

Development of Real-Time Systems

Assignment 5

Elyasin Shaladi

July 17, 2016

Contents

1	Simulation assignment	3
1.1	Simulation assignment — Modification of a real-time simulator .	3
1.1.1	Modifying the partitioned RM scheduler	3
2	Programming assignment	6
2.1	Provide a screen-shot of the running system	9

List of Tables

1	Tasks T1 to T11	3
---	---------------------------	---

Listings

1	Affect a task to the first processor with utilization lower than U_{rm}	4
2	Creation of tasks and queue	6
3	Definition of the Message <i>struct</i>	6
4	Sending a message to the message queue	7
5	Receiving a message from the message queue	7

List of Figures

1	Screen-shot of the Gantt diagram of the simulation	5
2	Screen-shot FreeRTOS scheduler execution	9

1 Simulation assignment

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

1.1 Simulation assignment — Modification of a real-time simulator

The partitioned RM scheduler in SimSo is modified to schedule according to a First-Fit algorithm described in the assignment description. Instead of scheduling the task to the CPU with the lowest utilization we chose the first one, which has a lower utilization than $U_{rm}(x + 1)$ where x is the already scheduled tasks on the respective CPU.

Task	p_i	e_i	D_i
Task 1	2	1	2
Task 2	2.5	0.1	2.5
Task 3	3	1	3
Task 4	4	1	4
Task 5	4.5	0.1	4.5
Task 6	5	1	5
Task 7	7	1	6
Task 8	7	1	7
Task 9	8	1	8
Task 10	8.5	0.1	8.5
Task 11	9	1	9

Table 1: Tasks T1 to T11

1.1.1 Modifying the partitioned RM scheduler

First I copied and modified the P_RM.py file and named it P_RM_assignment_5.py. In order for SimSo to recognize the custom scheduler the class name of the partitioned scheduler must correspond to the file name. Therefore we need to modify the line

```
...  
class P_RM(PartitionedScheduler):  
...
```

to

```
...  
class P_RM_assignment_5(PartitionedScheduler):  
...
```

We need to modify the *packer* function and implement the First Fit algorithm. In this First Fit algorithm we assign a task to the first CPU with utilization lower than $U_{rm}(n+1)$ where n is the number of task already assigned to the respective CPU.

```
def packer(self):
    # First Fit RM
    cpus = [[cpu, 0, 0] for cpu in self.processors]
    for task in self.task_list:
        j = -1

        # Find first the processor with utilization lower
        # than U_rm(n+1) where n is the number of tasks
        # affected to cpu
        for i, c in enumerate(cpus):
            u = c[1] + float(task.wcet) / task.period
            u_rm = (c[2]+1) * (2*(1/float((c[2]+1)))) - 1
            if u <= u_rm:
                j = i
                break

        if j == -1:
            return False

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

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

        # Update number of tasks on cpu
        cpus[j][2] += 1
    return True
```

Listing 1: Affect a task to the first processor with utilization lower than U_{rm}

Below you can see the result of the scheduling simulation in form of a screen-shot of a Gantt diagram. With this report you'd have also received the SimSo files so that you can perform the simulation yourself. Simply open the file *custom_scheduler.xml* with SimSo and set the *Scheduler Path* to where the file *P_RM_assignment.5.py* is located on your machine.

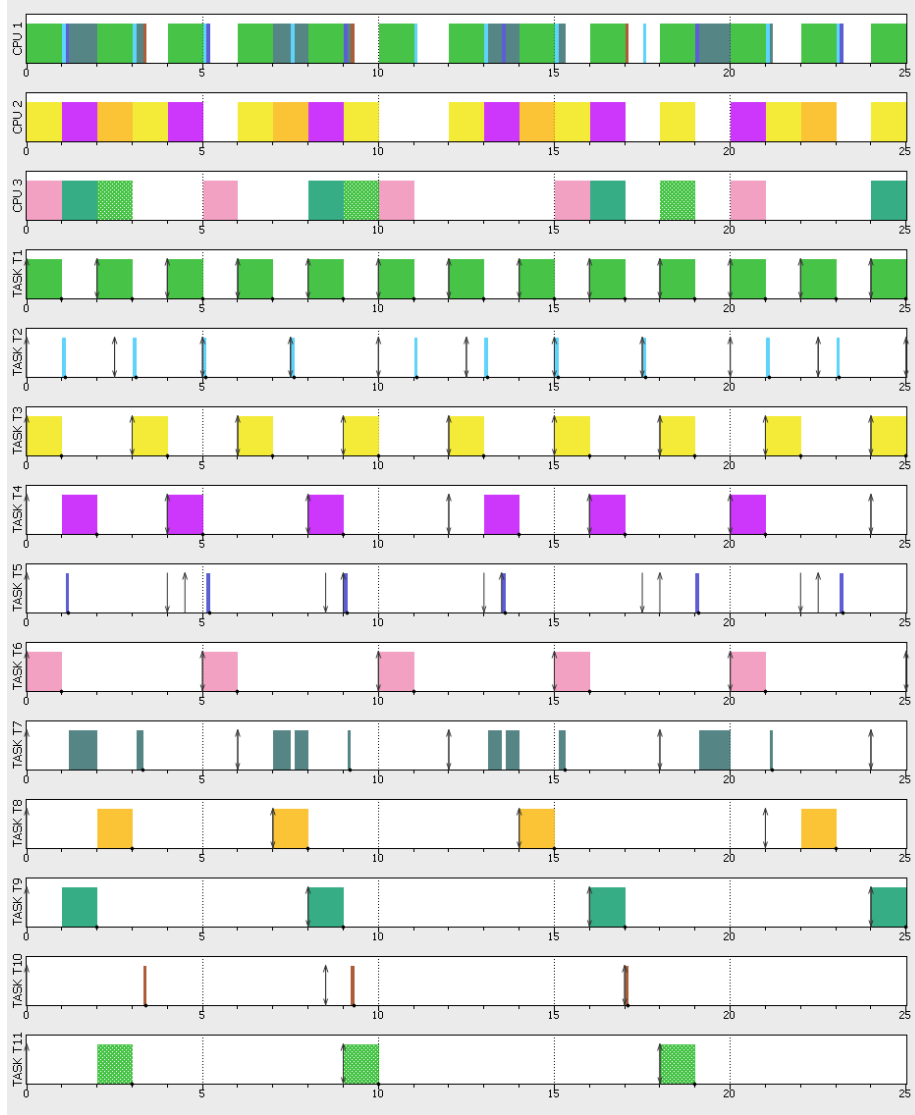


Figure 1: Screen-shot of the Gantt diagram of the simulation

2 Programming assignment

In this programming assignment, we will use message queues in FreeRTOS. We will use the matrix task with the functionality given in the Assignment 5, which is a computationally intensive task, in order to implement the message queue demo.

The queue is created in the *main* function with a queue length of 3 and an item size of a pointer, which is a pointer to a *struct*.

In the main function the tasks matrix task and reader task are also created. I chose to give both the tasks the same priority.

```
...
// Handles for tasks and queues
xTaskHandle matrix_handle = 0;
xTaskHandle reader_handle = 0;
QueueHandle_t matrix_queue = 0;
...
int main(void) {
    /* This demo uses heap_5.c, so start by defining some heap
       regions. This is only done to provide an example as this
       demo could easily create one large heap region instead of
       multiple smaller heap regions
       */
    prvInitialiseHeap();

    /*FreeRTOS scheduling 1*/
    /*Create the matrix task */
    xTaskCreate((pdTASK_CODE) matrix_task, (signed char * ) "Matrix",
               1000, NULL, 3, &matrix_handle);
    /*Create the reader task */
    xTaskCreate((pdTASK_CODE) reader_task, (signed char * ) "Reader",
               1000, NULL, 3, &reader_handle);
    /*Create the matrix queue */
    matrix_queue = xQueueCreate(3, sizeof(struct Message *));

    if (matrix_queue == NULL)
        printf("Failed to create matrix queue.\n");

    //This starts the real-time scheduler
    vTaskStartScheduler();
    for (;;)
        ;
    return 0;
}
```

Listing 2: Creation of tasks and queue

The structure of the message is simple as it holds only a pointer to a pointer to a double data type:

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

Listing 3: Definition of the Message *struct*

The matrix task is modified so that it sends a message to the message queue before it is put to sleep. The message will contain a the value of the variable *c*, which is a pointer to a pointer to a double data type. Messaging in FreeRTOS is done by copying the values to the message queue as described in the FreeRTOS documentation. So we don't need to deal with data races in this case (the matrix task allocates new memory at each job run). However, concurrency can be a problem in certain cases here. But since it was not required to solve concurrency in this assignment I just skip that step. (A solution would probably be to use semaphores.)

```
...
    struct Message *pxMessage;
    xMessage.c = c;
    pxMessage = &xMessage;
    if (xQueueSend(matrix_queue, &pxMessage, (TickType_t) 0)
        != pdPASS) {
        printf("Failed to post the message.\n");
        fflush(stdout);
    }
    vTaskDelay(100);
...
```

Listing 4: Sending a message to the message queue

Finally the reader task is defined. The reader task does not block if a message is not immediately available (for this simple example this should be sufficient I think). It keeps polling the queue and when it receives a message then it prints out the data.

The rows of the received matrix data is printed out line by line. On small screens it might be a little bit difficult to read. Feel free to modify the output according to your needs.

```
// Reader task to receive from the queue
void reader_task(void *pvParameters) {

    while (1) {
        if (matrix_queue != 0) {

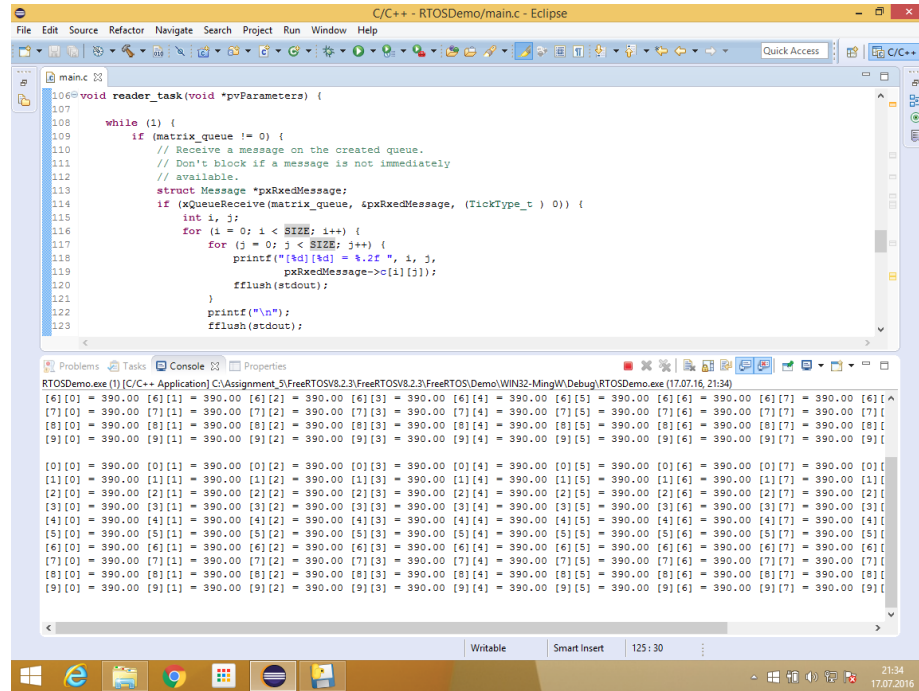
            if (uxQueueMessagesWaiting(matrix_queue)) {
                // Receive a message on the created queue.
                // Don't block if a message is not immediately
                // available.
                struct Message *pRxedMessage;
                if (xQueueReceive(matrix_queue, &pRxedMessage,
                    (TickType_t) 0)) {
                    int i, j;
                    for (i = 0; i < SIZE; i++) {
                        for (j = 0; j < SIZE; j++) {
                            printf("[%d][%d] = %.2f", i, j,
                                pRxedMessage->c[i][j]);
                            fflush(stdout);
                        }
                        printf("\n");
                        fflush(stdout);
                    }
                    printf("\n");
                    fflush(stdout);
                }
            }
        }
    }
}
```

```
}  
}  
}  
}  
}
```

Listing 5: Receiving a message from the message queue

2.1 Provide a screen-shot of the running system

I use Eclipse for development. The screen-shot shows the project in execution.



```
106 void reader_task(void *pvParameters) {
107     while (1) {
108         if (matrix_queue != 0) {
109             // Receive a message on the created queue.
110             // Don't block if a message is not immediately
111             // available.
112             struct Message *pxRxdMessage;
113             if (xQueueReceive(matrix_queue, &pxRxdMessage, (TickType_t) 0)) {
114                 int i, j;
115                 for (i = 0; i < SIZE; i++) {
116                     for (j = 0; j < SIZE; j++) {
117                         printf("[%d][%d] = %.2f ", i, j,
118                             pxRxdMessage->c[i][j]);
119                         fflush(stdout);
120                     }
121                     printf("\n");
122                     fflush(stdout);
123                 }
124             }
125         }
126     }
127 }
```

RTOSDemo.exe (1) [C/C++ Application] C:\Assignment_5\FreeRTOSV8.2.3\FreeRTOS\Demo\WIN32-MingW_Debug\RTOSDemo.exe (17.07.16, 21:34)

[6][0] = 390.00 [6][1] = 390.00 [6][2] = 390.00 [6][3] = 390.00 [6][4] = 390.00 [6][5] = 390.00 [6][6] = 390.00 [6][7] = 390.00 [6][8] = 390.00 [6][9] = 390.00 [7][0] = 390.00 [7][1] = 390.00 [7][2] = 390.00 [7][3] = 390.00 [7][4] = 390.00 [7][5] = 390.00 [7][6] = 390.00 [7][7] = 390.00 [7][8] = 390.00 [7][9] = 390.00 [8][0] = 390.00 [8][1] = 390.00 [8][2] = 390.00 [8][3] = 390.00 [8][4] = 390.00 [8][5] = 390.00 [8][6] = 390.00 [8][7] = 390.00 [8][8] = 390.00 [8][9] = 390.00 [9][0] = 390.00 [9][1] = 390.00 [9][2] = 390.00 [9][3] = 390.00 [9][4] = 390.00 [9][5] = 390.00 [9][6] = 390.00 [9][7] = 390.00 [9][8] = 390.00 [9][9] = 390.00 [0][0] = 390.00 [0][1] = 390.00 [0][2] = 390.00 [0][3] = 390.00 [0][4] = 390.00 [0][5] = 390.00 [0][6] = 390.00 [0][7] = 390.00 [0][8] = 390.00 [0][9] = 390.00 [1][0] = 390.00 [1][1] = 390.00 [1][2] = 390.00 [1][3] = 390.00 [1][4] = 390.00 [1][5] = 390.00 [1][6] = 390.00 [1][7] = 390.00 [1][8] = 390.00 [1][9] = 390.00 [2][0] = 390.00 [2][1] = 390.00 [2][2] = 390.00 [2][3] = 390.00 [2][4] = 390.00 [2][5] = 390.00 [2][6] = 390.00 [2][7] = 390.00 [2][8] = 390.00 [2][9] = 390.00 [3][0] = 390.00 [3][1] = 390.00 [3][2] = 390.00 [3][3] = 390.00 [3][4] = 390.00 [3][5] = 390.00 [3][6] = 390.00 [3][7] = 390.00 [3][8] = 390.00 [3][9] = 390.00 [4][0] = 390.00 [4][1] = 390.00 [4][2] = 390.00 [4][3] = 390.00 [4][4] = 390.00 [4][5] = 390.00 [4][6] = 390.00 [4][7] = 390.00 [4][8] = 390.00 [4][9] = 390.00 [5][0] = 390.00 [5][1] = 390.00 [5][2] = 390.00 [5][3] = 390.00 [5][4] = 390.00 [5][5] = 390.00 [5][6] = 390.00 [5][7] = 390.00 [5][8] = 390.00 [5][9] = 390.00 [6][0] = 390.00 [6][1] = 390.00 [6][2] = 390.00 [6][3] = 390.00 [6][4] = 390.00 [6][5] = 390.00 [6][6] = 390.00 [6][7] = 390.00 [6][8] = 390.00 [6][9] = 390.00 [7][0] = 390.00 [7][1] = 390.00 [7][2] = 390.00 [7][3] = 390.00 [7][4] = 390.00 [7][5] = 390.00 [7][6] = 390.00 [7][7] = 390.00 [7][8] = 390.00 [7][9] = 390.00 [8][0] = 390.00 [8][1] = 390.00 [8][2] = 390.00 [8][3] = 390.00 [8][4] = 390.00 [8][5] = 390.00 [8][6] = 390.00 [8][7] = 390.00 [8][8] = 390.00 [8][9] = 390.00 [9][0] = 390.00 [9][1] = 390.00 [9][2] = 390.00 [9][3] = 390.00 [9][4] = 390.00 [9][5] = 390.00 [9][6] = 390.00 [9][7] = 390.00 [9][8] = 390.00 [9][9] = 390.00

Writable Smart Insert 125:30

Figure 2: Screen-shot FreeRTOS scheduler execution