Write a c program to simulatereal time CPU scheduling algorithms.a)Multi level scheduling b)rate monotonic.

LAB 4.

Write a c program to simulate multilevel scheduling algorithm considering the following scenario. All the process in the system are divided into 2 categories - system process and user process. system process have higher priority Use FCFS scheduling for process in each queue

```
# include <stdio.h>
# define MAX 20
typedef struct
{
  int number;
  int b-id[MAX];
  int bat[MAX];
  int bbt[MAX];
  int arrival-time[MAX];
  int cpu-time[MAX];
} process;

void main()
{ int n,i,id, puame[MAX], total-time=0;
  int total;
  float avgtat=0; avgwt=0;
  process sp, bb;
  printf("Enter the no: of system process");
  scanf("%d", &sp.number);
  printf("Enter arrival time & burst time for
          system process\n");
  for(i=0; i<sp.number; i++)
  {
    scanf("%d", sp.arrival_time[i]);
    scanf("%d", sp.cpu-time[i]);
    sp.b-id[i] = 10+i+1;
  }
```

```
printf ("Enter the no. of user process.");
scanf ("%d", &up.number);
printf ("Enter the arrival time and burst
        time for user process\n");
for (i=0; i<up.number; i++)
{ scanf ("%d", &up.arrival_time[i]);
  scanf ("%d", &up.cpu_time[i]);
  up.pid[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=20; j=20;
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++;
}
else if (up.arrival_time[j] < sp.arrival_time[i]
  && sp.arrival_time[i] > time)
{ time += up.cpu_time[i]);
  int abt = up.cpu_time[j];
  sbt = sp.cpu_time[i];
while (time < sp.arrival_time[i])
{ time++;
  up--;
}
```

```
        time += sbt[j];
        sp.tat[i] = time - sp.arrival_time[i];
        sp.wt[i] = sp.tat[i] - sb.cpu_time[i];
        i++;
        time += upt[j];
        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
        && sp.arrival_time < 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("\n Process\t Arrival time\t Burst time
    \t Waiting time\t Turnaround time\n");
for(i=0; i<sp.number; i++)
{
printf("\n%d\t\t%d\t\t%d\t\t%d\t\t%d",
    i, sp.arrival_time[i], up.cpu_time[i],
    sp.wt[i], up.tat[i]);
    avg_tat += sp.tat[i];
    avg_wt += sp.wt[i];
}
```

```
for(i=0
& printf
    i+1
    up.cp
    a

}
avg_tat
avg_wt
printf
printf
}
Output
Enter
Enter th
Process
0  2
1  3
&  5
Enter
Enter
Time
0
```

```
for (i=0; i<up.number; i++)
{ printf ("\n %d\t %d\t\t %d\t\t %d\t\t %d",
    i+1, up.arrival_time[i], up.burst_time[i],
    up.wt[i], up.tat[i]);
  avgtat += up.tat[i];
  avgwt += up.wt[i];
}
avgtat /= (sp.number + up.number);
avgwt /= (sp.number + up.number);
printf ("\n Average Turnaround time -- %f",
                avgtat);
printf ("\n Average waiting time -- %f",
                avgwt);
}
```

Output:

Enter the number of system process: 3
Enter the arrival time & burst time for system processes.

0 2
1 3
8 5

Enter the number of user process: 3
Enter the arrival time and the burst time for user processes:

0 2
0 3
2 4

Process	Arrival time	Burst time	Waiting t	Turnaround
S0	0	2	0	2
S1	1	3	1	4
S2	8	5	0	5
V0	0	2	5	7
V1	0	3	12	15
V_2	2	4	13	17

Average turnaround time -- 8.3333

Average waiting time -- 5.16667

Gantt chart

S0	S1	V0	V1	S2	V1	V_2

0 2 5 7 8 13 15 19

(a) write a c program to stimulate real-time CPU scheduling algorithm rate-monotonic

```
#include <stdio.h>
#define MAX_TASKS 100
typedef struct
{
    int pid;
    int period;
    int exec_time;
    int deadline;
} Task;
float cpu_utilization (Task tasks[], int n)
{
    float total_util = 0.0;
    for (int i=0; i<n; i++)
    {
        float task_util = (float) tasks[i].
                                execution_time /
                                tasks[i].period;
        total_util += task_util;
    }
    float cpu_util = total_util * 100;
    return cpu_util;
}
void ratemonotonic()
{
    int n, i;
    printf ("Enter the number of tasks:");
    scanf ("%d", &n);
    Task tasks[MAX_TASKS];
    for (i=0; i<n; i++)
    {
        printf ("Task %d \n", i+1);
        printf ("Enter period:");
        scanf ("%d", &tasks[i].period);
        printf ("Enter execution time:");
        scanf ("%d", &tasks[i].exec_time);
```

```
printf ("Enter deadline: ");
scanf ("%d", &Tasks[i].deadline);
    Tasks[i].pid = i+1;
}
float cpu_util = cpu_utilization (Tasks, n);
printf ("CPU utilization = %f %%\n",
                cpu_util);
}
void main()
{ int choice, i, n;
  printf ("1. Rate monotonic \n 2. exit \n\n")
    while (1)
  { scanf ("%d", &choice);
    switch (choice)
  case 1: Rate Monotonic();
          break;
  case 2 :- exit(0);
    }
  }
}
```

Output:

1) Rate Monotonic
2) Exit

1) Enter the number of tasks : 3

Task 1

Enter period : 20

Enter execution_time : 3

Enter deadline : 20

Task 2

Enter period : 5

Enter de

Task 3

Enter p

Enter e

Enter

CPU u

Grant

P2	P3
0	2

Task 3

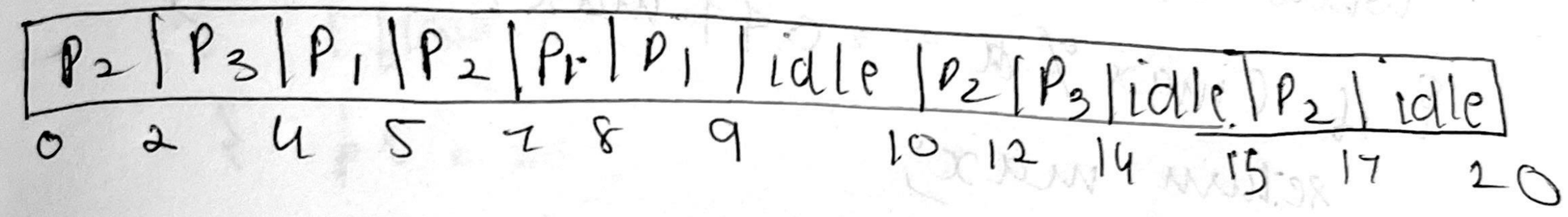
Enter period : 10

Enter execution time : 2

Enter deadline : 10

CPU utilization : 75.000%

Gantt chart



```
2
3
5
Enter the arrival times for User Processes:
0
0
2
Enter the burst times for User Processes:
2
3
4

0 SP1 2 SP2 5 UP1 7 UP2 10 SP3 15 UP3 19
System Processes:
SP1 2 0
SP2 4 0
SP3 7 0
Average Turnaround Time (System Processes): 4.33
Average Waiting Time (System Processes): 0

User Processes:
UP1 7 5
UP2 10 7
UP3 17 13
Average Turnaround Time (User Processes): 11.33
Average Waiting Time (User Processes): 8.33

Process returned 0 (0x0)   execution time : 69.771 s
Press any key to continue.
```

```
Enter the number of processes: 3
Enter execution times:
3
2
2
Enter deadlines:
20
5
10
0 P2 0 P3 2 P1 4 P2 5 P1 7 9 Idle P2 10 P3 12 14 Idle P2 15 17 Idle P2 20
```