**Student Name:** Ankit Kumar Sinha

**Student ID:** 1707333(A24)

**Email Address:** [ankit.nick1543@gmail.com](mailto:ankit.nick1543@gmail.com)

**GitHub Link:**

**Problem:**

**Ques1.** Design a scheduling program to implements a Queue with two levels:  
Level 1 : Fixed priority preemptive Scheduling  
Level 2: Round Robin SchedulingFor a Fixed priority preemptive Scheduling (Queue1), 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.

**Algorithm:**

**Description:**

**Code:**

#include<stdio.h>

#define N 10

//pid is process id

//pp is process priority

//bt is process burst time

//at is process arrival time

typedef struct

{

int pid, at,bt,pp;

int q, ready;

}process\_structure;

int Queue(int t1)

{

if(t1==0 || t1==1 || t1==2 || t1==3)

{

return 1;

}

else

{

return 2;

}

}

int main()

{

int limit, count, temp\_process, time, j, y;

process\_structure temp;

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

scanf("%d",&limit);

process\_structure process[limit];

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

{

printf("\nProcess ID:\t");

scanf("%d", &process[count].pid);

printf("Arrival Time: ");

scanf("%d", &process[count].at);

printf("Burst Time: ");

scanf("%d", &process[count].bt);

printf("Process Priority: ");

scanf("%d", &process[count].pp);

temp\_process = process[count].pp;

process[count].q = Queue(temp\_process);

process[count].ready = 0;

}

time = process[0].bt;

for(y=0;y<limit;y++)

{

for(count=y;count<limit;count++)

{

if(process[count].at < time)

{

process[count].ready = 1;

}

}

for(count=y;count<limit-1;count++)

{

for(j=count+1; j<limit; j++)

{

if(process[count].ready == 1 && process[j].ready == 1)

{

if(process[count].q == 2 && process[j].q == 1)

{

temp = process[count];

process[count] = process[j];

process[j] = temp;

}

}

}

}

for(count=y;count<limit-1;count++)

{

for(j=count+1;j<limit;j++)

{

if(process[count].ready==1 && process[j].ready==1)

{

if(process[count].q==1 && process[j].q==1)

{

if(process[count].bt > process[j].bt)

{

temp=process[count];

process[count]=process[j];

process[j]=temp;

}

else

{

break;

}

}

}

}

}

printf("\nProcess[%d]:\tTime:\t%d To %d\n", process[y].pid, time, time + process[y].bt);

time = time + process[y].bt;

for(count=y; count<limit; count++)

{

if(process[count].ready==1)

{

process[count].ready=0;

}

}

}

return 0;

}

