**Student Name: Swagath Reddy**

**Student ID: 11813094**

**Email Address: malkannagariswagath@gmail.com**

**Question 12**

**Code:**

#include<stdio.h>

#include <stdlib.h>

struct process{

int priority;

int burst\_time;

int pid;

int waiting\_time;

int turnaround\_time;

int remaining\_time;

int arrival\_time;

};

void getInput();

void calcWaitingTime(struct process \*q,int);

void calcTurnAroundTime(struct process \*q,int);

void printQueue(struct process \*q,int size);

void RoundRobin1();

void RoundRobin2();

void FCFS();

void printQueueI(struct process);

void printQueue(struct process \*,int);

int q1\_n=0,q2\_n=0,q3\_n=0,q4\_n=0,n=0;

struct process \*q1,\*q3,\*q4;

int time\_quantum1 = 8;

int time\_quantum2 = 16;

void getInput(){

printf("\n Enter the Total Number of Processes :\t");

scanf("%d",&n);

q1 = (struct process \*)malloc(n\*sizeof(struct process));

q4 = (struct process \*)malloc(n\*sizeof(struct process));

q3 = (struct process \*)malloc(n\*sizeof(struct process));

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

struct process p;

printf("\n\t\tProcess %d\n-----------------------------------------\n\n",i+1); printf("PId:\t");

scanf("%d",&p.pid);

printf("Arrival Time :\t");

scanf("%d",&p.arrival\_time);

printf("Priority (1-9):\t");

scanf("%d",&p.priority);

printf("\nBurst Time: %d\t",p.burst\_time);

scanf("%d",&p.burst\_time);

p.remaining\_time = p.burst\_time;

if(p.priority>0 && p.priority<=3){

q3[q3\_n++] = p;

}else if(p.priority>3 && p.priority<=6){

q4[q4\_n++] = p;

}else{

q1[q1\_n++] = p;

}

}

}

void printQueue(struct process \*q,int size){

calcWaitingTime(q,size);

calcTurnAroundTime(q,size);

printf("\nPId\t\tPriority\t\tBurst Time\t\tWaiting Time\t\tTurnAround Time\t\tArrival");

printf("\n--------------------------------------------------------------------------------\n");

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

printQueueI(q[i]);

}

printf("\n\n");

}

void printQueueI(struct process p){

printf("\n%d\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d\t\t%d",p.pid,p.priority,p.burst\_time,p.waiting\_time,p.turnaround\_time,p.arrival\_time);

}

void calcWaitingTime(struct process \*q,int size){

q[0].waiting\_time = 0;

for(int i=1;i<size;i++){

q[i].waiting\_time = q[i-1].waiting\_time + q[i-1].burst\_time;

}

}

void calcTurnAroundTime(struct process \*q,int size){

q[0].waiting\_time = 0;

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

q[i].turnaround\_time = q[i].waiting\_time + q[i].burst\_time;

}

}

void RoundRobinAlgo1(struct process \*q,int size){

int time=0,i=0,remain=size,flag=0,wait\_time=0,tat\_time=0,total\_times=0;

for(time=0,i=0;remain!=0;){

struct process p = q[i];

if(p.remaining\_time<=time\_quantum1 && p.remaining\_time>0){

time += p.remaining\_time;

p.remaining\_time = 0;

flag = 1;

}else if(p.remaining\_time>time\_quantum1){

p.remaining\_time -= time\_quantum1;

time += time\_quantum1;

}

if(p.remaining\_time==0 && flag==1){

remain--;

printf("\n%d\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d",p.pid,p.priority,p.burst\_time,p.waiting\_time,p.turnaround\_time);

wait\_time += time -p.arrival\_time - p.burst\_time;

tat\_time += time -p.arrival\_time;

flag = 0;

}

if(i==remain-1){

i=0;

}else if(q[i+1].arrival\_time<time){

i++;

}else{

i=0;

}

q[i] = p;

}

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

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

}

void RoundRobinAlgo2(struct process \*q,int size){

int time=0,i=0,remain=size,flag=0,wait\_time=0,tat\_time=0,total\_times=0;

for(time=0,i=0;remain!=0;){

struct process p = q[i];

if(p.remaining\_time<=time\_quantum2 && p.remaining\_time>0){

time += p.remaining\_time;

p.remaining\_time = 0;

flag = 1;

}else if(p.remaining\_time>time\_quantum2){

p.remaining\_time -= time\_quantum2;

time += time\_quantum2;

}

if(p.remaining\_time==0 && flag==1){

remain--;

printf("\n%d\t\t%d\t\t\t%d\t\t\t%d\t\t\t%d",p.pid,p.priority,p.burst\_time,p.waiting\_time,p.turnaround\_time);

wait\_time += time -p.arrival\_time - p.burst\_time;

tat\_time += time -p.arrival\_time;

flag = 0;

}

if(i==remain-1){

i=0;

}else if(q[i+1].arrival\_time<time){

i++;

}else{

i=0;

}

q[i] = p;

}

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

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

}

void RoundRobin1(){

printf("\n\t\t\*\*\tRound Robin1\t\*\*\t\n\n");

printf("\nPId\t\tPriority\t\tBurst Time\t\tWaiting Time\t\tTurnAround Time");

printf("\n---------------------------------------------------------\n");

calcWaitingTime(q3,q3\_n);

calcTurnAroundTime(q3,q3\_n);

RoundRobinAlgo1(q3,q3\_n);

}

void RoundRobin2(){

printf("\n\t\t\*\*\tRound Robin2\t\*\*\t\n\n");

printf("\nPId\t\tPriority\t\tBurst Time\t\tWaiting Time\t\tTurnAround Time");

printf("\n-----n");

calcWaitingTime(q4,q4\_n);

calcTurnAroundTime(q4,q4\_n);

RoundRobinAlgo2(q4,q4\_n);

}

void FCFSAl(struct process \*q,int size){

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

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

if(q[j].arrival\_time>q[i].arrival\_time){

struct process t = q[i];

q[i] = q[j];

q[j] = t;

}

}

}

}

void FCFS(){

printf("\n\t\t\*\*\tFirst Come First Serve\t\*\*\t\n");

FCFSAl(q1,q1\_n);

printQueue(q1,q1\_n);

calcWaitingTime(q1,q1\_n);

calcTurnAroundTime(q1,q1\_n);

}

int main(){

getInput();

int i=1;

while(n>0){

switch(i){

case 3:

FCFS();

break;

case 2:

RoundRobin2();

break;

case 1:

RoundRobin1();

break;

}

i++;

}

printf("\n");

}

/\*test cases

Test case:1 ::

2

5 8 9 2

5 6 4 2

================================================

Test case:2 ::

2

2 4 7 1

6 5 1 3

Explanation:

In this problem I had to implement the multi-level feedback que with the help of round robin and FCFS codes where the process would considered on the basis of its arrival according to the level and also the time quantum of round robin code to make sure none of the processes starve

Constraints :

There were two constraints:-

1 using different time quantum for different levels

2 using FCFS on all the processes

Test case 1:

The first line means that there are two processes in total

The second line specifies the process id for first process

The third line specifies the arrival time of the first process

The fourth line indicates the priority of the current process

The fifth line shows the burst time of the current process

And the same goes for Second process and the other test cases also

**Question 15**

**#include<iostream>**

**#include<string.h>**

**#include<vector>**

**#include<algorithm>**

**using namespace std;**

**struct process**

**{**

**int aT,bT,copy\_bT,pN,cT,wT,tAT;**

**string processId;**

**};**

**int findMin(process [],int);**

**int findIndex(int,string,process []);**

**int m=0,cTsetter=0,checker=0,checker1=0;**

**class Scheduling**

**{**

**int noprocess,var,indef,var1,timer,minPriority,pPI,pIvar,x,previousminpriority;**

**process pro[50];**

**string previouspro;**

**vector<string> q1,q2;**

**vector<int> prioritySelector;**

**public:**

**void getData()**

**{**

**cout<<"\t\t\t--------------------------------------------"<<endl;**

**cout<<"\t\t\t|\t\t\tCPU Scheduling (Multi-Level) |"<<endl;**

**cout<<"\t\t\t|||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||"<<endl<<endl;**

**cout<<" Enter the number of processes :- ";**

**cin>>noprocess;**

**if(noprocess==0)**

**exit(0);**

**int p=65;**

**for(int i=0;i<noprocess;i++)**

**{**

**pro[i].processId=char(p);**

**cout<<" Enter Arrival time || Burst Time || Priority for process ["<<pro[i].processId<<"] respectively"<<endl;**

**cout<<" --> ";**

**cin>>pro[i].aT;**

**cin>>pro[i].bT;**

**pro[i].copy\_bT=pro[i].bT;**

**cin>>pro[i].pN;**

**pro[i].cT=0;**

**p++;**

**}**

**}**

**void findMinInArrivalTime()**

**{**

**int minval;**

**minval=findMin(pro,noprocess);**

**for(int i=0;i<noprocess;i++)**

**{**

**if(pro[i].aT==minval)**

**{**

**q1.push\_back(pro[i].processId);**

**prioritySelector.push\_back(pro[i].pN);**

**}**

**}**

**}**

**void processSelector()**

**{**

**minPriority=\*(std::min\_element(prioritySelector.begin(),prioritySelector.end()));**

**for(int i=0;i<prioritySelector.size();i++)**

**{**

**if(minPriority==prioritySelector.at(i))**

**{**

**var=i;**

**break;**

**}**

**}**

**for(int i=0;i<noprocess;i++)**

**{**

**if(pro[i].processId==q1[var])**

**{**

**var1=i;**

**}**

**}**

**if(m>=1)**

**{**

**if(q1[var]!=previouspro)**

**{**

**q1.erase(std::remove(q1.begin(),q1.end(),previouspro),q1.end());**

**if(checker==0)**

**{**

**prioritySelector.erase(prioritySelector.begin()+pIvar,prioritySelector.begin()+(pIvar+1));**

**}**

**else**

**{**

**checker=0;**

**}**

**if(pro[pPI].bT!=0)**

**{**

**q2.push\_back(previouspro);**

**}**

**}**

**}**

**m+=1;**

**pIvar=var;**

**previouspro=q1[var];**

**previousminpriority=minPriority;**

**pPI=var1;**

**if(checker1==1)**

**{**

**startQ1Process();**

**}**

**}**

**void startQ1Process()**

**{**

**while(q1.size()!=0)**

**{**

**checker1+=1;**

**int con=0;**

**if(pro[var1].bT==0)**

**{**

**for(int i=0;i<q1.size();i++)**

**{**

**if(q1.at(i)==pro[var1].processId)**

**{**

**indef=i;**

**}**

**}**

**q1.erase(std::remove(q1.begin(),q1.end(),q1.at(indef)),q1.end());**

**prioritySelector.erase(prioritySelector.begin()+indef,prioritySelector.begin()+(indef+1));**

**checker=1;**

**if(q1.size()==0)**

**{**

**checker1=0;**

**checker=0;**

**break;**

**}**

**processSelector();**

**}**

**if(cTsetter==0)**

**{**

**pro[var1].cT=pro[var1].aT;**

**cTsetter=1;**

**}**

**else**

**{**

**pro[var1].cT=timer;**

**}**

**pro[var1].bT-=1;**

**pro[var1].cT+=1;**

**timer=pro[var1].cT;**

**for(int i=0;i<noprocess;i++)**

**{**

**if(timer==pro[i].aT)**

**{**

**q1.push\_back(pro[i].processId);**

**prioritySelector.push\_back(pro[i].pN);**

**con=1;**

**}**

**}**

**if(con==1)**

**{**

**processSelector();**

**}**

**}**

**if(q2.size()!=0)**

**{**

**startQ2process();**

**}**

**}**

**void startQ2process()**

**{**

**int TQ=2,index2;**

**string pI,preprocess;**

**pI=q2.at(0);**

**index2=findIndex(noprocess,pI,pro);**

**while(q2.size()!=0)**

**{**

**pro[index2].cT=timer;**

**if(pro[index2].bT==1)**

**{**

**pro[index2].bT-=1;**

**pro[index2].cT+=1;**

**}**

**else**

**{**

**pro[index2].bT-=TQ;**

**pro[index2].cT+=TQ;**

**}**

**timer=pro[index2].cT;**

**if(pro[index2].bT==0)**

**{**

**q2.erase(std::remove(q2.begin(),q2.end(),pro[index2].processId),q2.end());**

**}**

**else**

**{**

**preprocess=pro[index2].processId;**

**q2.erase(std::remove(q2.begin(),q2.end(),pro[index2].processId),q2.end());**

**q2.push\_back(preprocess);**

**}**

**for(int i=0;i<noprocess;i++)**

**{**

**if(timer==pro[i].aT)**

**{**

**q1.push\_back(pro[i].processId);**

**prioritySelector.push\_back(pro[i].pN);**

**}**

**}**

**if(q1.size()!=0)**

**{**

**checker1=1;**

**m=0;**

**processSelector();**

**}**

**if(q2.size()==0)**

**{**

**break;**

**}**

**pI=q2.at(0);**

**index2=findIndex(noprocess,pI,pro);**

**}**

**}**

**void check()**

**{**

**x=pro[0].aT;**

**for(int i=0;i<noprocess;i++)**

**{**

**if(pro[i].aT>x)**

**{**

**x=pro[i].aT;**

**}**

**}**

**while(timer<=x)**

**{**

**int cod=0;**

**timer++;**

**for(int i=0;i<noprocess;i++)**

**{**

**if(timer==pro[i].aT)**

**{**

**q1.push\_back(pro[i].processId);**

**prioritySelector.push\_back(pro[i].pN);**

**cod=1;**

**}**

**}**

**if(cod==1)**

**{**

**m=0;**

**checker1=1;**

**processSelector();**

**}**

**}**

**}**

**void display()**

**{**

**cout<<" Data set given by User"<<endl;**

**cout<<" +-------------------------------------------------------------------------------------------------------------+"<<endl;**

**cout<<" | ProcessID \t\t\tArrival Time\t\t\tBurst Time\t\t\tPriority |"<<endl;**

**cout<<" +-------------------------------------------------------------------------------------------------------------+"<<endl;**

**for(int i=0;i<noprocess;i++)**

**{**

**cout<<" | "<<pro[i].processId<<"\t\t\t\t"<<pro[i].aT<<"\t\t\t\t"<<pro[i].bT<<"\t\t\t\t"<<pro[i].pN<<"\t |"<<endl;**

**}**

**cout<<" +-------------------------------------------------------------------------------------------------------------+"<<endl<<endl;**

**}**

**void finalResult()**

**{**

**float avgwT=0;**

**cout<<" +\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*+"<<endl;**

**cout<<" |\t\t\t\t\tFinal result timing table of processes\t\t\t\t\t |"<<endl;**

**cout<<" +\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*+"<<endl;**

**cout<<" | PId"<<"\t||\t"<<"AT"<<"\t||\t"<<"BT"<<"\t||\t"<<"CT"<<"\t||\t"<<"TAT"<<"\t||\t "<<"WT"<<"\t\t |"<<endl;**

**cout<<" +\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*+"<<endl;**

**for(int i=0;i<noprocess;i++)**

**{**

**pro[i].tAT=pro[i].cT-pro[i].aT;**

**pro[i].wT=pro[i].tAT-pro[i].copy\_bT;**

**avgwT+=pro[i].wT;**

**cout<<" | "<<pro[i].processId<<"\t\t||\t"<<pro[i].aT<<"\t||\t"<<pro[i].copy\_bT<<"\t||\t"<<pro[i].cT<<"\t||\t"<<pro[i].tAT<<"\t||\t "<<pro[i].wT<<"\t\t |"<<endl;**

**cout<<" +-------------------------------------------------------------------------------------------------------------+"<<endl;**

**}**

**cout<<" |\t\t\t\t\tAverage waiting time : "<<avgwT/noprocess<<"\t\t\t\t\t\t|"<<endl;**

**cout<<" +-------------------------------------------------------------------------------------------------------------+"<<endl;**

**}**

**};**

**int main()**

**{**

**Scheduling sc;**

**sc.getData();**

**sc.display();**

**sc.findMinInArrivalTime();**

**sc.processSelector();**

**sc.startQ1Process();**

**sc.check();**

**sc.finalResult();**

**}**

**int findMin(process pro[], int noprocess)**

**{**

**int mn=pro[0].aT;**

**for(int i=0;i<noprocess;i++)**

**{**

**int temp=pro[i].aT;**

**mn=std::min(mn,temp);**

**}**

**return mn;**

**}**

**int findIndex(int noprocess,string pI,process pro[])**

**{**

**int in;**

**for(int i=0;i<noprocess;i++)**

**{**

**if(pI==pro[i].processId)**

**{**

**in=i;**

**}**

**}**

**return in;**

**}**

**/\***

**Test case:1 ::**

**4**

**5 6 2**

**5 4 1**

**8 2 7**

**33 2 9**

**================================================**

**Test case:2 ::**

**4**

**0 9 4**

**2 6 1**

**7 7 2**

**6 2 4**

**==================================================**

**Test case: 3 ::**

**4**

**0 3 2**

**4 5 2**

**18 9 5**

**2 8 5**

**==================================================**

**Test case: 4 ::**

**5**

**0 5 7**

**11 5 7**

**4 8 12**

**6 12 8**

**11 3 5**

**\*/**

Explanation:

In this problem I had to implement 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.

Constraints :

There were constraints:-

1. There should be a priority given to each process
2. One of the processes should be given priority 0
3. The length of time quantum should be provided
4. None of the processes should starve

Test case 1:

The first line shows the number of processes in total

The second line specifies the Arrival time ,Burst time and priority for first process 1

The third line specifies the Arrival time ,Burst time and priority for first process 2

The fourth line specifies the Arrival time ,Burst time and priority for first process 3

And so onn for all the processes