**LOVELY PROFESSIONAL UNIVERSITY**

**Academic Task-3**

**(Operating System)**

School of Computer Science and Engineering( Faculty of Technology And Sciences )

**Name of the faculty member: Ashu**

**Course Code: CSE 316 Course Title:** Operating System

**Term:**219202

Date of Allotment: 01/03/2020 Date of Submission: 31/03/2020

**Name: Lavkush singh**

**Reg No:** 11805239

**Roll No: 07**

**Section:** K18UW

**Email Address:** ls8853083440@gmail.com

**GitHub Link:** <https://github.com/singh1512000lav/Operating_system_Project>

**Problem** **1**:

Design a scheduling program to implements a Queue with two levels:

Level 1 : Fixed priority preemptive Scheduling

Level 2 : Round Robin Scheduling

For a Fixed priority preemptive Scheduling (Queue 1), the Priority 0 is highest priority. If one process P1 is scheduled and running, another process P2 with higher priority comes. The New process (high priority) process P2 preempts currently running process P1 and process P1 will go to second level queue. Time for which process will strictly execute must be considered in the multiples of 2..

All the processes in second level queue will complete their execution according to round robin scheduling.

Consider: 1. Queue 2 will be processed after Queue 1 becomes empty.

2. Priority of Queue 2 has lower priority than in Queue 1.

**Description:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::**

To schedule process we have two methods,one is preemptive and the other is non-preemptive,

in these two methods of scheduling the need of leaving a process incomplete before completion

is known as preemptive,whereas the process of completing the processing and then leaving the

processor is known as non-preemptive .

Explanation-

Preemptive methode consists of the round robin algorithm,where the processess are given a particular time slice for getting there processe done by the processor ,and different processes are known by p1 and p2.

**Algorithm::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::**

**#include <bits/stdc++.h>**

**using namespace std;**

**struct Process\_Data**

**{**

**int Num;**

**int ID;**

**int ARI\_time;**

**int BUR\_time;**

**int Priority;**

**int FINISH\_time;**

**int Remaining\_time;**

**int WAIT\_time;**

**int START\_time;**

**int Responce\_time;**

**};**

**struct Process\_Data cur;**

**typedef struct Process\_Data pr\_da ;**

**bool idsort(const pr\_da& x , const pr\_da& y)**

**{**

**return x.ID < y.ID;**

**}**

**/\*\* base of arrival\_time \*\*/**

**bool arrivalsort( const pr\_da& x ,const pr\_da& y)**

**{**

**if(x.ARI\_time < y.ARI\_time)**

**return true;**

**else if(x.ARI\_time > y.ARI\_time)**

**return false;**

**if(x.Priority < y.Priority)**

**return true;**

**else if(x.Priority > y.Priority)**

**return false;**

**if(x.ID < y.ID)**

**return true;**

**return false;**

**}**

**bool Numsort( const pr\_da& x ,const pr\_da& y)**

**{**

**return x.Num < y.Num;**

**}**

**/\* base of Priority if that same then on the base of ID\*/**

**struct comPare**

**{**

**bool operator()(const pr\_da& x ,const pr\_da& y)**

**{**

**if( x.Priority > y.Priority )**

**return true;**

**else if( x.Priority < y.Priority )**

**return false;**

**if( x.ID > y.ID )**

**return true;**

**return false;**

**}**

**};**

**/\*\*To check the Input \*\*/**

**void my\_check(vector<pr\_da> mv)**

**{**

**for(unsigned int t= 0; t < mv.size() ;t++)**

**{**

**cout<<" ID :"<<mv[t].ID<<" \_time : "<<mv[t].ARI\_time<<" BUR\_time : "<<mv[t].BUR\_time<<" Priority : "<<mv[t].Priority<<endl;**

**}**

**}**

**int main()**

**{**

**int i;**

**vector< pr\_da > input;**

**vector<pr\_da> input\_copy;**

**pr\_da temp;**

**int pq\_process = 0;**

**int rq\_process = 0;**

**int ARI\_time;**

**int BUR\_time;**

**int ID;**

**int Priority;**

**int n;**

**int clock;**

**int total\_exectiontime = 0;**

**cout<<"\n\*\* Enter the number of processes : ";**

**cin>>n;**

**int j=1;**

**for( i= 0; i< n; i++ )**

**{**

**cout<<"\n\nEnter details of process ["<<j<<"] :";**

**cout<<"\n\n-Arrival Time :- ";**

**cin>>ARI\_time;**

**cout<<"-Burst Time :- ";**

**cin>>BUR\_time;**

**cout<<"-Priority :- ";**

**cin>>Priority;**

**temp.Num = i+1;**

**temp.ARI\_time = ARI\_time;**

**temp.BUR\_time = BUR\_time;**

**temp.Remaining\_time = BUR\_time;**

**temp.ID = j;**

**temp.Priority = Priority;**

**input.push\_back(temp);**

**j=j+1;**

**}**

**input\_copy = input;**

**sort( input.begin(), input.end(), arrivalsort );**

**total\_exectiontime = total\_exectiontime + input[0].ARI\_time;**

**for( i= 0 ;i< n; i++ )**

**{**

**if( total\_exectiontime >= input[i].ARI\_time )**

**{**

**total\_exectiontime = total\_exectiontime +input[i].BUR\_time;**

**}**

**else**

**{**

**int diff = (input[i].ARI\_time - total\_exectiontime);**

**total\_exectiontime = total\_exectiontime + diff + BUR\_time;**

**}**

**}**

**int Ghant[total\_exectiontime]={0};**

**for( i= 0; i< total\_exectiontime; i++ )**

**{**

**Ghant[i]=-1;**

**}**

**priority\_queue < pr\_da ,vector<Process\_Data> ,comPare> pq;**

**queue< pr\_da > rq;**

**int cpu\_state = 0;**

**int quantum = 4 ;**

**cur.ID = -2;**

**cur.Priority = 999999;**

**for ( clock = 0; clock< total\_exectiontime; clock++ )**

**{**

**/\*\*Insert the process with same Arrival time in Priority Queue\*\*/**

**for( int j = 0; j< n ; j++ )**

**{**

**if(clock == input[j].ARI\_time)**

**{**

**pq.push(input[j]);**

**}**

**}**

**if(cpu\_state == 0)**

**{**

**if(!pq.empty())**

**{**

**cur = pq.top();**

**cpu\_state = 1;**

**pq\_process = 1;**

**pq.pop();**

**quantum = 4;**

**}**

**else if(!rq.empty())**

**{**

**cur = rq.front();**

**cpu\_state = 1;**

**rq\_process = 1;**

**rq.pop();**

**quantum = 4;**

**}**

**}**

**else if(cpu\_state == 1)**

**{**

**if(pq\_process == 1 && (!pq.empty()))**

**{**

**if(pq.top().Priority < cur.Priority )**

**{**

**rq.push(cur);**

**cur = pq.top();**

**pq.pop();**

**quantum = 4;**

**}**

**}**

**else if(rq\_process == 1 && (!pq.empty()))**

**{**

**rq.push(cur);**

**cur = pq.top();**

**pq.pop();**

**rq\_process = 0;**

**pq\_process = 1;**

**quantum = 4 ;**

**}**

**}**

**if(cur.ID != -2)**

**{**

**cur.Remaining\_time--;**

**quantum--;**

**Ghant[clock] = cur.ID;**

**if(cur.Remaining\_time == 0)**

**{**

**cpu\_state = 0 ;**

**quantum = 4 ;**

**cur.ID = -2;**

**cur.Priority = 999999;**

**rq\_process = 0;**

**pq\_process = 0;**

**}**

**else if(quantum == 0 )**

**{**

**rq.push(cur);**

**cur.ID = -2;**

**cur.Priority = 999999;**

**rq\_process = 0;**

**pq\_process = 0;**

**cpu\_state=0;**

**}**

**}**

**}**

**sort( input.begin(), input.end(), idsort );**

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

**{**

**for(int k=total\_exectiontime;k>=0;k--)**

**{**

**if(Ghant[k]==i+1)**

**{**

**input[i].FINISH\_time=k+1;**

**break;**

**}**

**}**

**}**

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

**{**

**for(int k=0;k<total\_exectiontime;k++)**

**{**

**if(Ghant[k]==i+1)**

**{**

**input[i].START\_time=k;**

**break;**

**}**

**}**

**}**

**sort( input.begin(), input.end(), Numsort );**

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

**{**

**input[i].Responce\_time=input[i].START\_time-input[i].ARI\_time;**

**input[i].WAIT\_time=(input[i].FINISH\_time-input[i].ARI\_time)-input[i].BUR\_time;**

**}**

**cout<<"\n\n\n\n\*\* Now Processes According to Priority given in Problem \*\*";**

**cout<<"\n\n----------------------------------------------------------\n";**

**cout<<" process Responce\_time Finish\_time Waiting\_time \n";**

**cout<<"----------------------------------------------------------\n";**

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

**{**

**cout<<" "<<input[i].ID<<" "<<input[i].Responce\_time<<" "<<input[i].FINISH\_time<<" "<<input[i].WAIT\_time<<endl;**

**}**

**cout<<"----------------------------------------------------------\n\n\n\n";**

**return 0;**

**}**