Master 2 - Control & Robotics - Embedded Real Time Systems
Real Time Scheduling Course
14/11/2020

Rate Monotonic Scheduler

Viktoria BILIOURI

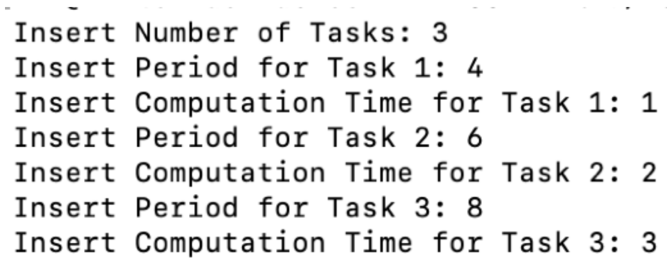
Synopsis

The purpose of this project is to simulate the behaviour of a Preemptive Fixed Priority-Driven Scheduler using the scheduling algorithm of Rate Monotonic. The scheduler aims to be used in sets of periodic tasks, which deadline we assume to be equal to their period, and all the tasks are released simultaneously at time 0. Our project is divided in the following parts: 1. Data acquisition, 2. Schedulability analysis, 3. Simulation, and 4. Display of main metrics.

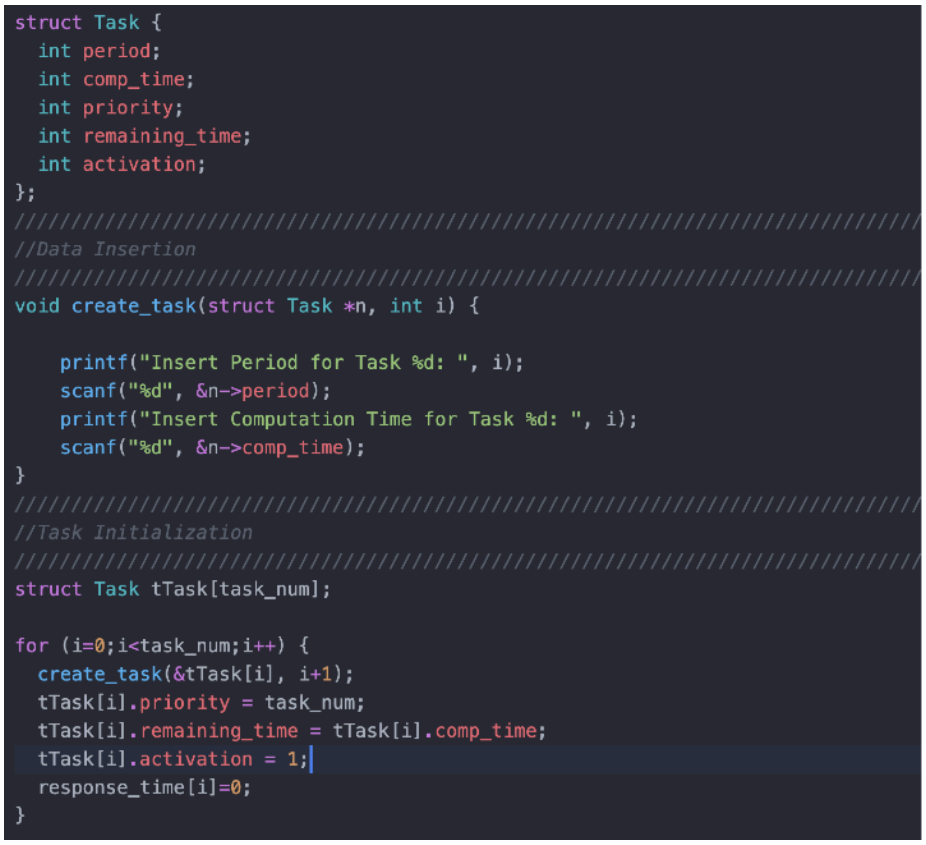
The Rate Monotonic algorithm aims at the scheduling of tasks, depending on their priority, which is given in accordance with the tasks' period. In other words, higher priority is given in the task with the lowest period.

Data Acquisition

In the beginning of the execution of the algorithm, the user should define the number of tasks that are going to be scheduled, as well as the period and the worst case computation time of each task. These data are stored and the tasks are created with these characteristics. Each task's details are stored to a Task data structure (struct) and all the tasks are stored in an array of Task structs.
In the following image we can see an execution example of the data insertion by the user:


```
Insert Number of Tasks: 3
Insert Period for Task 1: 4
Insert Computation Time for Task 1: 1
Insert Period for Task 2: 6
Insert Computation Time for Task 2: 2
Insert Period for Task 3: 8
Insert Computation Time for Task 3: 3
```

Afterwards, the data are obtained and stored to the Task struct, and finally all the tasks' information are stored to a task array:



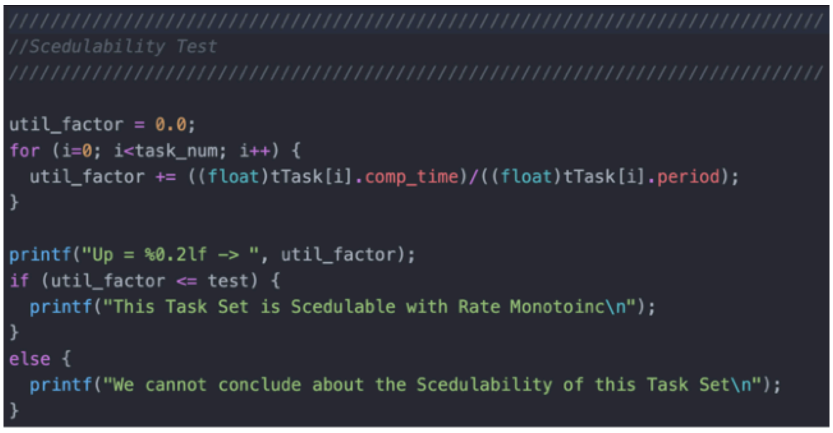
```
struct Task {
  int period;
  int comp_time;
  int priority;
  int remaining_time;
  int activation;
};
////////////////////////////////////////////////////////////////////////////////
//Data Insertion
////////////////////////////////////////////////////////////////////////////////
void create_task(struct Task *n, int i) {

    printf("Insert Period for Task %d: ", i);
    scanf("%d", &n->period);
    printf("Insert Computation Time for Task %d: ", i);
    scanf("%d", &n->comp_time);
}
////////////////////////////////////////////////////////////////////////////////
//Task Initialization
////////////////////////////////////////////////////////////////////////////////
struct Task tTask[task_num];

for (i=0;i<task_num;i++) {
  create_task(&tTask[i], i+1);
  tTask[i].priority = task_num;
  tTask[i].remaining_time = tTask[i].comp_time;
  tTask[i].activation = 1;
  response_time[i]=0;
}
```

Schedulability Analysis

The schedulability test of Rate Monotonic Scheduler is :

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1)$$

So our program calculates the Utilization factor and compares it with the testing value:



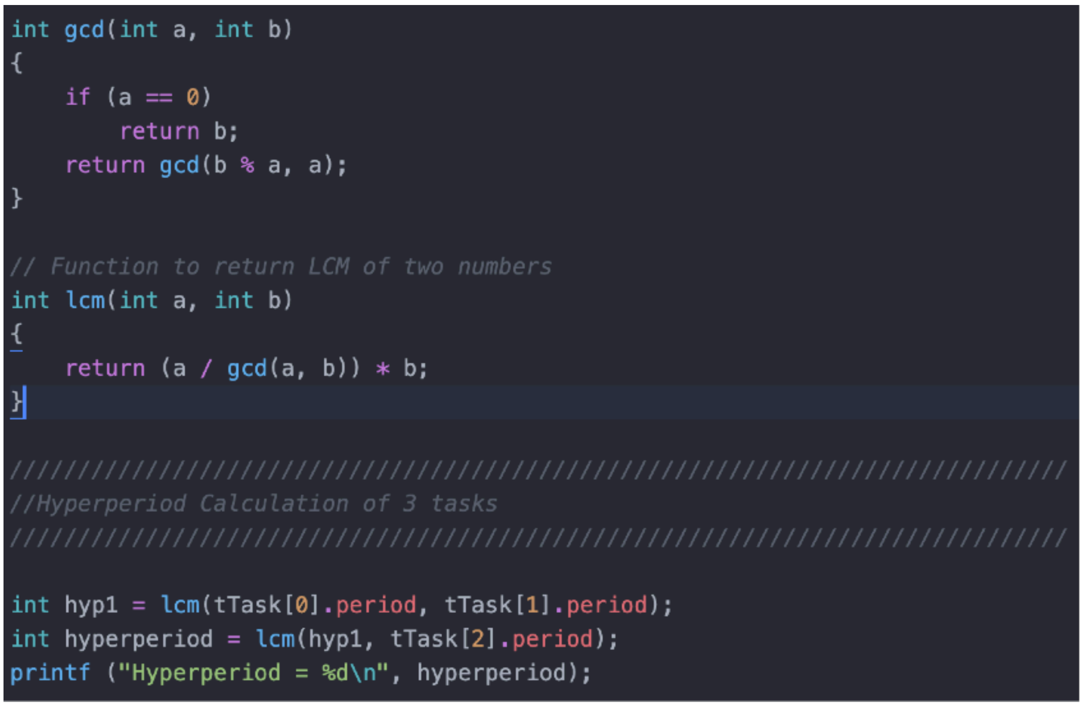
```
////////////////////////////////////////////////////////////////////////////
//Scedulability Test
////////////////////////////////////////////////////////////////////////////

util_factor = 0.0;
for (i=0; i<task_num; i++) {
  util_factor += ((float)tTask[i].comp_time)/((float)tTask[i].period);
}

printf("Up = %0.2lf -> ", util_factor);
if (util_factor <= test) {
  printf("This Task Set is Scedulable with Rate Monotoinc\n");
}
else {
  printf("We cannot conclude about the Scedulability of this Task Set\n");
}
```

If the above condition is satisfied, the task set is definitely schedulable, if not we should draw the simulation of the scheduling of the task set in order to verify its schedulability. However for the purpose of this project, the simulation is done in both cases.

Simulation

In the next step, we need to simulate the scheduling of the task set for one hyperperiod, using Rate Monotonic Scheduler. First, we need to calculate the **hyperperiod (H)** of our Task set, as the hyperperiod is the period of time that defines the complete performance of the scheduler. That is to say, if a periodic task set is schedulable during one hyperperiod, it will be schedulable in every other period of time. The hyperperiod is the Least Common Multiple of the periods of all tasks:

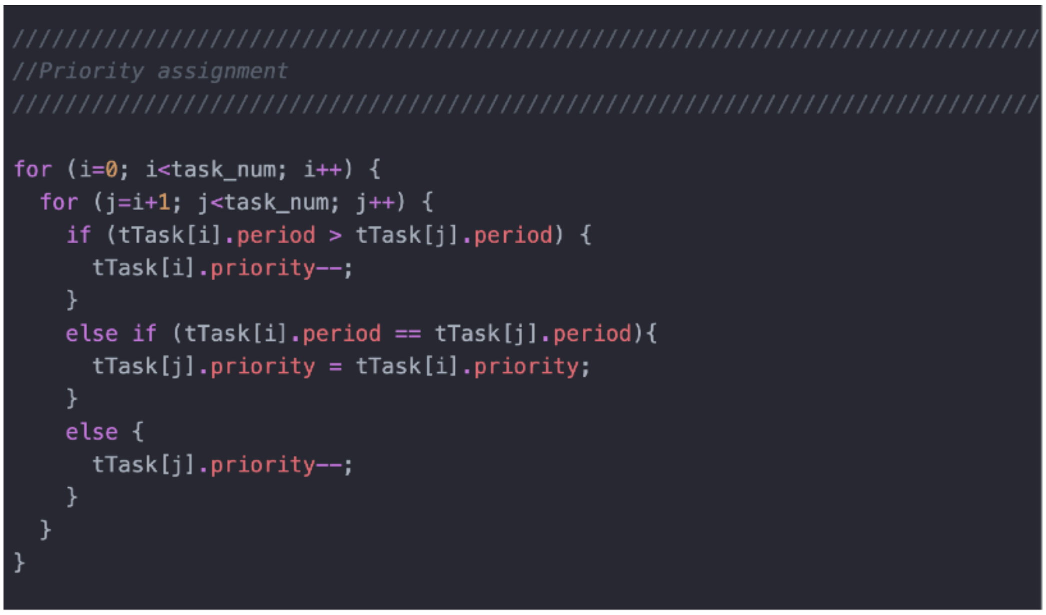
H = LCM(period of Task 1, period of Task 2, …… , period of Task n)



```
int gcd(int a, int b)
{
    if (a == 0)
        return b;
    return gcd(b % a, a);
}

// Function to return LCM of two numbers
int lcm(int a, int b)
{
    return (a / gcd(a, b)) * b;
}

//////////////////////////////////////////////////////////////////////////////
//Hyperperiod Calculation of 3 tasks
//////////////////////////////////////////////////////////////////////////////

int hyp1 = lcm(tTask[0].period, tTask[1].period);
int hyperperiod = lcm(hyp1, tTask[2].period);
printf ("Hyperperiod = %d\n", hyperperiod);
```

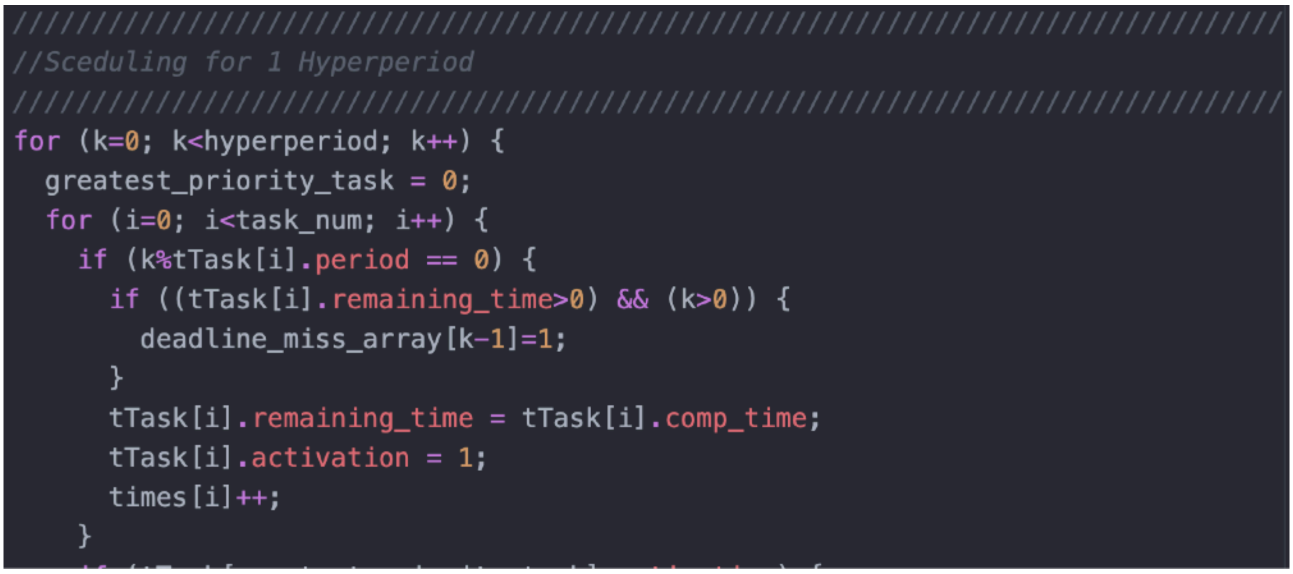
After the calculation of the algorithm, we should assign a priority to every task, using the RM protocol mentioned above. During the initialization of the tasks, their priority is initialized same with the number of tasks that exist in the task set, and every time a task has higher period than an other, its priority is decreased. The search is done using two nested for loops, and each task is compared to every other task of the task set. The Rate Monotonic algorithm is a fixed point algorithm, so the priorities are assigned only in the beginning of the execution, and they remain stable during the whole process.

The next image, indicates the priority assignment process:

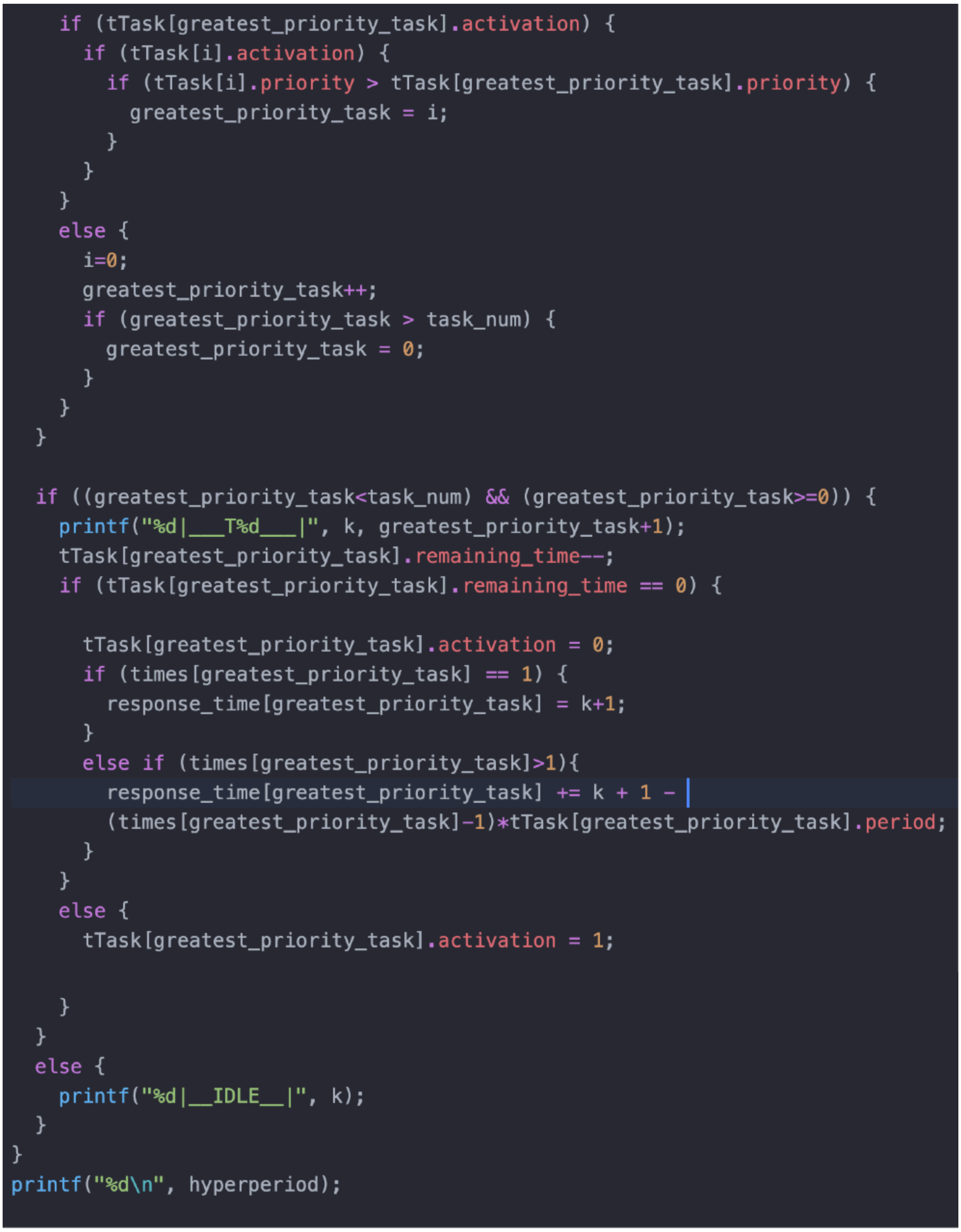


```
///////////////////////////////////////////////////////////////////////////////
//Priority assignment
///////////////////////////////////////////////////////////////////////////////

for (i=0; i<task_num; i++) {
  for (j=i+1; j<task_num; j++) {
    if (tTask[i].period > tTask[j].period) {
      tTask[i].priority--;
    }
    else if (tTask[i].period == tTask[j].period){
      tTask[j].priority = tTask[i].priority;
    }
    else {
      tTask[j].priority--;
    }
  }
}
```

Then, we continue with the Simulation algorithm. The RM algorithm, is constructed using two nested for loops, the first one is counting the units of time until the first hyperperiod, and the second is counting the number of tasks that exist in the current task set. With these nested loops, we search every time to find the task with the greatest priority, and if it is active (activation parameter = 1 -> if remaining time greater than 0), it executes, reducing its remaining time after every unit of time. The remaining time of every task is reinitialized in the beginning of its period, and if the remaining time of the task with the greatest priority is 0 (so the activation parameter is set to 0), then the algorithm searches again for the task with the second highest priority task. When there is no active task to be executed, we get Idle time units.

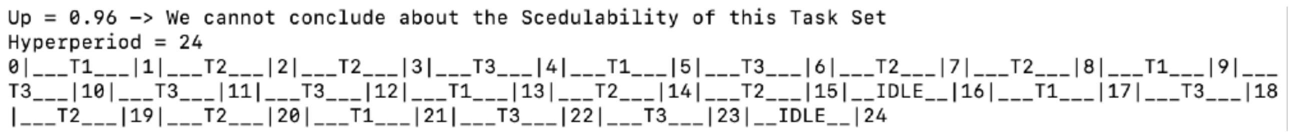


```
///////////////////////////////////////////////////////////////////////////////
//Sceduling for 1 Hyperperiod
///////////////////////////////////////////////////////////////////////////////
for (k=0; k<hyperperiod; k++) {
  greatest_priority_task = 0;
  for (i=0; i<task_num; i++) {
    if (k%tTask[i].period == 0) {
      if ((tTask[i].remaining_time>0) && (k>0)) {
        deadline_miss_array[k-1]=1;
      }
      tTask[i].remaining_time = tTask[i].comp_time;
      tTask[i].activation = 1;
      times[i]++;
    }
```



```
    if (tTask[greatest_priority_task].activation) {
      if (tTask[i].activation) {
        if (tTask[i].priority > tTask[greatest_priority_task].priority) {
          greatest_priority_task = i;
        }
      }
    }
    else {
      i=0;
      greatest_priority_task++;
      if (greatest_priority_task > task_num) {
        greatest_priority_task = 0;
      }
    }
  }

  if ((greatest_priority_task<task_num) && (greatest_priority_task>=0)) {
    printf("%d|___T%d___|", k, greatest_priority_task+1);
    tTask[greatest_priority_task].remaining_time--;
    if (tTask[greatest_priority_task].remaining_time == 0) {

      tTask[greatest_priority_task].activation = 0;
      if (times[greatest_priority_task] == 1) {
        response_time[greatest_priority_task] = k+1;
      }
      else if (times[greatest_priority_task]>1){
        response_time[greatest_priority_task] += k + 1 -
        (times[greatest_priority_task]-1)*tTask[greatest_priority_task].period;
      }
    }
    else {
      tTask[greatest_priority_task].activation = 1;

    }
  }
  else {
    printf("%d|__IDLE__|", k);
  }
}
printf("%d\n", hyperperiod);
```

The execution result of the previous example is shown below:

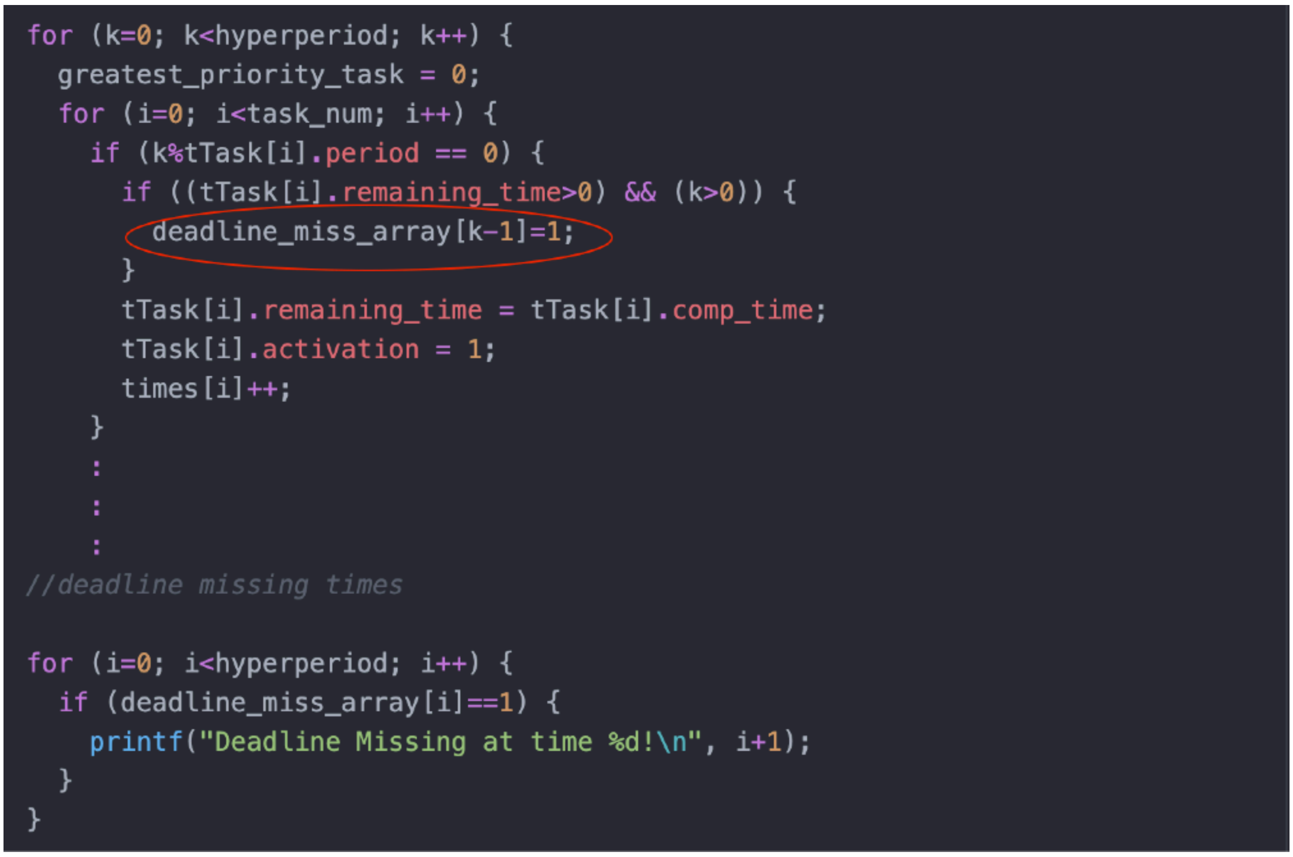


```
Up = 0.96 -> We cannot conclude about the Scedulability of this Task Set
Hyperperiod = 24
0|___T1___|1|___T2___|2|___T2___|3|___T3___|4|___T1___|5|___T3___|6|___T2___|7|___T2___|8|___T1___|9|___
T3___|10|___T3___|11|___T3___|12|___T1___|13|___T2___|14|___T2___|15|__IDLE__|16|___T1___|17|___T3___|18
|___T2___|19|___T2___|20|___T1___|21|___T3___|22|___T3___|23|__IDLE__|24
```

Display of main metrics

The main metrics that we want to display in the final step of our program, is the Average Response Time for each task, the Average Waiting Time for each task and the times that we have a Deadline Missing.

- The deadline missing, is the indication that a task did not finish its execution before its deadline, and in our case before its period (since we assumed that deadline = period). In order to display the deadline missing times, I created an array, with number of positions equal to the hyperperiod, and if at the time that a job starts, the previous job of the same task has not finished, then the current position of the deadline missing array gets an “1”, while if all previous jobs have finished their execution it gets a “0”.

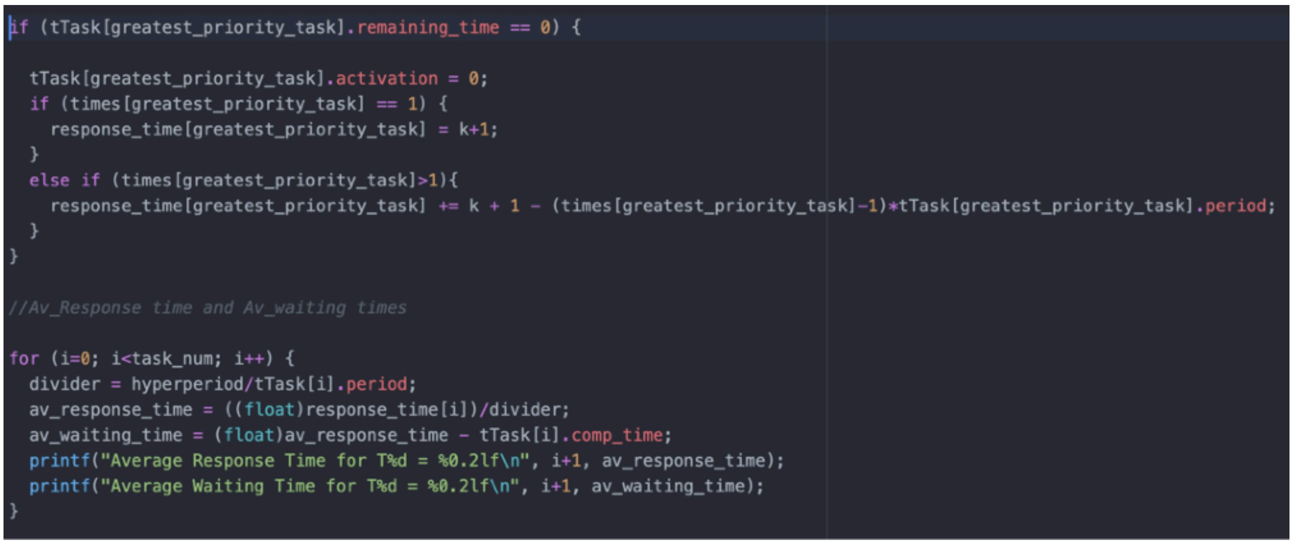


```
for (k=0; k<hyperperiod; k++) {
  greatest_priority_task = 0;
  for (i=0; i<task_num; i++) {
    if (k%tTask[i].period == 0) {
      if ((tTask[i].remaining_time>0) && (k>0)) {
        deadline_miss_array[k-1]=1;
      }
      tTask[i].remaining_time = tTask[i].comp_time;
      tTask[i].activation = 1;
      times[i]++;
    }
    :
    :
    :
//deadline missing times

for (i=0; i<hyperperiod; i++) {
  if (deadline_miss_array[i]==1) {
    printf("Deadline Missing at time %d!\n", i+1);
  }
}
```

- Next we have to calculate the average response time and the average waiting time for each task.

Average_Response_Time = AV(Finishing Time(ji) - Release Time(ji)), i=1 to n jobs

Average_Waiting_Time = Average_Response Time - period



```c
if (tTask[greatest_priority_task].remaining_time == 0) {

  tTask[greatest_priority_task].activation = 0;
  if (times[greatest_priority_task] == 1) {
    response_time[greatest_priority_task] = k+1;
  }
  else if (times[greatest_priority_task]>1){
    response_time[greatest_priority_task] += k + 1 - (times[greatest_priority_task]-1)*tTask[greatest_priority_task].period;
  }
}

//Av_Response time and Av_waiting times

for (i=0; i<task_num; i++) {
  divider = hyperperiod/tTask[i].period;
  av_response_time = ((float)response_time[i])/divider;
  av_waiting_time = (float)av_response_time - tTask[i].comp_time;
  printf("Average Response Time for T%d = %0.2lf\n", i+1, av_response_time);
  printf("Average Waiting Time for T%d = %0.2lf\n", i+1, av_waiting_time);
}
```

Examples

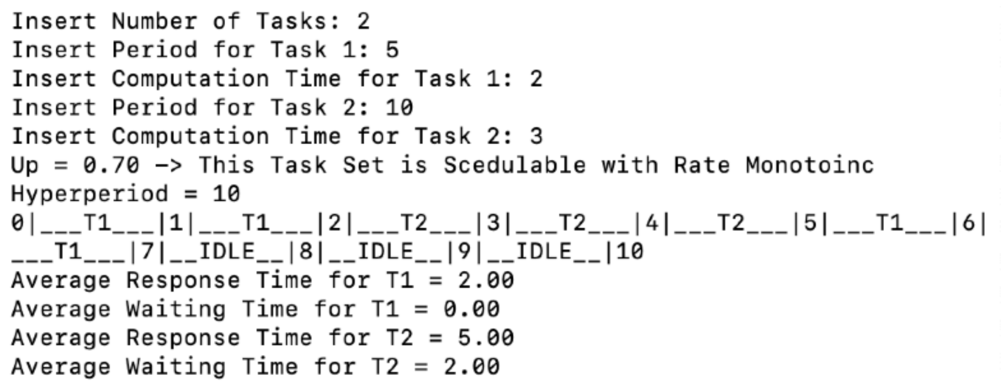
1. In the first example we assume we have :

$$\tau 1(T1 = 4, C1 = 1)$$
$$\tau 2(T2 = 6, C2 = 2)$$
$$\tau 3(T3 = 8, C3 = 3)$$

```
Insert Number of Tasks: 3
Insert Period for Task 1: 4
Insert Computation Time for Task 1: 1
Insert Period for Task 2: 6
Insert Computation Time for Task 2: 2
Insert Period for Task 3: 8
Insert Computation Time for Task 3: 3
Up = 0.96 -> We cannot conclude about the Scedulability of this Task Set
Hyperperiod = 24
0|___T1___|1|___T2___|2|___T2___|3|___T3___|4|___T1___|5|___T3___|6|___T2___|7
|___T2___|8|___T1___|9|___T3___|10|___T3___|11|___T3___|12|___T1___|13|___T2__
_|14|___T2___|15|__IDLE__|16|___T1___|17|___T3___|18|___T2___|19|___T2___|20|_
__T1___|21|___T3___|22|___T3___|23|__IDLE__|24
Deadline Missing at time 8!
Average Response Time for T1 = 1.00
Average Waiting Time for T1 = 0.00
Average Response Time for T2 = 2.50
Average Waiting Time for T2 = 0.50
Average Response Time for T3 = 3.67
Average Waiting Time for T3 = 0.67
```

2. Example with 2 Tasks:

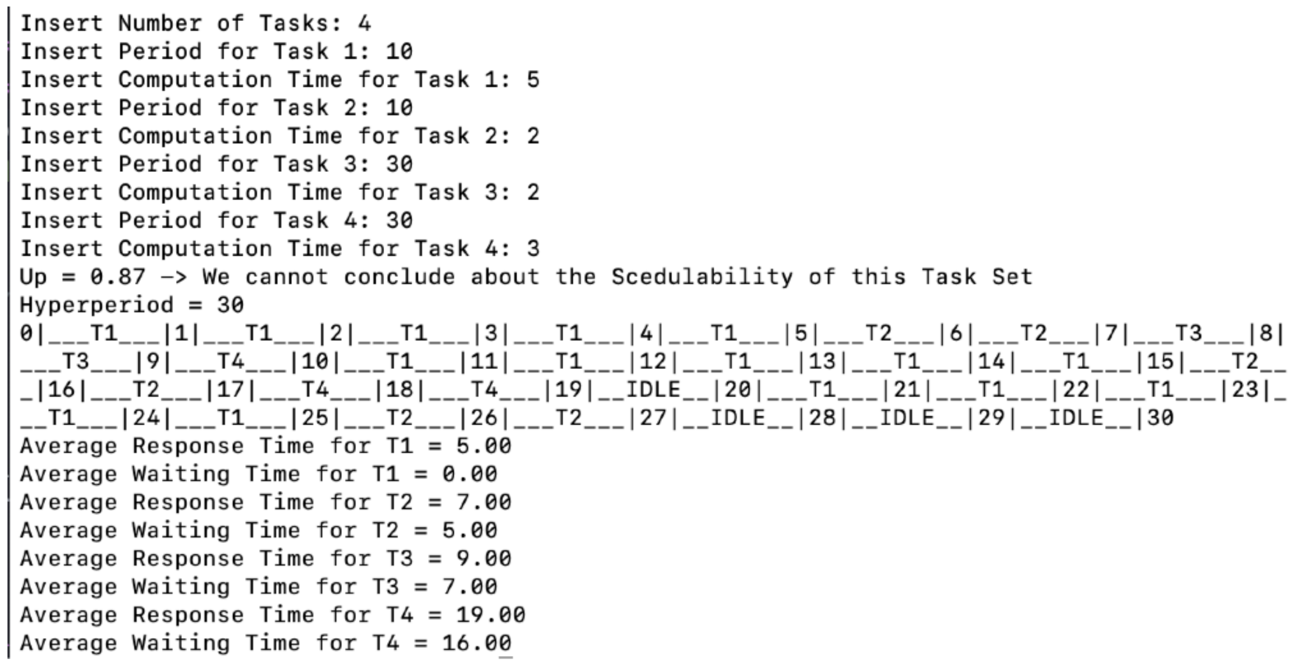
τ1 (T1 = 5, C1 = 2)
τ2 (T2 = 10, C2 = 3)



```
Insert Number of Tasks: 2
Insert Period for Task 1: 5
Insert Computation Time for Task 1: 2
Insert Period for Task 2: 10
Insert Computation Time for Task 2: 3
Up = 0.70 -> This Task Set is Scedulable with Rate Monotoinc
Hyperperiod = 10
0|___T1___|1|___T1___|2|___T2___|3|___T2___|4|___T2___|5|___T1___|6|
___T1___|7|__IDLE__|8|__IDLE__|9|__IDLE__|10
Average Response Time for T1 = 2.00
Average Waiting Time for T1 = 0.00
Average Response Time for T2 = 5.00
Average Waiting Time for T2 = 2.00
```

3. Example with 4 Tasks:

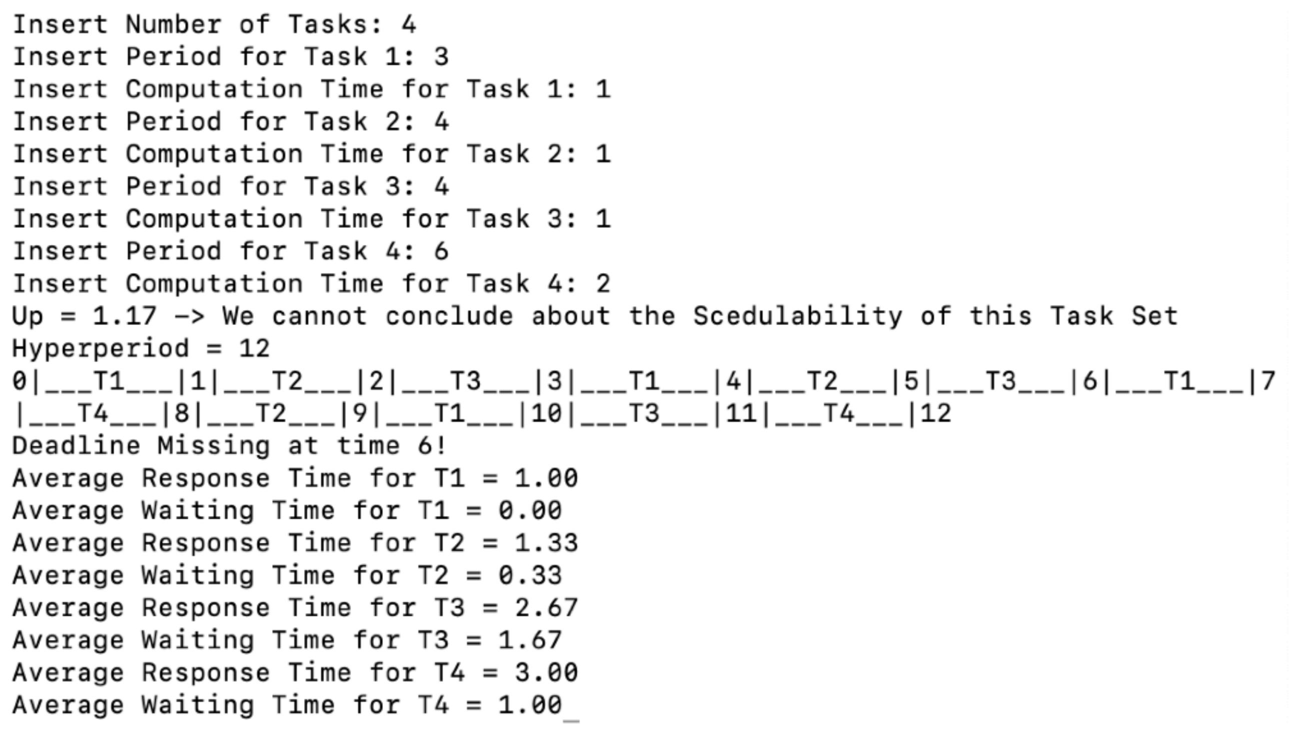
τ1(T1 = 10, C1 = 5)
τ2(T2 = 10, C2 = 2)
τ3(T3 = 30, C3 = 2)
τ4(T4 = 30, C4 = 3)



```
Insert Number of Tasks: 4
Insert Period for Task 1: 10
Insert Computation Time for Task 1: 5
Insert Period for Task 2: 10
Insert Computation Time for Task 2: 2
Insert Period for Task 3: 30
Insert Computation Time for Task 3: 2
Insert Period for Task 4: 30
Insert Computation Time for Task 4: 3
Up = 0.87 -> We cannot conclude about the Scedulability of this Task Set
Hyperperiod = 30
0|___T1___|1|___T1___|2|___T1___|3|___T1___|4|___T1___|5|___T2___|6|___T2___|7|___T3___|8|
___T3___|9|___T4___|10|___T1___|11|___T1___|12|___T1___|13|___T1___|14|___T1___|15|___T2__
_|16|___T2___|17|___T4___|18|___T4___|19|__IDLE__|20|___T1___|21|___T1___|22|___T1___|23|_
__T1___|24|___T1___|25|___T2___|26|___T2___|27|__IDLE__|28|__IDLE__|29|__IDLE__|30
Average Response Time for T1 = 5.00
Average Waiting Time for T1 = 0.00
Average Response Time for T2 = 7.00
Average Waiting Time for T2 = 5.00
Average Response Time for T3 = 9.00
Average Waiting Time for T3 = 7.00
Average Response Time for T4 = 19.00
Average Waiting Time for T4 = 16.00
```

4. Example with 4 not scedulable tasks:

τ1(T1 = 3, C1 = 1)
τ2(T2 = 4, C2 = 1)
τ3(T3 = 4, C3 = 1)
τ4(T4 = 6, C4 = 2)



```
Insert Number of Tasks: 4
Insert Period for Task 1: 3
Insert Computation Time for Task 1: 1
Insert Period for Task 2: 4
Insert Computation Time for Task 2: 1
Insert Period for Task 3: 4
Insert Computation Time for Task 3: 1
Insert Period for Task 4: 6
Insert Computation Time for Task 4: 2
Up = 1.17 -> We cannot conclude about the Scedulability of this Task Set
Hyperperiod = 12
0|___T1___|1|___T2___|2|___T3___|3|___T1___|4|___T2___|5|___T3___|6|___T1___|7
|___T4___|8|___T2___|9|___T1___|10|___T3___|11|___T4___|12
Deadline Missing at time 6!
Average Response Time for T1 = 1.00
Average Waiting Time for T1 = 0.00
Average Response Time for T2 = 1.33
Average Waiting Time for T2 = 0.33
Average Response Time for T3 = 2.67
Average Waiting Time for T3 = 1.67
Average Response Time for T4 = 3.00
Average Waiting Time for T4 = 1.00_
```

Conclusion

From the above examples we can reassure the functionality of our algorithm, which was designed in accordance with the Rate Monotonic Sceduler principles.

Appendix

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

struct Task {
  int period;
  int comp_time;
  int priority;
  int remaining_time;
  int activation;
};

int gcd(int a, int b)
{
    if (a == 0)
        return b;
    return gcd(b % a, a);
}

// Function to return LCM of two numbers
int lcm(int a, int b)
{
    return (a / gcd(a, b)) * b;
}

void create_task(struct Task *n, int i) {

    printf("Insert Period for Task %d: ", i);
    scanf("%d", &n->period);
    printf("Insert Computation Time for Task %d: ", i);
    scanf("%d", &n->comp_time);
}

int main(int argc, char* argv[]) {

  int task_num;
  int i, j, k;
  float util_factor, test;
  int greatest_priority_task, divider;

  printf("Insert Number of Tasks: ");
  scanf("%d", &task_num);

  int response_time[task_num];

  ////////////////////////////////////////////////////////////////////////
  //Task Initialization
  ////////////////////////////////////////////////////////////////////////

  struct Task tTask[task_num];

  for (i=0;i<task_num;i++) {
    create_task(&tTask[i], i+1);
    tTask[i].priority = task_num;
    tTask[i].remaining_time = tTask[i].comp_time;
    tTask[i].activation = 1;
    response_time[i]=0;
  }

  ////////////////////////////////////////////////////////////////////////////
  //Scedulability Test
  ////////////////////////////////////////////////////////////////////////////

  switch(task_num) {
    case 1:
      test = 1.0;
      break;
    case 2:
      test = 0.83;
      break;
    case 3:
      test = 0.78;
      break;
    case 4:
      test = 0.76;
      break;
    case 5:
      test = 0.74;
      break;
    default:
      test = 0.69;
      break;
  }

  util_factor = 0.0;
  for (i=0; i<task_num; i++) {
    util_factor += ((float)tTask[i].comp_time)/((float)tTask[i].period);
  }

  printf("Up = %0.2lf -> ", util_factor);
  if (util_factor <= test) {
    printf("This Task Set is Scedulable with Rate Monotoinc\n");
  }
  else {
    printf("We cannot conclude about the Scedulability of this Task 
Set\n");
  }

  ////////////////////////////////////////////////////////////////////////////
  //Hyperperiod Calculation of 3 tasks
  ////////////////////////////////////////////////////////////////////////////

  int hyp1 = lcm(tTask[0].period, tTask[1].period);
  int hyp2 = lcm(hyp1, tTask[2].period);
  int hyperperiod = lcm(hyp2, tTask[3].period);
  printf ("Hyperperiod = %d\n", hyperperiod);

  ////////////////////////////////////////////////////////////////////////////
  //Priority assignment
  ////////////////////////////////////////////////////////////////////////////
  for (i=0; i<task_num; i++) {
    for (j=i+1; j<task_num; j++) {
      if (tTask[i].period > tTask[j].period) {
        tTask[i].priority--;
      }
      else if (tTask[i].period == tTask[j].period){
        tTask[j].priority = tTask[i].priority;
      }
      else {
        tTask[j].priority--;
      }
    }
  }

  ///////////////////////////////////////////////////////////////////////////
  //Deadline Missing Array Initialization
  ///////////////////////////////////////////////////////////////////////////

  int deadline_miss_array[hyperperiod];

  for(i=0;i<hyperperiod;i++) {
    deadline_miss_array[i] = 0;
  }

  ///////////////////////////////////////////////////////////////////////////
  //Sceduling for 1 Hyperperiod
  ///////////////////////////////////////////////////////////////////////////

  int times[task_num];

  for(i=0;i<task_num;i++) {
    times[i]=0;
  }

  for (k=0; k<hyperperiod; k++) {
    greatest_priority_task = 0;
    for (i=0; i<task_num; i++) {
      if (k%tTask[i].period == 0) {
        if ((tTask[i].remaining_time>0) && (k>0)) {
          deadline_miss_array[k-1]=1;
        }
        tTask[i].remaining_time = tTask[i].comp_time;
        tTask[i].activation = 1;
        times[i]++;
      }
      if (tTask[greatest_priority_task].activation) {
        if (tTask[i].activation) {
          if (tTask[i].priority > tTask[greatest_priority_task].priority)
{
            greatest_priority_task = i;
          }
        }
      }
      else {
        i=0;
        greatest_priority_task++;
        if (greatest_priority_task > task_num) {
          greatest_priority_task = 0;
        }
      }
    }

    if ((greatest_priority_task<task_num) && (greatest_priority_task>=0))
{
      printf("%d|___T%d___|", k, greatest_priority_task+1);
      tTask[greatest_priority_task].remaining_time--;
      if (tTask[greatest_priority_task].remaining_time == 0) {

        tTask[greatest_priority_task].activation = 0;
        if (times[greatest_priority_task] == 1) {
          response_time[greatest_priority_task] = k+1;
        }
        else if (times[greatest_priority_task]>1){
          response_time[greatest_priority_task] += k + 1 -
(times[greatest_priority_task]-1)*tTask[greatest_priority_task].period;
        }
      }
      else {
        tTask[greatest_priority_task].activation = 1;

      }
    }
    else {
      printf("%d|__IDLE__|", k);
    }
  }
  printf("%d\n", hyperperiod);

  ////////////////////////////////////////////////////////////////////////
  //Metrics
  ////////////////////////////////////////////////////////////////////////

  float av_waiting_time, av_response_time;

  for (i=0; i<hyperperiod; i++) {
    if (deadline_miss_array[i]==1) {
      printf("Deadline Missing at time %d!\n", i+1);
    }
  }
  for (i=0; i<task_num; i++) {
    divider = hyperperiod/tTask[i].period;
    av_response_time = ((float)response_time[i])/divider;
    av_waiting_time = (float)av_response_time - tTask[i].comp_time;
    printf("Average Response Time for T%d = %0.2lf\n", i+1,
av_response_time);
    printf("Average Waiting Time for T%d = %0.2lf\n", i+1,
av_waiting_time);
  }

  return (0);
}
```