Lab 4: Write a C program to simulate a multi-level queue scheduling algorithm considering the following scenario. All the processes in the system are divided into two categories – system processes and user processes. System processes are to be given higher priority than user processes. Use FCFS scheduling for the processes in each queue.

12/7/23

Q3) Write a C program to ~~sh~~ simulate multi-level queue scheduling algorithm considering the following ~~sce~~ scenario. All the processes in the system are divided into 2 categories- System processes & user processes. System processes are to be given higher priority than user processes. Use FCFS scheduling for the processes in each queue.

```
#include<stdio.h>

#define Max 50

typedef struct
{
  int number;
  int p_id[MAX];
  int tat[MAX];
  int wt[MAX];
  int arrival_time[MAX];
  int cpu_time[MAX];
} Process;

void main()
{
  int n, i, j, pname[MAX], total_time=0, time=0;
  int totaltat[MAX], totalwt[MAX];
  float avgtat=0, avgwt=0;
  Process sp, up;

  printf("Enter the number of system processes: ");
  scanf("%d", &sp.number);
```

```
printf("Enter the number of user processes: ");
scanf("%d", &up.number);

printf("Enter the Arrival time and the Burst
time for system processes: \n");

for(i=0; i<sp.number; i++)
{
   scanf("%d", &sp.arrival_time[i]);
   scanf("%d", &sp.cpu_time[i]);
   //sp.p_id[i] = 10+i+1;
}

printf("Enter the Arrival time and the Burst
time for user processes: \n");

for(i=0; i<up.number; i++)
{
   scanf("%d", &up.arrival_time[i]);
   scanf("%d", &up.cpu_time[i]);

   //up.p_id[i] = 20+i+1;
}

for(i=0; i<sp.number; i++)
      total_time += sp.cpu_time[i];

for(i=0; i<up.number; i++)
      total_time += up.cpu_time[i];

i=0, j=0;
```

```
while (time < total_time)
{
  if (sp.arrival_time[i] <= up.arrival_time[j])
  {
     time += sp.cpu_time[i];
     sp.tat[i] = time - sp.arrival_time[i];
     sp.wt[i] = sp.tat[i] - sp.cpu_time[i];
     i++;
  }
```

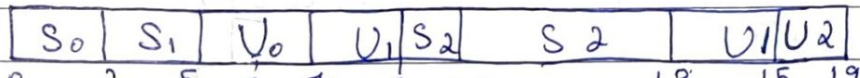
store
67

```
else if (up.arrival_time[j] < sp.arrival_time[i]
      && sp.arrival_time[i] > time)
{
  if ((time + up.cpu_time[j]) > sp.arrival_time[i])
  {
     int ubt = up.cpu_time[j];
     int sbt = sp.cpu_time[i];
     while (time < sp.arrival_time[i])
     {
        time++;
        ubt--;
     }

     time += sbt;

     sp.tat[i] = time - sp.arrival_time[i];
     sp.wt[i] = sp.tat[i] - sp.cpu_time[i];
     i++;
     time += ubt;

     up.tat[j] = time - up.arrival_time[j];
     up.wt[j] = up.tat[j] - up.cpu_time[j];

     j++;

  }
```

```
else
{
    time += up.cpu_time[j];
    up.tat[j] = time - up.arrival_time[j];
    up.wt[j] = up.tat[j] - up.cpu_time[j];
    j++
}
}

else if (up.arrival_time[j] <= sp.arrival_time
        [i] && sp.arrival_time[i] < time)
{
    time += sp.cpu_time[i];
    sp.tat[i] = time - sp.arrival_time[i];
    sp.wt[i] = sp.tat[i] - sp.cpu_time[i];
    i++;
}

}

printf("\t Process \t Arrival time \t
        Burst time \t Waiting Time
        \t Turnaround time \n");

// Now add tat and wt of all user &
// system process to find average.
```

```
for (i=0; i < sp.number ; i++)
{
  print ("\n\t S°/od \t\t °/od \t\t °/od
  \t \t °/od \t\t °/od", i, sp.arrival_time[i],
  sp.cpu_time[i], sp.wt[i], sp.tat[i]);
  avgtat += sp.tat[i];
  avgwt += sp.wt[i];
}

for (i=0; i < up.number ; i++)
{
  printf ("\n\t U°/od \t\t °/od \t\t °/od
  \t\t °/od \t\t °/od", i, &up.arrival_time[i],
  up.cpu_time[i], up.wt[i], up.tat[i]);
  avgtat += up.tat[i];
  avg wt += up.wt[i];
}

avgtat /= (sp.number + up.number);
avgwt /= (sp.numbers + up.number);
printf ("\n Average Turnaround Time = °/of",
printf ("\n Average Waiting time =°/of", avgwt); avgtat);
}
```

Gantt chart



S_0	S_1	V_0	V_1	S_2	V_1	V_2
V_0	V_0	V_0		S_2	V_2	
V_1	V_1	V_1		V_2		
		V_2				

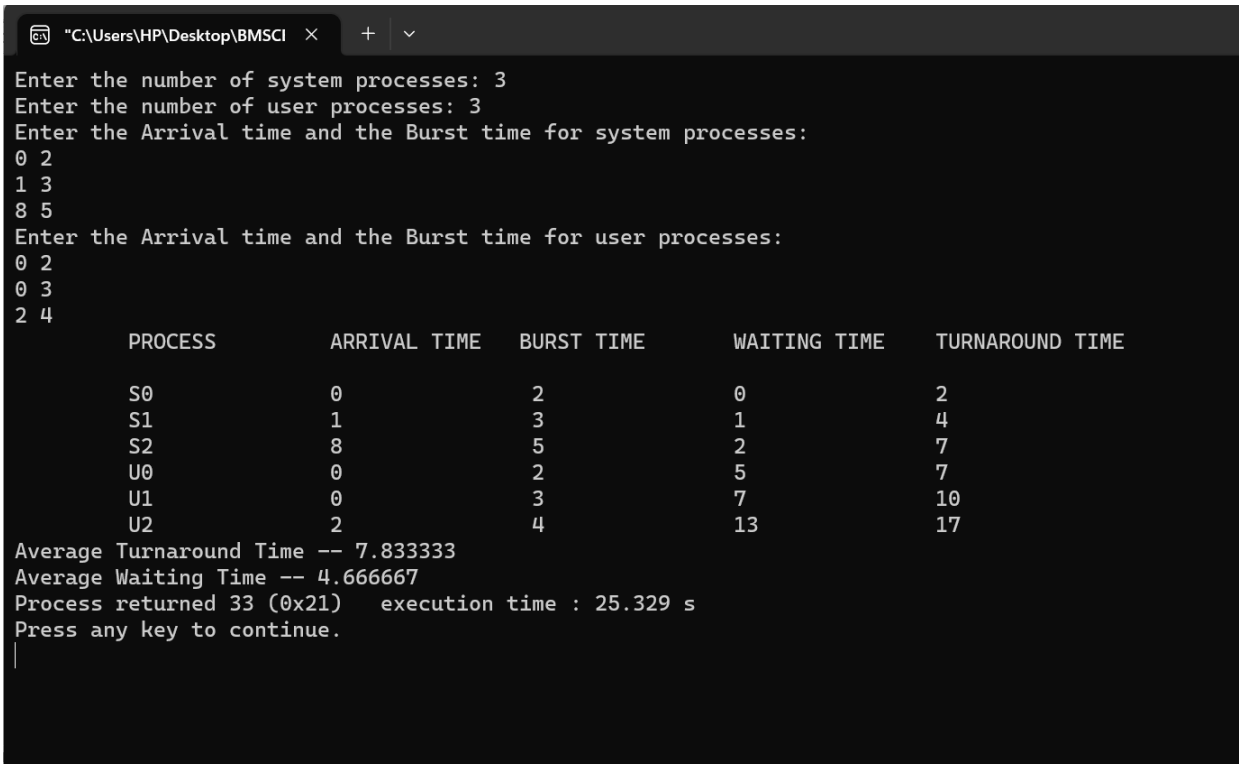
Output :-

Enter the number of system processes : 3
Enter the number of user processes : 3
Enter the Arrival time and the Burst time for system processes :
0 2
1 3
8 5
.
Enter the Arrival time and the Burst time for user processes :
0 2
0 3
2 4

Process	Arrival Time	Burst Time	WT	TAT
S0	0	2	0	2
S1	1	3	1	4
S2	8	5	0	5
U0	0	2	5	7
U1	0	3	12	15
U2	2	4	13	17

Average Turnaround Time = 8.333333.
Average Waiting Time = 5.166667

OUTPUT :


"C:\Users\HP\Desktop\BMSCI
Enter the number of system processes: 3
Enter the number of user processes: 3
Enter the Arrival time and the Burst time for system processes:
0 2
1 3
8 5
Enter the Arrival time and the Burst time for user processes:
0 2
0 3
2 4
PROCESS ARRIVAL TIME BURST TIME WAITING TIME TURNAROUND TIME
S0 0 2 0 2
S1 1 3 1 4
S2 8 5 2 7
U0 0 2 5 7
U1 0 3 7 10
U2 2 4 13 17
Average Turnaround Time -- 7.833333
Average Waiting Time -- 4.666667
Process returned 33 (0x21) execution time : 25.329 s
Press any key to continue.

Write a C program to simulate Real-Time CPU Scheduling algorithms:
a) Rate- Monotonic

store
67

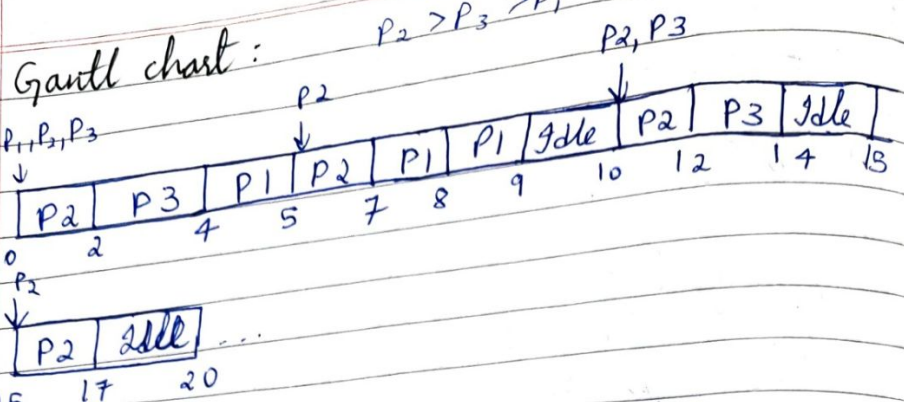
8+) Write a C program to simulate Real-Time CPU Scheduling algorithms:
a) Rate Monotonic
b)
c)

```
#include<stdio.h>
#define MAX
int
void
void
*

typedef struct
{
  int p_id;
  int period;
  int execution_time;
  int deadline;
} Process;

float calculate_cpu_utilization(Process tasks[],
                                int n)
{
   float total_utilization = 0.0;

   for(int i=0; i<n; i++)
   {
     float task_utilization = (float) tasks[i].
       execution_time / tasks[i].period;
   } total_utilization += task_utilization;
}
```

```
    float cpu_utilization = total_utilization * 100;
    return cpu_utilization;
}

void rate_monotonic ()
{
  int n, i;

  printf ("Enter the number of processes: ");
  scanf ("%d", &n);

  Process tasks [MAX];

  for (i=0; i<n; i++)
  {
   printf ("Process %d \n", i+1);
   printf ("Enter period : ");
   scanf ("%d", &tasks[i].period);
   printf ("Enter execution Time : ");
   scanf ("%d", &task[i].execution_time);
   printf("Enter deadline : ");
   scanf ("%d", &tasks [i].deadline );

   tasks[i].p_id = i+1;
}

float cpu_utilization = calculate_cpu_utilization
                        (tasks, n);
printf ("CPU Utilization : %0.2f %% \n",
         cpu_utilization);
}
```

```
int main()
{
  int i, ch;

  while(1)
  {
    printf("Menu \n");
    printf("1. Rate Monotonic \n 2. Earliest Dead-
    -line first \n 3. Proportional scheduling \n
    4. Exit");
    scanf("%d", &ch);

    switch(ch)
    {
      case 1: rate_monotonic();
              break;
      case 2:
              break;
      case 3:
              break;

      case 4: exit(0);

      default: printf("\n Wron Choice ! Try again");
    }
  }
  return 0;
}
```

Gantt chart: $P_2 > P_3 > P_1$



Output:

Menu

1.

2.

3.

4. Exit

1

Enter the numbe of processes: 3

Process 1

Period: 20 Execution time: 3 Deadline: 20

Process 2

Period: 5 Execution time: 2 Deadline: 5

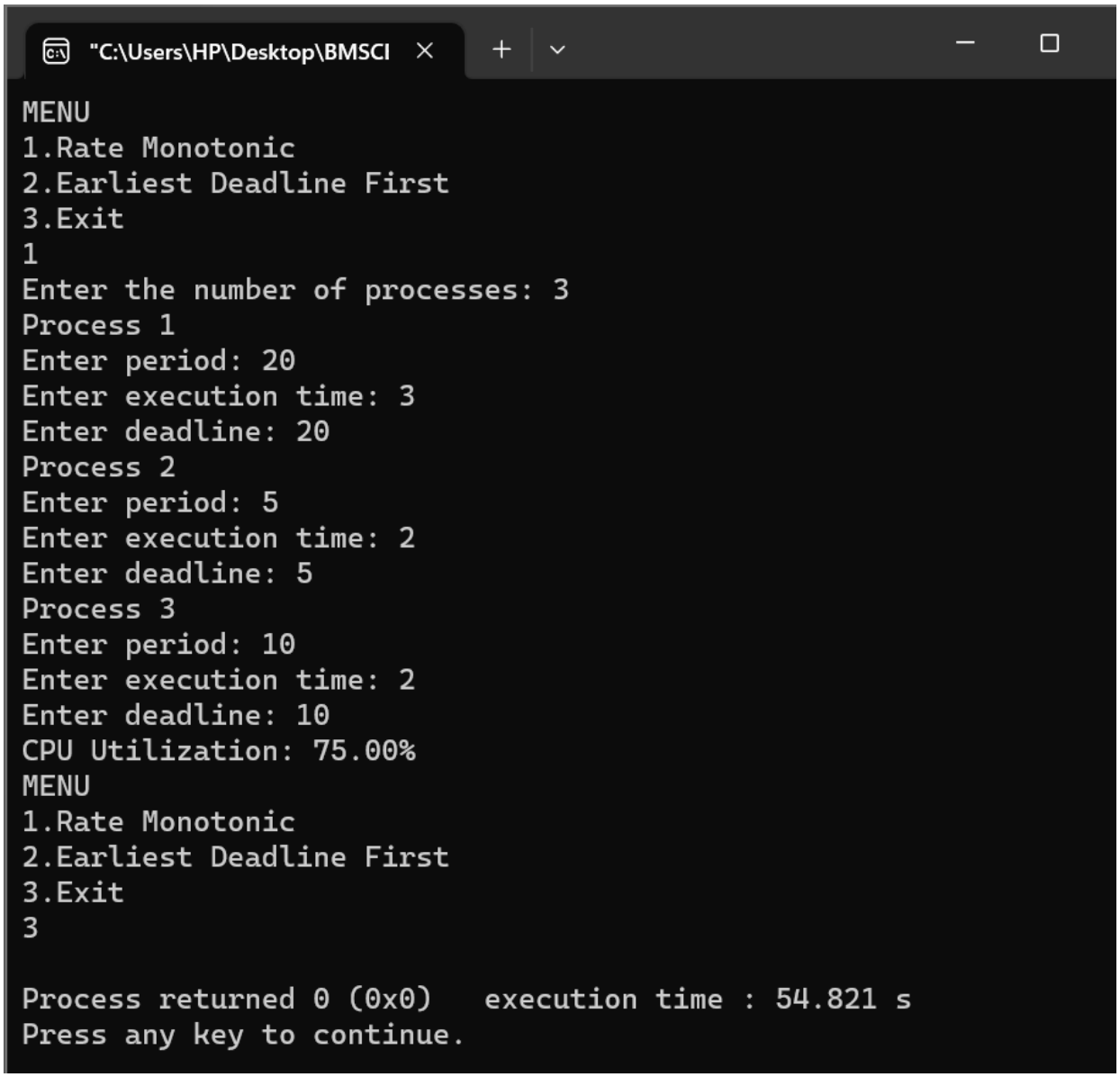
Process 3

Period: 10 Execution time: 2 Deadline: 10

CPU utilization: 75%

3/4

OUTPUT :


"C:\Users\HP\Desktop\BMSCI
MENU
1.Rate Monotonic
2.Earliest Deadline First
3.Exit
1
Enter the number of processes: 3
Process 1
Enter period: 20
Enter execution time: 3
Enter deadline: 20
Process 2
Enter period: 5
Enter execution time: 2
Enter deadline: 5
Process 3
Enter period: 10
Enter execution time: 2
Enter deadline: 10
CPU Utilization: 75.00%
MENU
1.Rate Monotonic
2.Earliest Deadline First
3.Exit
3
Process returned 0 (0x0) execution time : 54.821 s
Press any key to continue.