

|| OPERATING SYSTEM ||

ASSIGNMENT

**SUBMITTED TO** **:**

NAME : MR. Manpreet Singh

**SUBMITTED BY :**

NAME : J.VENKATA SAI SUDHEER REDDY

REG NO : 11703680

SECTION : K17GB

ROLL NO : A30

**ALLOTTED QUESTION AND WORKING OF FUNCTIONS:**

Q4. 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.

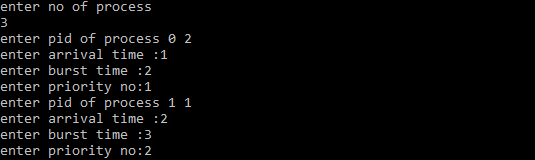
2. Priority of Queue 2 has lower priority than in Queue 1.

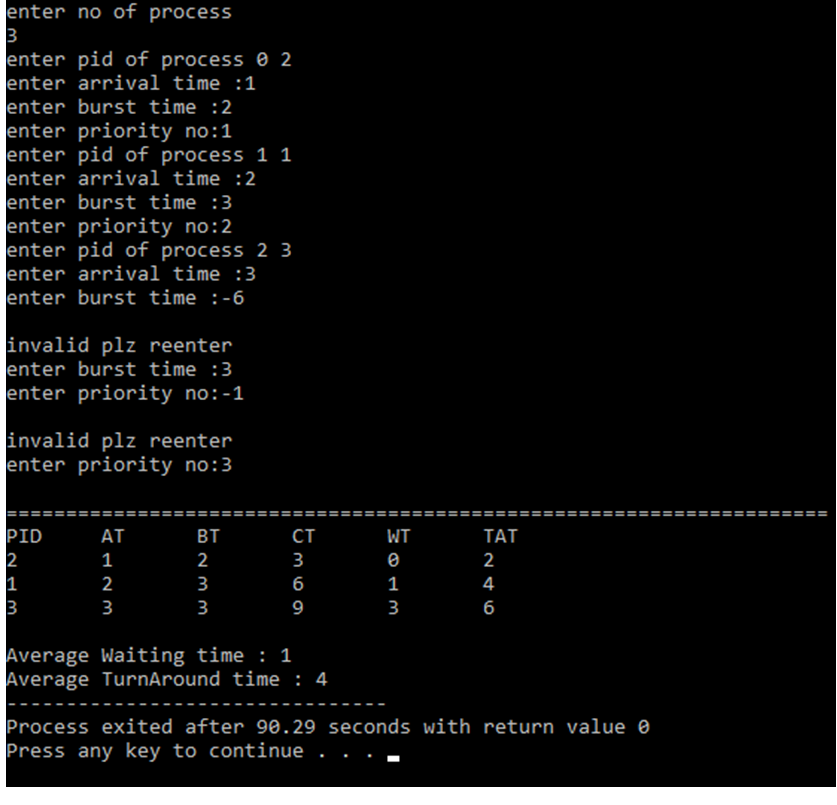
Working:

At first the user have to enter the number of process and then for each process user have to enter the pid,arrival time,burst time and priority.if user enter the invalid data it will again go to the previous step and asks user to reenter again.

I have created queue it stores the processes.queue1 is fixed priority preemptive scheduling and queue 2 is round robin scheduling.The time quantum for RR is initialized as 4.it start processing the algorithms and final it calculates the completion time,waiting time and turnaround time.

Output format:





SOURCE CODE:

#include<stdio.h>

#include<iostream>

#include<queue>

using namespace std;

#define MAX 1000

#define TQ 4

int flag[MAX],at[MAX],bt[MAX],pt[MAX],rt[MAX],ct[MAX],fe[MAX],fe\_flag[MAX],pid[MAX],tms,qt[MAX];

//at arrival time

//bt burst time

//rt Response Time

//pt priority

//pid process id

//ct completion time

queue<int> q;

void RR()

{

if(!q.empty())

{

if(rt[q.front()]>0 && qt[q.front()]<4)

{

qt[q.front()]++;

rt[q.front()]--;

if(rt[q.front()]==0)

{

ct[q.front()]=tms+1;

q.pop();

}

if(rt[q.front()]!=0 && qt[q.front()]==4)

{

qt[q.front()]=0;

q.push(q.front());

q.pop();

}

}

}

}

int main()

{

int i=0,n=0,smallest=0,last\_smallest=-1,min,sum=0,large=0,avgWT=0,avgTAT=0;

printf("enter no of process\n");

scanf("%d",&n);

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

{

printf("enter pid of process %d ",i);

scanf("%d",&pid[i]);

A:

printf("enter arrival time :");

scanf("%d",&at[i]);

if(at[i]<0)

{

printf("\ninvalid plz reenter\n");

goto A;

}

B:

printf("enter burst time :");

scanf("%d",&bt[i]);

if(bt[i]<1)

{

printf("\ninvalid plz reenter\n");

goto B;

}

C:

printf("enter priority no:");

scanf("%d",&pt[i]);

if(pt[i]<0)

{

printf("\ninvalid plz reenter\n");

goto C;

}

if(at[i]>large)

large=at[i];

sum+=bt[i];

rt[i]=bt[i];

}

min=MAX;

for(tms=0;!q.empty() || tms<=sum+large ;tms++)

{

min=MAX;

smallest=-1;

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

{

if(at[i]<=tms && pt[i]<min && rt[i]>0 && !flag[i])

{

min=pt[i];

smallest=i;

}

}

if(smallest==-1 && !q.empty())

{

if(last\_smallest !=-1 && rt[last\_smallest]==0)

{

ct[last\_smallest]=tms;

flag[last\_smallest]=1;

}

last\_smallest=-1;

RR();

continue;

}

else if(smallest!=-1 && !q.empty() && last\_smallest==-1)

{

if(qt[q.front()]<=4 && qt[q.front()]>0)

{

q.push(q.front());

q.pop();

}

}

if(smallest!=-1 && !fe\_flag[smallest])

{

fe[smallest]=tms-at[smallest];

fe\_flag[smallest]=1;

}

if( smallest!=last\_smallest && last\_smallest!=-1 && !flag[last\_smallest])

{

q.push(last\_smallest);

flag[last\_smallest]=1;

}

if(smallest !=-1)

rt[smallest]--;

if((smallest !=-1) && ((rt[smallest]==0) ||(bt[smallest]-rt[smallest])==TQ))

{

if((bt[smallest]-rt[smallest])==TQ && rt[smallest]!=0)

{

flag[smallest]=1;

q.push(smallest);

}

else if(smallest!=-1)

{

ct[smallest]=tms+1;

flag[smallest]=1;

}

}

last\_smallest=smallest;

}

cout<<"\n=====================================================================\n";

cout<<"PID\tAT\tBT\tCT\tWT\tTAT\n";

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

cout<<pid[i]<<"\t"<<at[i]<<"\t"<<bt[i]<<"\t"<<ct[i]<<"\t"<<ct[i]-bt[i]-at[i]<<"\t"<<ct[i]-at[i]<<endl;

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

avgWT=avgWT+ct[i]-bt[i]-at[i];

avgTAT=avgTAT+ct[i]-at[i];

}

cout<<"\nAverage Waiting time : "<<avgWT/n;

cout<<"\nAverage TurnAround time : "<<avgTAT/n;

return 0;

}