OS LAB

VIBHA HUGAR CS 255

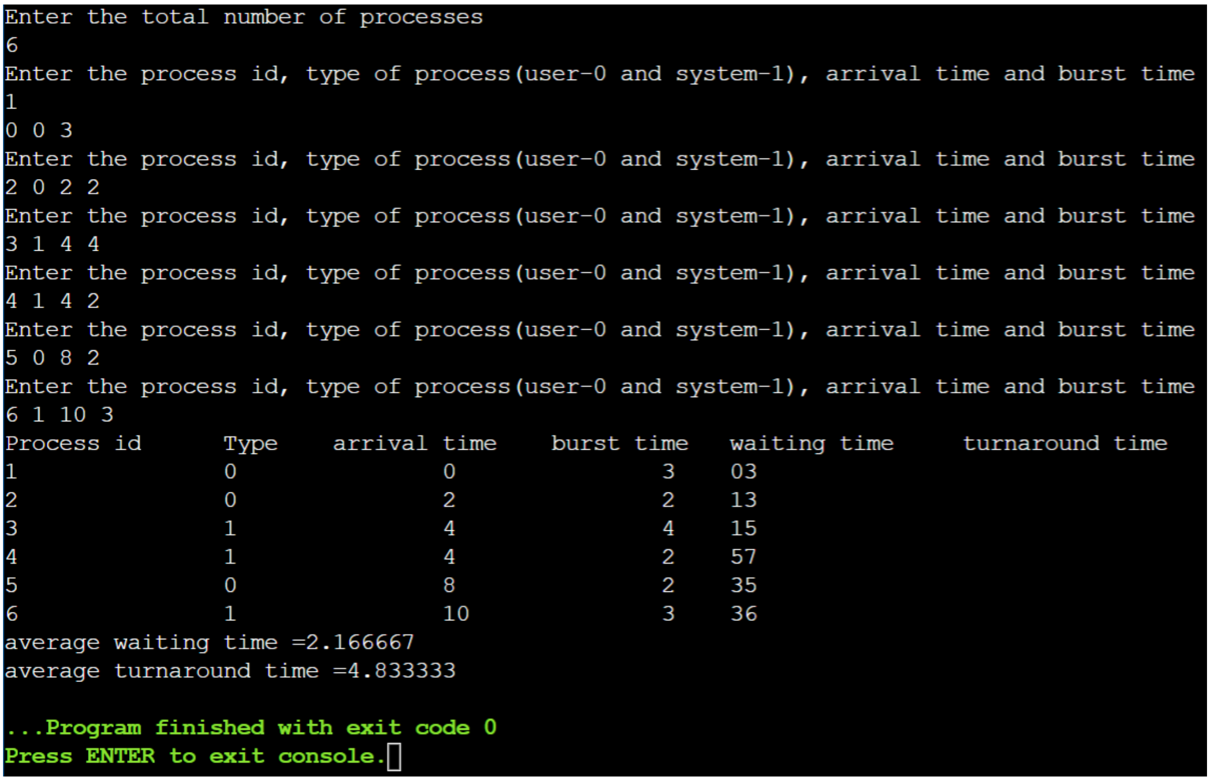
MULTILEVEL QUEUE CODE

```
#include<stdio.h>
void swap(int *a,int *b)
{
    int temp;
    temp=*a;
    *a=*b;
    *b=temp;
}
void main()
{
    int n,pid[10],burst[10],type[10],arr[10],wt[10],ta[10],ct[10],i,j;
    float avgwt=0,avgta=0;
    int sum = 0;
    printf("Enter the total number of processes\n");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("Enter the process id, type of process(user-0 and system-1), arrival time and burst
time\n");
        scanf("%d",&pid[i]);
        scanf("%d",&type[i]);
        scanf("%d",&arr[i]);
        scanf("%d",&burst[i]);
    }
    //sorting the processes according to arrival time
    for(i=0;i<n-1;i++)
```

```
{
   for(j=0;j<n-i-1;j++)
   {
      if(arr[j]>arr[j+1])
      {
         swap(&arr[j],&arr[j+1]);
         swap(&pid[j],&pid[j+1]);
         swap(&burst[j],&burst[j+1]);
         swap(&type[j],&type[j+1]);

      }
   }
}
//assuming only two process can have same arrival time and different priority
for(i=0;i<n-1;i++)
{
   for(j=0;j<n-i-1;j++)
   {
      if(arr[j]==arr[j+1] && type[j]<type[j+1])
      {
         swap(&arr[j],&arr[j+1]);
         swap(&pid[j],&pid[j+1]);
         swap(&burst[j],&burst[j+1]);
         swap(&type[j],&type[j+1]);
      }
   }
}
//calculating completion time, arrival time and waiting time
sum = sum + arr[0];
for(i = 0;i<n;i++){
   sum = sum + burst[i];
```

```
        ct[i] = sum;
        ta[i] = ct[i] - arr[i];
        wt[i] = ta[i] - burst[i];
        if(sum<arr[i+1]){
            int t = arr[i+1]-sum;
            sum = sum+t;
        }
    }

    printf("Process id\tType\tarrival time\tburst time\twaiting time\tturnaround time\n");
    for(i=0;i<n;i++)
    {
        avgta+=ta[i];
        avgwt+=wt[i];
        printf("%d\t\t%d\t\t%d\t\t%d\t\t%d\t\t%d\n",pid[i],type[i],arr[i],burst[i],wt[i],ta[i]);
    }
    printf("average waiting time =%f\n",avgwt/n);
    printf("average turnaround time =%f",avgta/n);

}
```

OUTPUT



```
Enter the total number of processes
6
Enter the process id, type of process(user-0 and system-1), arrival time and burst time
1
0 0 3
Enter the process id, type of process(user-0 and system-1), arrival time and burst time
2 0 2 2
Enter the process id, type of process(user-0 and system-1), arrival time and burst time
3 1 4 4
Enter the process id, type of process(user-0 and system-1), arrival time and burst time
4 1 4 2
Enter the process id, type of process(user-0 and system-1), arrival time and burst time
5 0 8 2
Enter the process id, type of process(user-0 and system-1), arrival time and burst time
6 1 10 3
Process id      Type    arrival time    burst time   waiting time     turnaround time
1               0               0               3    03
2               0               2               2    13
3               1               4               4    15
4               1               4               2    57
5               0               8               2    35
6               1               10              3    36
average waiting time =2.166667
average turnaround time =4.833333

...Program finished with exit code 0
Press ENTER to exit console.
```

RATE MONOTONIC SCHEDULING CODE

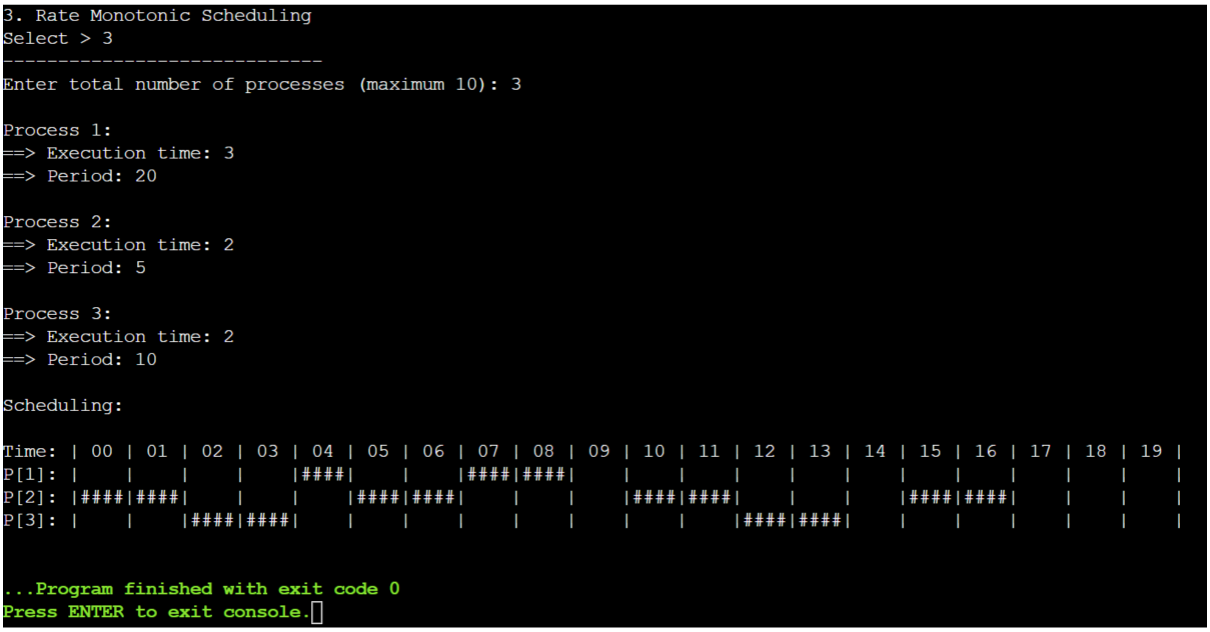
```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdbool.h>
#define MAX_PROCESS 10
int num_of_process = 3, count, remain, time_quantum;
int execution_time[MAX_PROCESS], period[MAX_PROCESS],
   remain_time[MAX_PROCESS], deadline[MAX_PROCESS],
   remain_deadline[MAX_PROCESS];
int burst_time[MAX_PROCESS], wait_time[MAX_PROCESS],
   completion_time[MAX_PROCESS], arrival_time[MAX_PROCESS];
// collecting details of processes
void get_process_info(int selected_algo)
{
   printf("Enter total number of processes (maximum %d): ",
         MAX_PROCESS);
   scanf("%d", &num_of_process);
   if (num_of_process < 1)
   {
      printf("Do you really want to schedule %d processes?",
            num_of_process);
      exit(0);
   }
   for (int i = 0; i < num_of_process; i++)
   {
      printf("\nProcess %d:\n", i + 1);
      printf("==> Execution time: ");
      scanf("%d", &execution_time[i]);
      remain_time[i] = execution_time[i];
```

```
        printf("==> Period: ");
        scanf("%d", &period[i]);
    }
}
// get maximum of three numbers
int max(int a, int b, int c)
{
    int max;
    if (a >= b && a >= c)
        max = a;
    else if (b >= a && b >= c)
        max = b;
    else if (c >= a && c >= b)
        max = c;
    return max;
}
// calculating the observation time for scheduling timeline
int get_observation_time(int selected_algo)
{

    return max(period[0], period[1], period[2]);
}
// print scheduling sequence
void print_schedule(int process_list[], int cycles)
{
    printf("\nScheduling:\n\n");
    printf("Time: ");
    for (int i = 0; i < cycles; i++)
    {
        if (i < 10)
```

```
            printf("| 0%d ", i);
        else
            printf("| %d ", i);
    }
    printf("|\n");
    for (int i = 0; i < num_of_process; i++)
    {
        printf("P[%d]: ", i + 1);
        for (int j = 0; j < cycles; j++)
        {
            if (process_list[j] == i + 1)
                printf("|####");
            else
                printf("|    ");
        }
        printf("|\n");
    }
}
void rate_monotonic(int time)
{
    int process_list[100] = {0}, min = 999, next_process = 0;
    float utilization = 0;
    for (int i = 0; i < num_of_process; i++)
    {
        utilization += (1.0 * execution_time[i]) / period[i];
    }
    int n = num_of_process;
    if (utilization > n * (pow(2, 1.0 / n) - 1))
    {
        printf("\nGiven problem is not schedulable under the said scheduling algorithm.\n");
        exit(0);
    }
    for (int i = 0; i < time; i++)
    {
        min = 1000;
        for (int j = 0; j < num_of_process; j++)
        {
            if (remain_time[j] > 0)
            {
                if (min > period[j])
                {
                    min = period[j];
                    next_process = j;
                }
            }
        }
        if (remain_time[next_process] > 0)
        {
            process_list[i] = next_process + 1; // +1 for catering 0 array index.
            remain_time[next_process] -= 1;
        }
        for (int k = 0; k < num_of_process; k++)
        {
            if ((i + 1) % period[k] == 0)
            {
                remain_time[k] = execution_time[k];
                next_process = k;
            }
        }
    }
    print_schedule(process_list, time);
```

```
}
int main(int argc, char *argv[])
{
    int option = 0;
    printf("3. Rate Monotonic Scheduling\n");
    printf("Select > ");
    scanf("%d", &option);
    printf("-----------------------------\n");
    get_process_info(option); // collecting processes detail
    int observation_time = get_observation_time(option);
    if (option == 3)
        rate_monotonic(observation_time);
    return 0;
}
```

OUTPUT



```
3. Rate Monotonic Scheduling
Select > 3
-----------------------------
Enter total number of processes (maximum 10): 3

Process 1:
==> Execution time: 3
==> Period: 20

Process 2:
==> Execution time: 2
==> Period: 5

Process 3:
==> Execution time: 2
==> Period: 10

Scheduling:

Time: | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 |
P[1]: |    |    |    |    |####|    |    |####|####|    |    |    |    |    |    |    |    |    |    |    |
P[2]: |####|####|    |    |    |####|####|    |    |    |####|####|    |    |    |####|####|    |    |    |
P[3]: |    |    |####|####|    |    |    |    |    |    |    |    |####|####|    |    |    |    |    |    |


...Program finished with exit code 0
Press ENTER to exit console.
```

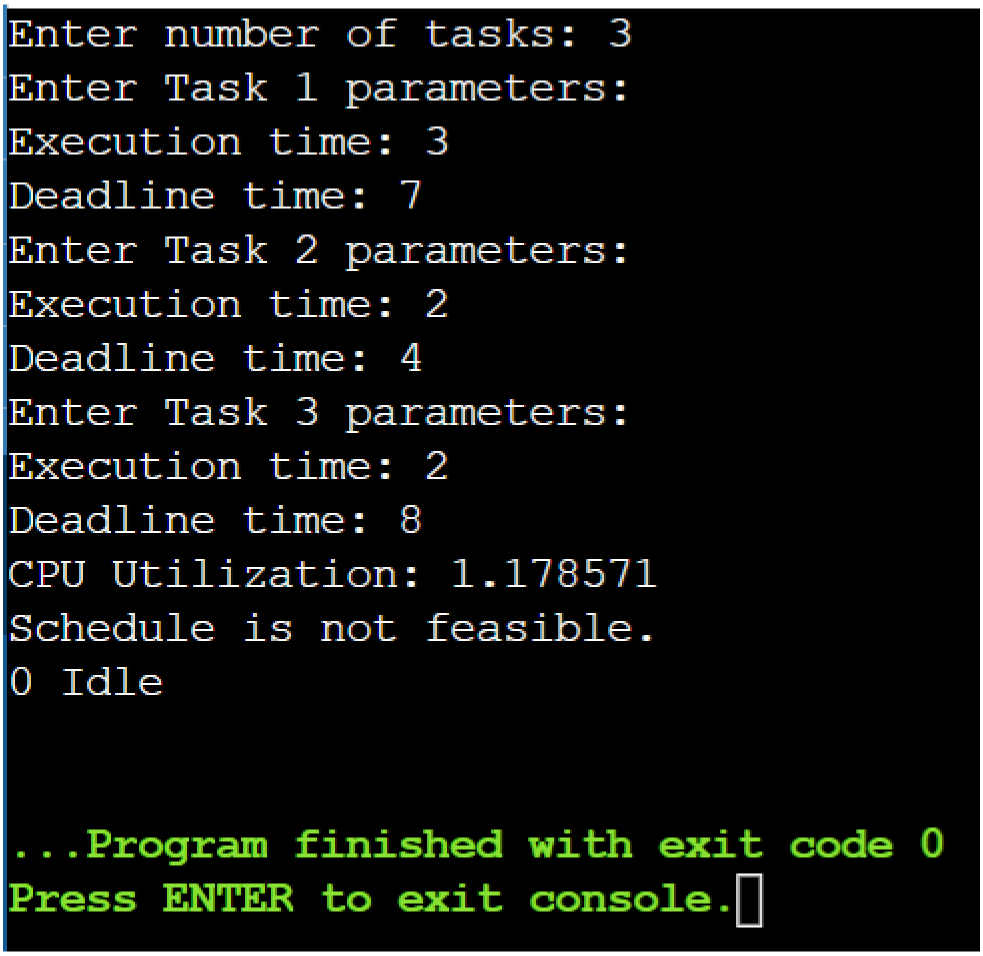
EDF CODE

```
#include <stdio.h>
#include <stdlib.h>

typedef struct {
    int deadline;
    int execution;
    int execution_copy;
} Task;

int min(Task *tasks, int n);
void update_execution_copy(Task *tasks, int n);
void execute_task(Task *tasks, int task_id, int timer);

int main() {
    int n, timer = 0;
    float cpu_utilization;

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

    Task *tasks = malloc(n * sizeof(Task));

    // Input task parameters
    for (int i = 0; i < n; i++) {
        printf("Enter Task %d parameters:\n", i + 1);
        printf("Execution time: ");
        scanf("%d", &tasks[i].execution);
        printf("Deadline time: ");
        scanf("%d", &tasks[i].deadline);
        tasks[i].execution_copy = 0;
    }

    // Calculate CPU utilization
    cpu_utilization = 0;
    for (int i = 0; i < n; i++) {
        cpu_utilization += (float)tasks[i].execution / (float)tasks[i].deadline;
    }

    printf("CPU Utilization: %f\n", cpu_utilization);

    if (cpu_utilization < 1)
        printf("Tasks can be scheduled.\n");
    else
        printf("Schedule is not feasible.\n");

    while (1) {
        int active_task_id = min(tasks, n);

        if (active_task_id == -1) {
            printf("%d Idle\n", timer);
        } else {
            execute_task(tasks, active_task_id, timer);
            if (tasks[active_task_id].execution_copy == 0) {
                update_execution_copy(tasks, active_task_id);
            }
        }

        timer++;
```

```
        // Exit condition: All tasks have completed execution
        int all_completed = 1;
        for (int i = 0; i < n; i++) {
            if (tasks[i].execution_copy > 0) {
                all_completed = 0;
                break;
            }
        }
        if (all_completed) {
            break;
        }
    }

    free(tasks);
    return 0;
}

int min(Task *tasks, int n) {
    int min_deadline = __INT_MAX__;
    int task_id = -1;

    for (int i = 0; i < n; i++) {
        if (tasks[i].execution_copy > 0 && tasks[i].deadline < min_deadline) {
            min_deadline = tasks[i].deadline;
            task_id = i;
        }
    }

    return task_id;
}
```

```
void update_execution_copy(Task *tasks, int n) {
   tasks[n].execution_copy = tasks[n].execution;
}


void execute_task(Task *tasks, int task_id, int timer) {
   tasks[task_id].execution_copy--;
   printf("%d Task %d\n", timer, task_id + 1);
}
```

OUTPUT



```
Enter number of tasks: 3
Enter Task 1 parameters:
Execution time: 3
Deadline time: 7
Enter Task 2 parameters:
Execution time: 2
Deadline time: 4
Enter Task 3 parameters:
Execution time: 2
Deadline time: 8
CPU Utilization: 1.178571
Schedule is not feasible.
0 Idle


...Program finished with exit code 0
Press ENTER to exit console.
```