Coursera's
Development of Real-Time Systems
Peer-graded Assignment: Assignment 4

Requirement

1- Simulation assignment

The assignment is to modify a real-time simulator to verify feasibility of a set of tasks.

- Modify the python code in P_RM.py to use firstfit instead of the current algorithm. Please follow the steps in This document.

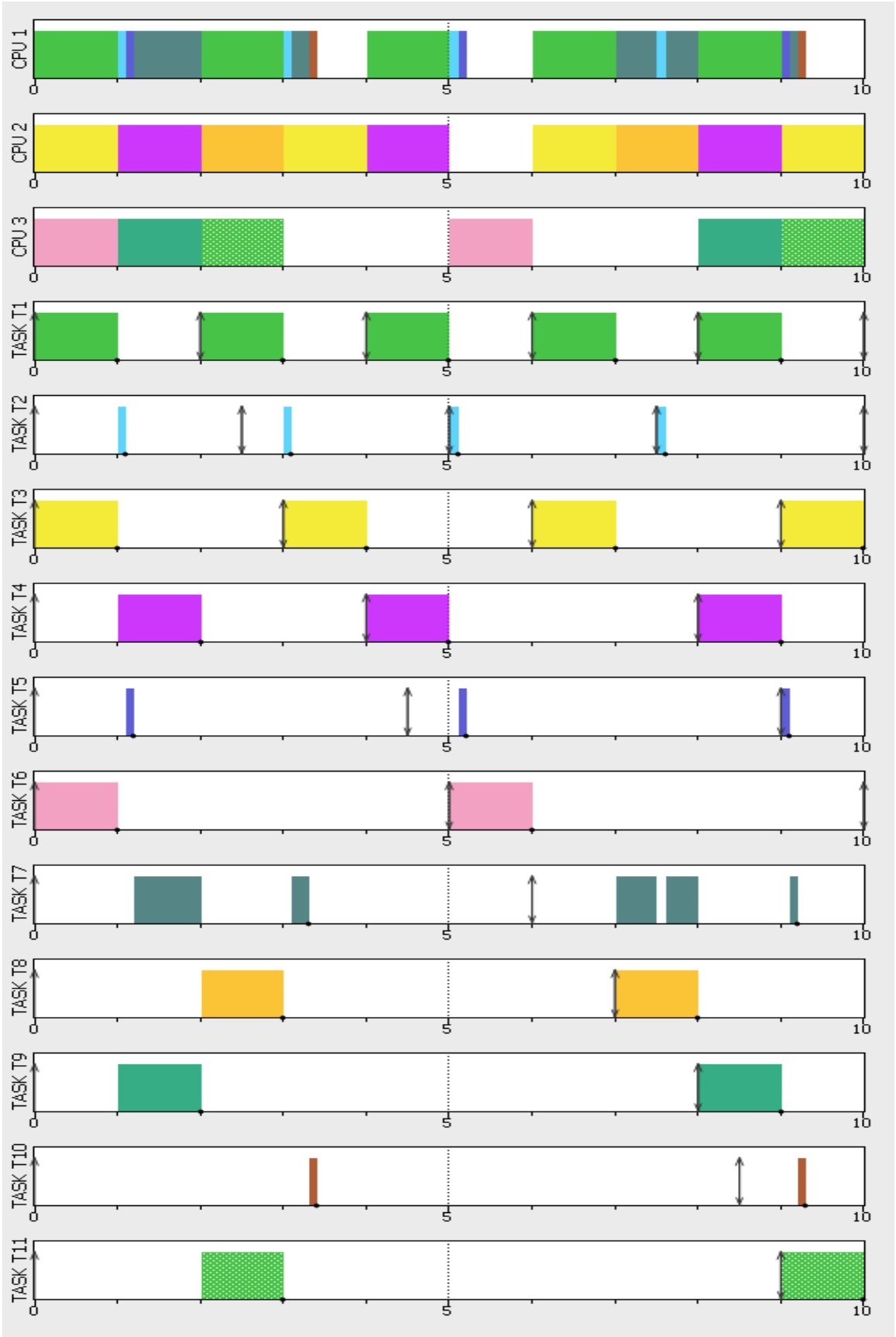
- Hint: Instead of scheduling the task to the CPU with the lowest utilization chose the first one which has a lower utilization than Urm(x+1) where x is the already scheduled tasks on the CPU

- Hint2: have a look at the def packer(self) function in the file P_RM.py

- Schedule the following task set on three processors using your modified algorithm.

- T1(2,1) T2(2.5,0.1) T3(3,1) T4(4,1) T5(4.5,0.1) T6(5,1) T7(6,1) T8(7,1) T9(8,1) T10(8.5,0.1) T11(9,1)

- The schedule should look like the following image:


CPU 1
CPU 2
CPU 3
TASK T1
TASK T2
TASK T3
TASK T4
TASK T5
TASK T6
TASK T7
TASK T8
TASK T9
TASK T10
TASK T11
0
5
10

How to Modify Scheduler in SimSo

1. Go to the location of your SimSo installation

2. Go to the folder “schedulers”

3. Copy the file P_RM.py to a folder with read/write permissions (for example your Downloads folder)

4. Change filename to P_RMFF.py

5. Open the file P_RMFF.py with your favorite text editor as seen in Figure 3

6. Modify the code according to the description in the assignment

7. Change the scheduler info to SchedulerInfo("simso.schedulers.RM_monoFF")) in P_RMFF.py

8. Copy the file P_RMFF.py back to the SimSo scheduler folder as seen in Figure 4

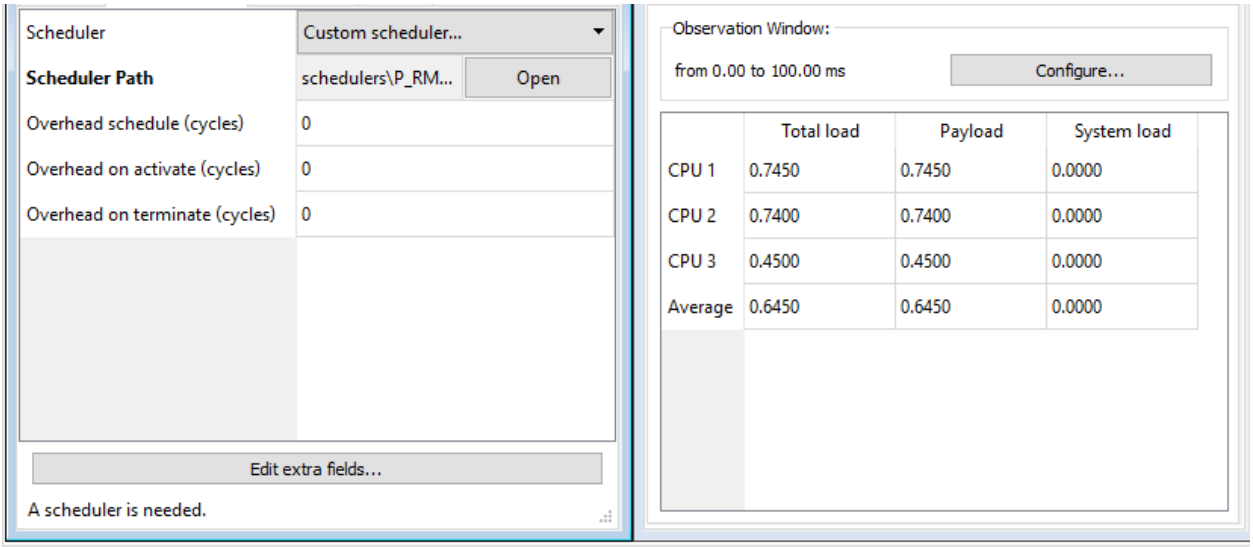
9. Start SimSo and select “Custom Scheduler”

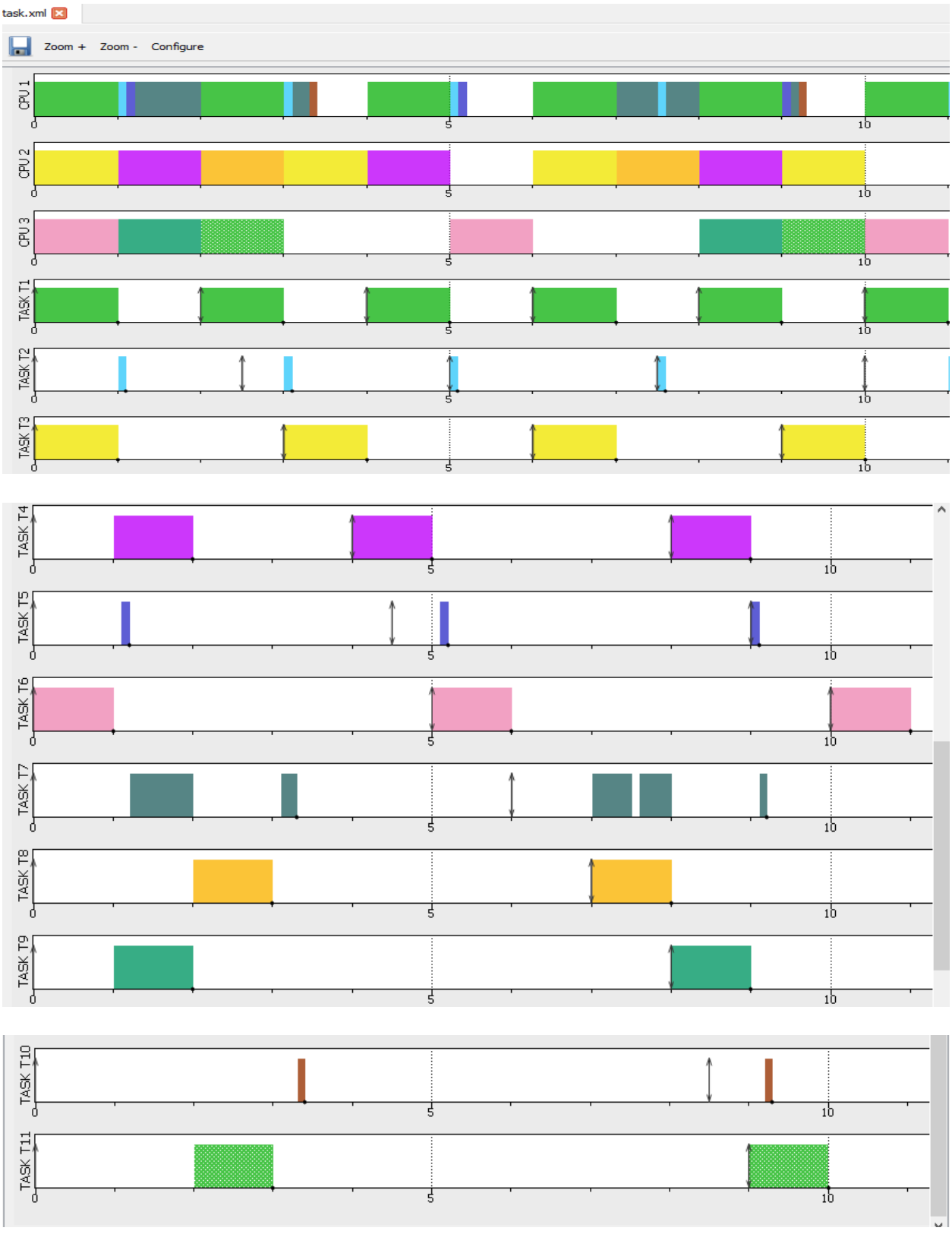
10. In “Scheduler Path” click “open” select the location of the P_RMFF.py file

11. Schedule your tasks

Task1- Scheduler

id	Name	Task type	Abort on miss	Act. Date (ms)	Period (ms)	List of Act. dates (ms)	Deadline (ms)	WCET (ms)	Followed by
1	TASK T1	Periodic	Yes	0.0	2.0	-	2.0	1.0	
2	TASK T2	Periodic	Yes	0.0	2.5	-	2.5	0.1	
3	TASK T3	Periodic	Yes	0.0	3.0	-	3.0	1.0	
4	TASK T4	Periodic	Yes	0.0	4.0	-	4.0	1.0	
5	TASK T5	Periodic	Yes	0.0	4.5	-	4.5	0.1	
6	TASK T6	Periodic	Yes	0.0	5.0	-	5.0	1.0	
7	TASK T7	Periodic	Yes	0.0	6.0	-	6.0	1.0	
8	TASK T8	Periodic	Yes	0.0	7.0	-	7.0	1.0	
9	TASK T9	Periodic	Yes	0.0	8.0	-	8.0	1.0	
10	TASK T10	Periodic	Yes	0.0	8.5	-	8.5	0.1	
11	TASK T11	Periodic	Yes	0.0	9.0	-	9.0	1.0	




task.xml
Zoom +
Zoom -
Configure
CPU 1
CPU 2
CPU 3
TASK T1
TASK T2
TASK T3
TASK T4
TASK T5
TASK T6
TASK T7
TASK T8
TASK T9
TASK T10
TASK T11
0
5
10

- Source Code (P_RM.py Modified)

```
from simso.schedulers import scheduler

@scheduler("simso.schedulers.P_RM")
class P_RM(PartitionedScheduler):
    def init(self):
        PartitionedScheduler.init(
            self, SchedulerInfo("simso.schedulers.RM_mono"))

    def packer(self):
        # First Fit
        cpus = [[cpu, 0] for cpu in self.processors]

        TaskNum = [0] * len(cpus)

        for task in self.task_list:
            #m = cpus[0][1]
            j = 0

            # Find the processor with the lowest load.
            for i, c in enumerate(cpus):

                n = (TaskNum[i]+1.0)
                URM =  n * (pow(2.0, 1/n) - 1.0)
                U = (c[1] + (float(task.wcet) / task.period))

                if U < URM:
                    j = i
                    break

            TaskNum[j] +=1

            # Affect it to the task.
            self.affect_task_to_processor(task, cpus[j][0])

            # Update utilization.
            cpus[j][1] += float(task.wcet) / task.period
        return True
```

1- Programming Assignment

- Create a task "matrixtask" containing the following functionality:

```
#define SIZE 10
#define ROW SIZE
#define COL SIZE
static void matrix_task()
{
  int i;
  double **a = (double **)pvPortMalloc(ROW * sizeof(double*));
  for (i = 0; i < ROW; i++) a[i] = (double *)pvPortMalloc(COL * sizeof(double));
  double **b = (double **)pvPortMalloc(ROW * sizeof(double*));
  for (i = 0; i < ROW; i++) b[i] = (double *)pvPortMalloc(COL * sizeof(double));
  double **c = (double **)pvPortMalloc(ROW * sizeof(double*));
  for (i = 0; i < ROW; i++) c[i] = (double *)pvPortMalloc(COL * sizeof(double));

  double sum = 0.0;
  int j, k, l;

  for (i = 0; i < SIZE; i++) {
    for (j = 0; j < SIZE; j++) {
      a[i][j] = 1.5;
      b[i][j] = 2.6;
    }
  }

  while (1) {
    /*
    * In an embedded systems, matrix multiplication would block the CPU for a
      long time
    * but since this is a PC simulator we must add one additional dummy delay.
    */
    long simulationdelay;
    for (simulationdelay = 0; simulationdelay<1000000000; simulationdelay++)
      ;
    for (i = 0; i < SIZE; i++) {
      for (j = 0; j < SIZE; j++) {
        c[i][j] = 0.0;
      }
    }

    for (i = 0; i < SIZE; i++) {
      for (j = 0; j < SIZE; j++) {
        sum = 0.0;
        for (k = 0; k < SIZE; k++) {
          for (l = 0; l<10; l++) {
            sum = sum + a[i][k] * b[k][j];
          }
        }
        c[i][j] = sum;
      }
    }
```

- Create a queue and send the content of (double **)c to the queue in matrixtask with before the vTaskDelay() call (hint: place the c variable in a struct).

- Create a reader task which reads the content of the queue in case there is something in the queue.

- In case the queue has some content it should save the data in a local (double **) variable.

- Print out the content of the (double **)c variable in case the content is updated. The data transferred from c should be a 10x10 matrix with the value 390 in each slot.

<u>Task Solution :-</u>

```
Send Data
Received Data = 390.000000
Received Data = 390.000000
Received Data = 390.000000
Received Data = 390.000000
Received Data = 390.000000
Received Data = 390.000000
Received Data = 390.000000
Received Data = 390.000000
Received Data = 390.000000
Received Data = 390.000000
Received Data = 390.000000
Received Data = 390.000000
Received Data = 390.000000
Received Data = 390.000000
Received Data = 390.000000
Received Data = 390.000000
Received Data = 390.000000
Received Data = 390.000000
Received Data = 390.000000
Received Data = 390.000000
Received Data = 390.000000
Received Data = 390.000000
Received Data = 390.000000
Received Data = 390.000000
Received Data = 390.000000
Received Data = 390.000000
Received Data = 390.000000
Received Data = 390.000000
Received Data = 390.000000
```

- **Source Code**

1- Create Tasks

```
void main_task( void )
{

    xTaskCreate((pdTASK_CODE)matrix_task, (signed char *)"Matrix", 1000, NULL, 3, &matrix_handle);

    xTaskCreate((pdTASK_CODE)Receiver_Task, (signed char *)"Receiver", 1000, NULL, 2, &Receiver_handle);

    /* Create a queue capable of containing 10 * 10 unsigned double values. */
    xQueue = xQueueCreate( 10 * 10, sizeof(xMessage) );

    if( xQueue == NULL )
    {
        /* Queue was not created and must not be used. */
        printf("Queue was not created\n");
    }
    // This starts the real-time scheduler
    vTaskStartScheduler();

    while(1)
    {

    }
}
```

2- Create Struct

```
struct AMessage
{
    double **c;
}xMessage;
```

3- Modified the Matrix Task to send Data

```
        xMessage.c = (double **)pvPortMalloc(ROW * sizeof(double*));
        for (i = 0; i < ROW; i++) xMessage.c[i] = \
                (double *)pvPortMalloc(COL * sizeof(double));
        for (i = 0; i < SIZE; i++)
        {
            for (j = 0; j < SIZE; j++)
            {
                xMessage.c[i][j] = c[i][j];
                c[i][j] = 0.0;
            }
        }

        printf("Send Data \n");
        fflush(stdout);

        if( xQueueSend( xQueue,  ( void * ) &xMessage, ( TickType_t ) 10 ) != pdPASS )
        {
            /* Failed to post the message, even after 10 ticks. */
            printf("Failed to post the message, even after 10 ticks\n");

        }
        vTaskDelay(100);
    }
}
```

4- Create Task To Receive The Data

```c
static void Receiver_Task()
{
    struct AMessage ReceivedValue;

    for( ;; )
    {
        if(xQueue != NULL)
        {
            if (xQueueReceive( xQueue, &ReceivedValue, ( TickType_t ) 10 ) == pdPASS)
            {
                for (int i = 0; i < SIZE; i++)
                {
                    for (int j = 0; j < SIZE; j++)
                    {
                        printf("Received Data = %f \n",ReceivedValue.c[i][j]);
                        fflush(stdout);
                    }
                }
            }
        }

        vTaskDelay(100);
    }
}
```