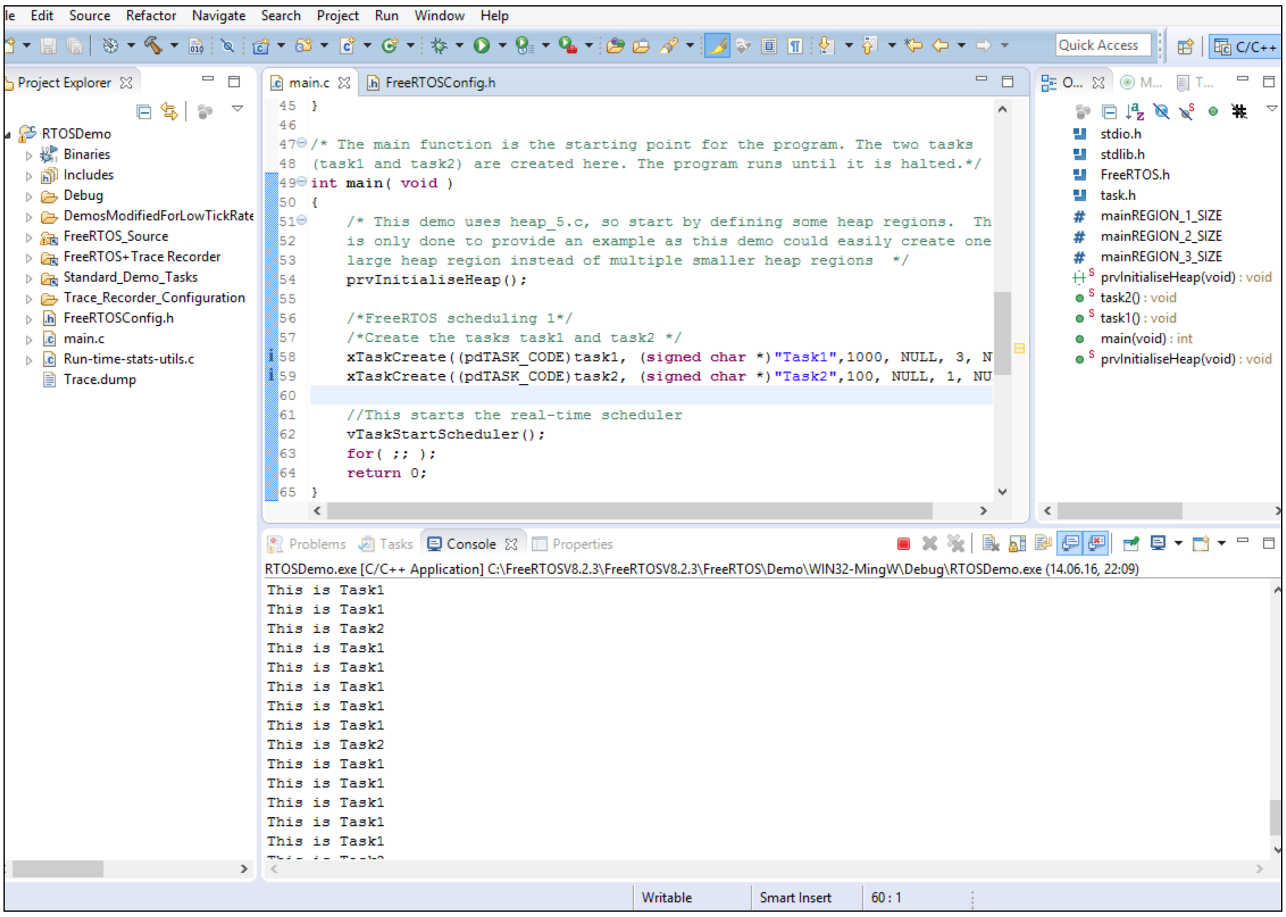
Development of Real-Time Systems

Assignment 1

Elyasin Shaladi - 14 June 2016



```
}

/* The main function is the starting point for the program. The two tasks
(task1 and task2) are created here. The program runs until it is halted.*/
int main( void )
{
    /* This demo uses heap_5.c, so start by defining some heap regions.  Th
    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 tasks task1 and task2 */
    xTaskCreate((pdTASK_CODE)task1, (signed char *)"Task1",1000, NULL, 3, N
    xTaskCreate((pdTASK_CODE)task2, (signed char *)"Task2",100, NULL, 1, NU

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

```
RTOSDemo.exe [C/C++ Application] C:\FreeRTOSV8.2.3\FreeRTOSV8.2.3\FreeRTOS\Demo\WIN32-MingW\Debug\RTOSDemo.exe (14.06.16, 22:09)
This is Task1
This is Task1
This is Task2
This is Task1
This is Task1
This is Task1
This is Task1
This is Task1
This is Task2
This is Task1
This is Task1
This is Task1
This is Task1
This is Task1
```

Assignment 1 Report

After installation of the Eclipse IDE for C/C++ developers I imported the FreeRTOS Demo project as suggested by the course instructions. I could successfully run it and so the stage for the first assignment was set.

I located the two functions *task1* and *task2* in the *main.c* file. They correspond to the instruction's specifications. I modified the parameters of the *vTaskDelay(…)* function in order to ensure that the respective tasks are blocked for 500ms and 100ms.

```
// task2 printing to the standard output console
// and waiting 500 ms
static void task2(){
  while(1){
            printf("This is Task2\n");
            fflush( stdout );
            vTaskDelay(500 / portTICK_PERIOD_MS);
  }
}

// task1 printing to the standard output console
// and waiting 100 ms
static void task1(){
  while(1){
            printf("This is Task1\n");
            fflush( stdout );
            vTaskDelay(100 / portTICK_PERIOD_MS);
  }
}
```

The next step was to create the tasks at RTOS program start. This is done in the *main(void)* function. I located the *main* function and made sure the tasks were created before the real-time scheduler is started.

```
/*FreeRTOS scheduling 1*/
xTaskCreate((pdTASK_CODE) task1, (signed char *)"Task1",1000, NULL, 3, &task1handle);
xTaskCreate((pdTASK_CODE) task2, (signed char *)"Task2",100, NULL, 1, &task2handle);
```

The tasks are created with the parameters according to the instructions given in the assignment. task1 is named *Task1*, has a stack size of *1000* and priority of *3*, and task2 is named *Task2*, has a stack size of *100* and priority of *1*. I removed most of the demo code as it is not used for this specific assignment to show that I studied the FreeRTOSConfig.h file a little bit and adopted it accordingly. The only piece of code from the demo that I kept for this assignment was the memory management. I used the heap_5.c file as it was delivered for simplicity.

In the screen shot on the first page of this report you can see an example. The output of the respective tasks are printed in the console.