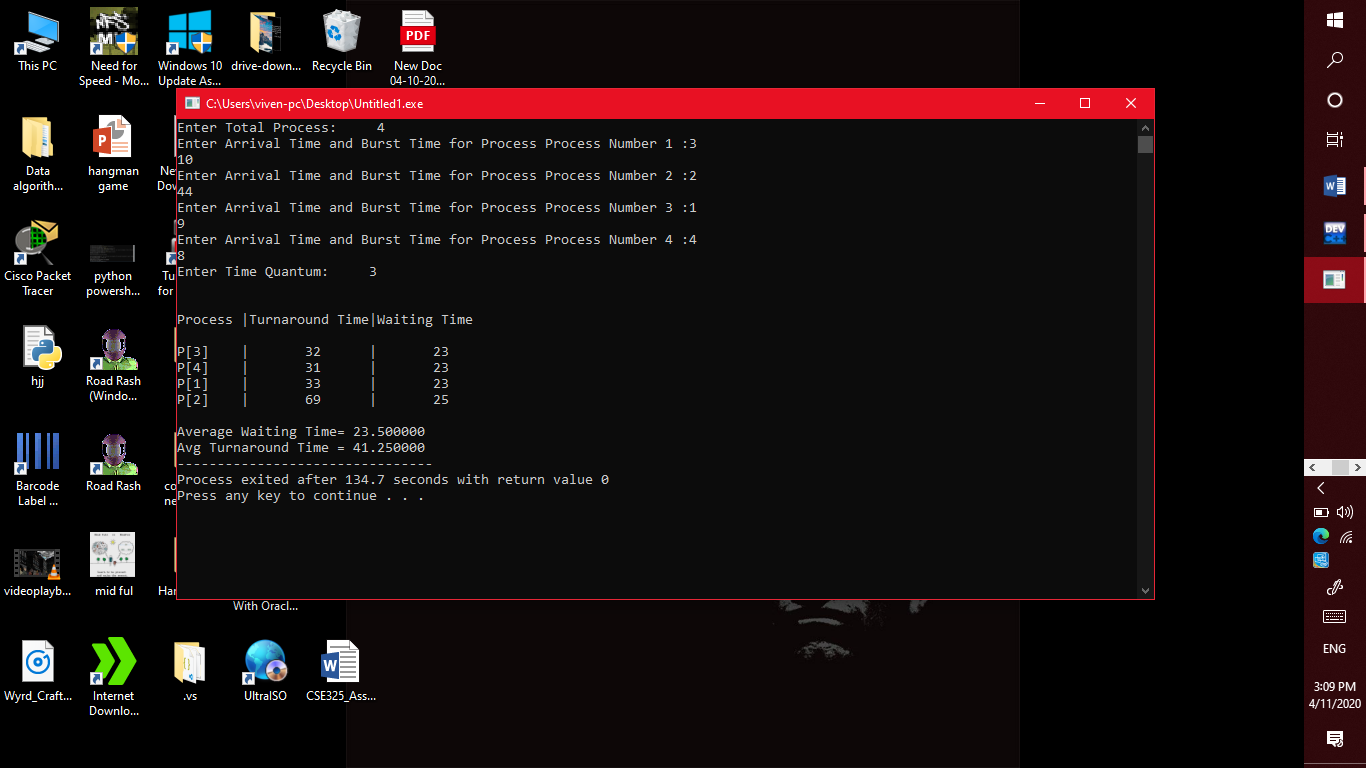
  printf("\nAverage Waiting Time= %f\n",wait\_time\*1.0/n);

  printf("Avg Turnaround Time = %f",turnaround\_time\*1.0/n);

  return 0;

}

**Output:**



**Description:**

Preemptive round robin scheduling algorithm:

Round robin is a CPU scheduling algorithm that is designed especially for time sharing systems. It is more like a FCFS scheduling algorithm with one change that in Round Robin processes are bounded with a quantum time size. A small unit of time is known as Time Quantum or Time Slice. Time quantum can range from 10 to 100 milliseconds. CPU treat ready queue as a circular queue for executing the processes with given time slice. It follows preemptive approach because fixed time are allocated to processes. The only disadvantage of it is overhead of context switching.

**Algorithm:**

* It is simple, easy to implement, and starvation-free as all processes get fair share of CPU.
* One of the most commonly used technique in CPU scheduling as a core.
* It is preemptive as processes are assigned CPU only for a fixed slice of time at most.
* The disadvantage of it is more overhead of context switching.