Assignment 2

Anders L. Hurum
Development of Real-Time Systems
EIT Digital
andershurum@gmail.com

September 17, 2017

Abstract

The second assignment for the Real-Time systems course consisted of developing an application for exploring how priorities, execution-time, periods influence eachother in practise in a Real-Time system.

For more details on the assignment, see the `assignment_2.md` document in the repository at github.

`http://github.com/peakbreaker/tuts_FreeRTOS`

Introduction

To fullfill the requirements by the assignment it is planned to implement the RTOS with three tasks:

- prioritysettask
- matrixtask
- communicationtask

The idea is that matrixtask will be responsible charge of matrix calculations, while communicationstask will handle communication to a peripheral (like transmitting the calculations from matrixtask). Finally prioritysettask will measure and manage the tasks, and set priorities according to the requirements of our application.

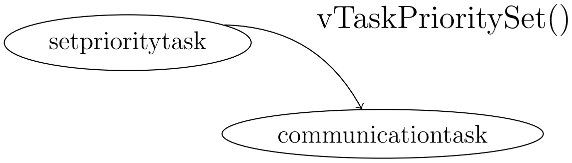


Figure 1: Tasks

We aim with this to answer questions relating to execution priorities in FreeRTOS, and find out how priorities, periods and pre-emptiveness influence the behavior of execution in Real Time systems.

Code

Worth noting in the code is that the task handlers are avaliable to all tasks in the main program file. Also important is the variables for time and task activation. These variables are incremented by the tick interrupt routine when the tasks are active. The reason for this is to provide us with the ability to inspect the amount of time that the tasks have been active, so we can implement the prioritysettask correctly. Thus we implement the main program like this, note here the handlers and timer variables.

```
/* Handlers for the tasks */
TaskHandle_t matrix_handle = NULL;
TaskHandle_t communication_handle = NULL;
TaskHandle_t priority_handle = NULL;

/* Timers for the tasks */
long matrix_ticks = 0;
long comm_ticks = 0;
boolean matrix_active = FALSE;
boolean comm_active = FALSE;

/*-------------------------------------------------------------*/

int main(void)
{
    prvInitialiseHeap();

    vTraceInitTraceData();
    xTickTraceUserEvent = xTraceOpenLabel("tick");

    /* Create the tasks */
    xTaskCreate((pdTASK_CODE)matrix_task, (signed char *)"Matrix", 1000,
        NULL, 3, &matrix_handle);
    xTaskCreate((pdTASK_CODE)communication_task, (signed char *)"Communication",
        configMINIMAL_STACK_SIZE, NULL, 4, &communication_handle);
    xTaskCreate((pdTASK_CODE)prioritysettask, (signed char *)"Priority",
        configMINIMAL_STACK_SIZE, NULL, 5, &priority_handle);

    // This starts the real-time scheduler
    vTaskStartScheduler();

    // Should not reach here
    for ( ;; );

}
```

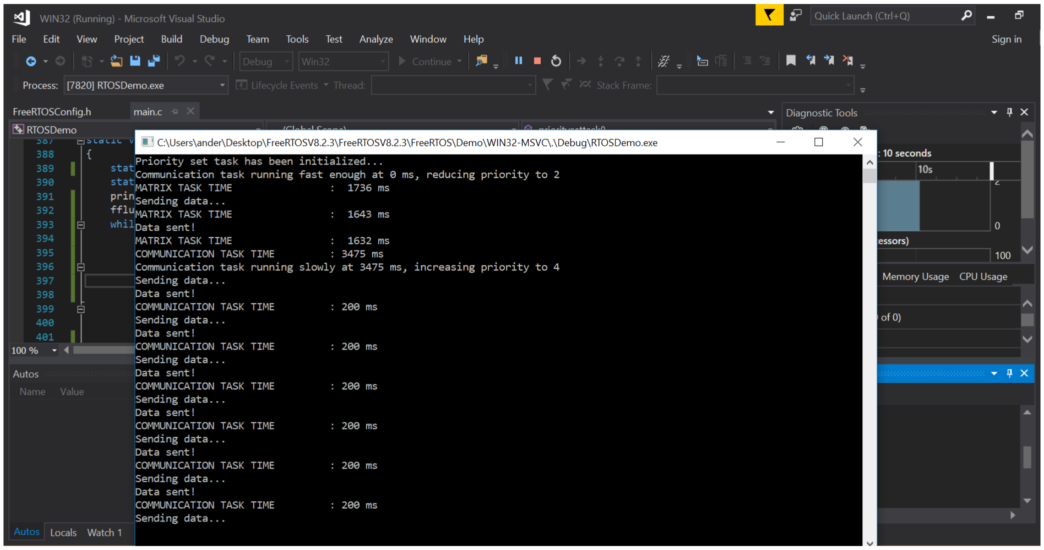
Since we were given the matrixcalculation task and communicationtask, we will simply look here at how we implemented the prioritysettask:

```
static void prioritysettask()
{
    static int comm_pri = 4;    // Using a local variable -> less api calls
    static int comm_time = 0;        // Variable to calc ticks -> time in ms
    printf("Priority set task has been initialized...\n");
    fflush(stdout);
    while (1) {
        // First calculate the communications time in ms
        comm_time = comm_ticks * portTICK_PERIOD_MS;
        // Uncomment for debug purposes
        //printf("PriSetTask getting COMTIME    : %i ms...\n", comm_time);
        //fflush(stdout);
        /* LOGIC : In the execution of this task we check the active ticks
            * for the communication task - if over 1000 ms, we raise priority,
            * while under 200 ms, we lower priority */
        if (comm_time > 1000 && comm_pri != 4) {
            comm_pri = 4;
            printf("Communication task running slowly at %i ms,
                    increasing priority to 4\n", comm_time);
            fflush(stdout);
            vTaskPrioritySet(communication_handle, comm_pri);
        }
        else if (comm_time < 200 && comm_pri != 2) {
            comm_pri = 2;
            printf("Communication task running fast enough at %i ms,
                    reducing priority to 2\n", comm_time);
            fflush(stdout);
            vTaskPrioritySet(communication_handle, comm_pri);
        }
        // -- Block the task for some ticks before we loop --
        vTaskDelay(1000);
    }
}
```

Since the prioritysettask should only run after we have gotten the exectutiontime of communicationtask, the communication task will suspend the prioritysettask when running, and reset the ticks variable.

Results

The resulting output from the program were as follows :



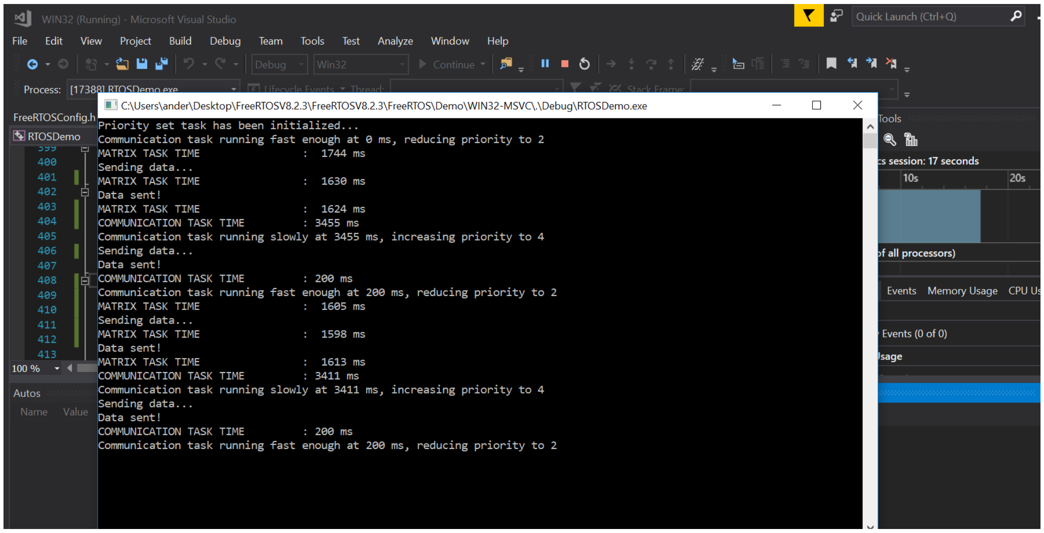
C:\Users\ander\Desktop\FreeRTOSV8.2.3\FreeRTOSV8.2.3\FreeRTOS\Demo\WIN32-MSVC\.\Debug\RTOSDemo.exe

```
Priority set task has been initialized...
Communication task running fast enough at 0 ms, reducing priority to 2
MATRIX TASK TIME                 :  1736 ms
Sending data...
MATRIX TASK TIME                 :  1643 ms
Data sent!
MATRIX TASK TIME                 :  1632 ms
COMMUNICATION TASK TIME          : 3475 ms
Communication task running slowly at 3475 ms, increasing priority to 4
Sending data...
Data sent!
COMMUNICATION TASK TIME          : 200 ms
Sending data...
Data sent!
COMMUNICATION TASK TIME          : 200 ms
Sending data...
Data sent!
COMMUNICATION TASK TIME          : 200 ms
Sending data...
Data sent!
COMMUNICATION TASK TIME          : 200 ms
Sending data...
Data sent!
COMMUNICATION TASK TIME          : 200 ms
Sending data...
Data sent!
COMMUNICATION TASK TIME          : 200 ms
Sending data...
```

Figure 2: Debug output

As seen in the debug output, when the program initializes the exec time of communicationstask is set to 0, so its priority is quickly set to 2. This causes the matrixcalculation task to block the communication task, which causes its time to be over 3000 ms. Therefore after its execution, its priority is increased, which causes it to consecutively execute at a little over 200 ms.

More interestingly though is in the case when I change the code of prioritysettask to change the prirotiy of the communication task if its executiontime is less than 220 ms rather than 200 ms. This causes its priority to be changed every other time it executes. This was not the requirement, just an interesting experiment with real time systems. The following is the output from this scenario:


WIN32 (Running) - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Debug
Win32
Continue
Quick Launch (Ctrl+Q)
Process: [17388] RTOSDemo.exe
Lifecycle Events
Thread:
Stack Frame:
FreeRTOSConfig.h
RTOSDemo
100 %
Autos
Name
Value
Tools
session: 17 seconds
10s
20s
of all processors)
Events
Memory Usage
CPU Us
Events (0 of 0)
Usage
C:\Users\ander\Desktop\FreeRTOSV8.2.3\FreeRTOSV8.2.3\FreeRTOS\Demo\WIN32-MSVC\.\Debug\RTOSDemo.exe

```
Priority set task has been initialized...
Communication task running fast enough at 0 ms, reducing priority to 2
MATRIX TASK TIME                :  1744 ms
Sending data...
MATRIX TASK TIME                :  1630 ms
Data sent!
MATRIX TASK TIME                :  1624 ms
COMMUNICATION TASK TIME         : 3455 ms
Communication task running slowly at 3455 ms, increasing priority to 4
Sending data...
Data sent!
COMMUNICATION TASK TIME         : 200 ms
Communication task running fast enough at 200 ms, reducing priority to 2
MATRIX TASK TIME                :  1605 ms
Sending data...
MATRIX TASK TIME                :  1598 ms
Data sent!
MATRIX TASK TIME                :  1613 ms
COMMUNICATION TASK TIME         : 3411 ms
Communication task running slowly at 3411 ms, increasing priority to 4
Sending data...
Data sent!
COMMUNICATION TASK TIME         : 200 ms
Communication task running fast enough at 200 ms, reducing priority to 2
```

Figure 3: Debug output

Learning outcome

- **Why is ”matrixtask” using most of the CPU utilization?**

  The matrix task is more computationally expensive than the other tasks, and its priority is higher.

- **Why must the priority of ”communicationtask” increase in order for it to work properly**

  The matrix task is blocking the communicationtask from executing due to it having a higher priority, and that it is computatuinally expensive, so it takes a while to execute. Therefore, for communicationtask to run properly, its must increase priority, so it can pre-empt the matrixtask.

- **What happens to the completion time of ”matrixtask” when the priority of ”communicationtask” is increased?**

  There seems to be very little difference, which is expected as commuincationtask consumes very little computational resources

- **How many seconds is the period of ”matrixtask”? (Hint: look at vApplicationTickHook() to measure it)**

Currently the code is measuring executiontime. To measure period, I removed the use of the matrix_active boolean, and just reset the matrix_ticks variable every time the matrixtask ran. The code at the beginning of the matrix task thus looks like this:

```
printf("MATRIX TASK TIME  :  %i ms \n", matrix_ticks * portTICK_PERIOD_MS);
fflush(stdout);
matrix_ticks = 0;
```

After this we get a period at about 1.7 sec.