Assignment (OS)

Name: Vaibhav Pandilwar

Roll No. 38

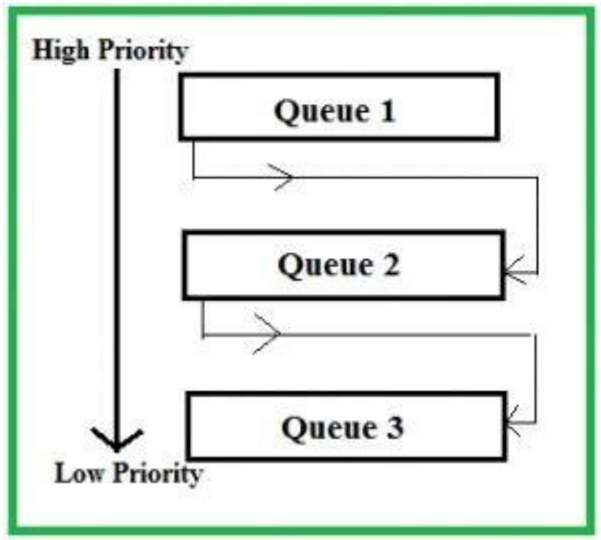
Reg No. 11813612

Section : K18ZV

group :2

Question no.12

This Scheduling is like Multilevel Queue(MLQ) Scheduling but in this process can move between the queues. **Multilevel Feedback Queue Scheduling (MLFQ)** keep analyzing the behavior (time of execution) of processes and according to which it changes its priority.Now, look at the diagram and explanation below to understand it properly.



Now let us suppose that queue 1 and 2 follow round robin with time quantum 4 and 8 respectively and queue 3 follow FCFS.One implementation of MFQS is given below –

1. When a process starts executing then it first enters queue1.

2. Inqueue1processexecutesfor4unitandifitcompletesinthis4unitoritgivesCPUfor I/O operation in this 4 unit than the priority of this process does not change and if it again comes in the ready queue than it again starts its execution in Queue1.
3. Ifaprocessinqueue1doesnotcompletein4unitthenitsprioritygetsreducedandit shifted to queue2.
4. Above points 2 and 3 are also true for queue 2 processes but the time quantum is 8 unit.Inageneralcaseifaprocessdoesnotcompleteinatimequantumthanitisshifted to the lower priorityqueue.
5. In the last queue, processes are scheduled in FCFSmanner.
6. Aprocessinlowerpriorityqueuecanonlyexecuteonlywhenhigherpriorityqueuesare empty.
7. Aprocessrunninginthelowerpriorityqueueisinterruptedbyaprocessarrivinginthe higher priorityqueue.

Well, above implementation may differ for example the last queue can also follow Round-robin Scheduling.

Problems in the above implementation – A process in the lower priority queue can suffer from starvation due to some short processes taking all the CPU time.

Solution – A simple solution can be to boost the priority of all the process after regular intervals and place them all in the highest priority queue.

What is the need of such complex Scheduling?

- Firstly, it is more flexible than the multilevel queuescheduling.
- To optimize turnaround time algorithms like SJF is needed which require the running time of processes to schedule them. But the running time of the process is not known in advance.MFQSrunsaprocessforatimequantumandthenitcanchangeitspriority(ifit is a long process). Thus it learns from past behavior of the process and then predicts its future behavior.This way it tries to run shorter process first thus optimizing turnaround time.
- MFQS also reduces the responsetime.

Example –

Consider a system which has a CPU bound process, which requires the burst time of 40 seconds.The multilevel Feed Back Queue scheduling algorithm is used and the queue time quantum ‘2’ seconds and in each level it is incremented by ‘5’ seconds.Then how many times the process will be interrupted and on which queue the process will terminate the execution?

Solution –

Process P needs 40 Seconds for total execution.

At Queue 1 it is executed for 2 seconds and then interrupted and shifted to queue 2.

At Queue 2 it is executed for 7 seconds and then interrupted and shifted to queue 3.

At Queue 3 it is executed for 12 seconds and then interrupted and shifted to queue 4.

At Queue 4 it is executed for 17 seconds and then interrupted and shifted to queue 5.

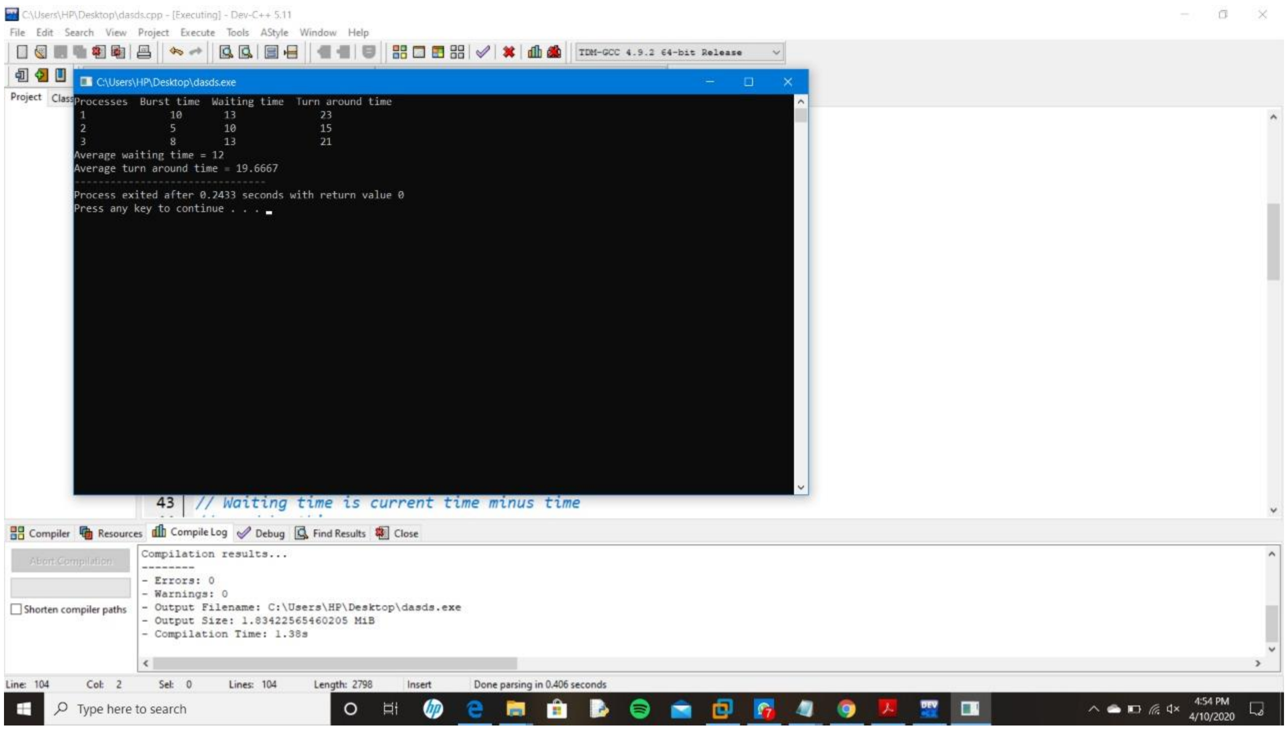
At Queue 5 it executes for 2 seconds and then it completes.

Hence the process is interrupted 4 times and completes on queue 5.

Code for above implementation

```
// C++ program for implementation of RR scheduling
#include<iostream>
using namespace std;

// Function to find the waiting time for all
// processes
void findWaitingTime(int processes[], int n,
                     int bt[], int wt[], int quantum)
{
   // Make a copy of burst times bt[] to store remaining
   // burst times.
   int rem_bt[n];
   for (int i = 0 ; i < n ; i++)
          rem_bt[i] = bt[i];

   int t = 0; // Current time

   // Keep traversing processes in round robin manner
   // until all of them are not done.
   while (1)
   {
          bool done = true;

          // Traverse all processes one by one repeatedly
          for (int i = 0 ; i < n; i++)
          {
                   // If burst time of a process is greater than 0
```

```
                // then only need to process further
                if (rem_bt[i] > 0)
                {
                        done = false; // There is a pending process

                        if (rem_bt[i] > quantum)
                        {
                                // Increase the value of t i.e. shows
                                // how much time a process has been processed
                                t += quantum;

                                // Decrease the burst_time of current process
                                // by quantum
                                rem_bt[i] -= quantum;
                        }

                        // If burst time is smaller than or equal to
                        // quantum. Last cycle for this process
                        else
                        {
                                // Increase the value of t i.e. shows
                                // how much time a process has been processed
                                t = t + rem_bt[i];

                                // Waiting time is current time minus time
                                // used by this process
                                wt[i] = t - bt[i];

                                // As the process gets fully executed
                                // make its remaining burst time = 0
                                rem_bt[i] = 0;
                        }
                }
        }

        // If all processes are done
        if (done == true)
        break;
  }
}

// Function to calculate turn around time
void findTurnAroundTime(int processes[], int n,
```



C:\Users\HP\Desktop\dasds.cpp - [Executing] - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
C:\Users\HP\Desktop\dasds.exe
Compiler Resources Compile Log Debug Find Results Close
Abort Compilation
Shorten compiler paths
Line: 104 Col: 2 Sel: 0 Lines: 104 Length: 2798 Insert Done parsing in 0.406 seconds
Type here to search
4:54 PM 4/10/2020

```
Processes  Burst time  Waiting time  Turn around time
 1          10          13              23
 2          5           10              15
 3          8           13              21
Average waiting time = 12
Average turn around time = 19.6667
--------------------------------
Process exited after 0.2433 seconds with return value 0
Press any key to continue . . .
```

```
43  // Waiting time is current time minus time
```

```
Compilation results...
--------
- Errors: 0
- Warnings: 0
- Output Filename: C:\Users\HP\Desktop\dasds.exe
- Output Size: 1.83422565460205 MiB
- Compilation Time: 1.38s
```

```
                                        int bt[], int wt[], int tat[])
{
    // calculating turnaround time by adding
    // bt[i] + wt[i]
    for (int i = 0; i < n ; i++)
            tat[i] = bt[i] + wt[i];
}

// Function to calculate average time
void findavgTime(int processes[], int n, int bt[],
                                                          int quantum)
{
    int wt[n], tat[n], total_wt = 0, total_tat = 0;

    // Function to find waiting time of all processes
    findWaitingTime(processes, n, bt, wt, quantum);

    // Function to find turn around time for all processes
    findTurnAroundTime(processes, n, bt, wt, tat);

    // Display processes along with all details
    cout << "Processes "<< " Burst time "
```

```
            << " Waiting time " << " Turn around time\n";

    // Calculate total waiting time and total turn
    // around time
    for (int i=0; i<n; i++)
    {
            total_wt = total_wt + wt[i];
            total_tat = total_tat + tat[i];
            cout << " " << i+1 << "\t\t" << bt[i] <<"\t "
                    << wt[i] <<"\t\t " << tat[i] <<endl;
    }

    cout << "Average waiting time = "
            << (float)total_wt / (float)n;
    cout << "\nAverage turn around time = "
            << (float)total_tat / (float)n;
}

// Driver code
int main()
{
    // process id's
    int processes[] = { 1, 2, 3};
    int n = sizeof processes / sizeof processes[0];

    // Burst time of all processes
    int burst_time[] = {10, 5, 8};

    // Time quantum
    int quantum = 2;
    findavgTime(processes, n, burst_time, quantum);
    return 0;
}
```

Question no. 15

Ques. 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.

```
#include<stdio.h>
#include<conio.h>
void main()
{
  char p[10][5],temp[5];
  int i,j,pt[10],wt[10],totwt=0,pr[10],temp1,n;
  float avgwt;
 printf("enter no ofprocesses:");
 scanf("%d",&n);
 for(i=0;i<n;i++)
  {
  printf("enter process%d name:",i+1);
 scanf("%s",&p[i]);
  printf("enter process time:");
 scanf("%d",&pt[i]);
 printf("enter priority:");
 scanf("%d",&pr[i]);
  }
 for(i=0;i<n-1;i++)
 {
 for(j=i+1;j<n;j++)
 {
   if(pr[i]>pr[j])
 {
   temp1=pr[i];
 pr[i]=pr[j];
 pr[j]=temp1;
 temp1=pt[i];
 pt[i]=pt[j];
 pt[j]=temp1;
 strcpy(temp,p[i]);
 strcpy(p[i],p[j]);
  strcpy(p[j],temp);
  }
  }
  }
 wt[0]=0;
  for(i=1;i<n;i++)
  {
   wt[i]=wt[i-1]+et[i-1];
   totwt=totwt+wt[i];
   }
avgwt=(float)totwt/n;
printf("p_name\t p_time\t priority\t w_time\n");
for(i=0;i<n;i++)
{
   printf(" %s\t %d\t %d\t %d\n" ,p[i],pt[i],pr[i],wt[i]);
   }
  printf("total waiting time=%d\n avg waiting time=%f",tot,avg);
  getch();
   }
```

OUTPUT:

```
enter no of processes: 5
enter process1 name:aaa
enter process time: 4
enterpriority:5
enter process2 name:bbb
enter process time: 3
enterpriority:4
enter process3 name:ccc
enter process time: 2
enterpriority:3
enter process4 name:ddd
enter process time: 5
enterpriority:2
enter process5 name:eee
enter process time: 1
enterpriority:1
p_name P_time priorityw_time
eee 1 10
ddd 5 21
ccc 2 36
bbb 3 48
aaa 4 5 11
total waiting time=26
avg waiting time=5.20
```