

# Lab 9

November 12, 2021

Delivery date: November 12th, 2021

## **FreeRTOS Threads and Queues**

*Javier Mondragon Martin del Campo*

*A01365137*

Prof. Matías Vázquez Piñón  
Tecnológico de Monterrey

# 1 Introduction

During the practice, execution routines with different tasks and priorities were developed and executed with an operating system developed for microprocessors called "FreeRTOS". The operating system allows to have different tasks with infinite cycles that are executed "at the same time" (executing a task every certain time simulating the instantaneous execution of each one). Each task has its own execution time and executes its own defined instructions.

The first program prints out hello world with a counter for every time the message is printed.

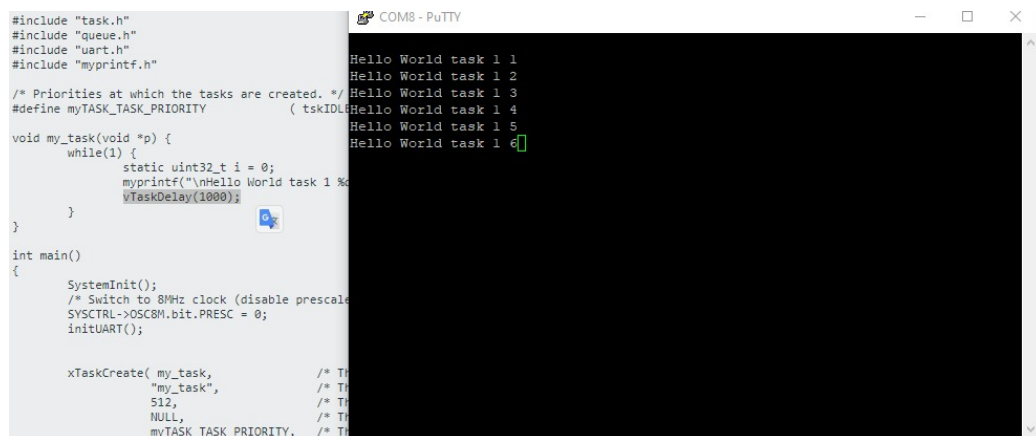
The second program is a demonstration of its operation showing messages every certain time and in case the receiver task (in charge of showing the output of a task), does not receive the message, it shows an error message.

In the case of the third program, the tasks are divided in evaluating each port and in case of detecting that the port has an input of 0, a message is displayed.

## 2 Results

### 2.1 Part I

Screenshot of properly-displayed messages from main.c on the serial port Terminal (PuTTY):



```
#include "task.h"
#include "queue.h"
#include "uart.h"
#include "myprintf.h"

/* Priorities at which the tasks are created. */
#define myTASK_TASK_PRIORITY (tskIDLE_PRIORITY + 1)

void my_task(void *p) {
    while(1) {
        static uint32_t i = 0;
        myprintf("\nHello World task 1 %d", i);
        vTaskDelay(1000);
        i++;
    }
}

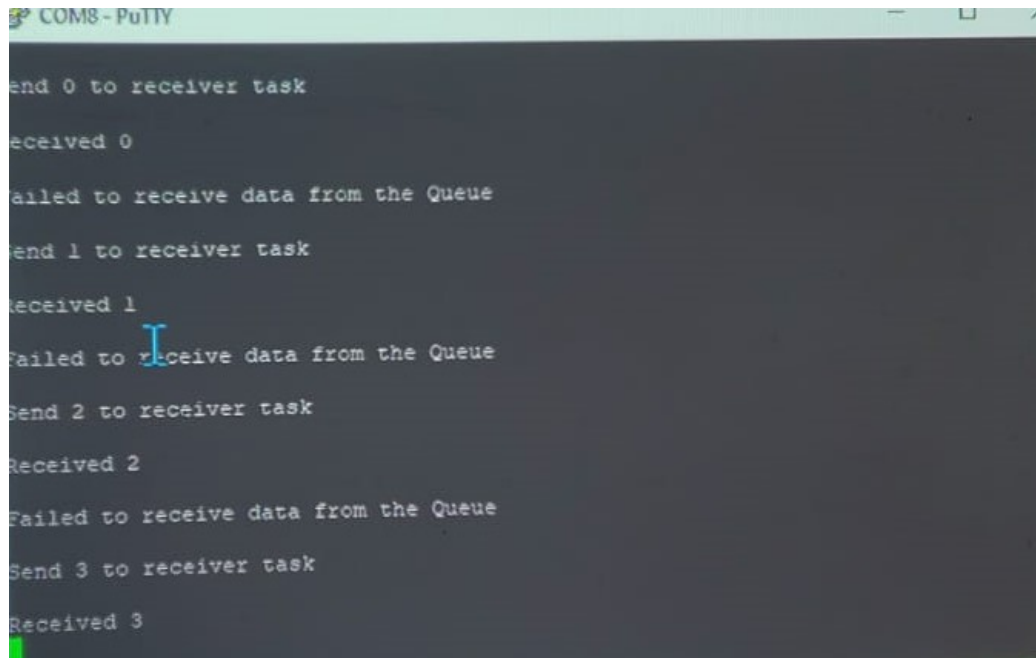
int main()
{
    SystemInit();
    /* Switch to 8MHz clock (disable prescaler) */
    SYSCTRL->OSCBM.bit.PRESC = 0;
    initUART();

    xTaskCreate( my_task, /* Task function */
                "my_task", /* Task name */
                512, /* Task stack size */
                NULL, /* Task parameter */
                myTASK_TASK_PRIORITY, /* Task priority */
                NULL); /* Task handle */
}
```

Figure 1: Properly-displayed messages

## 2.2 Part II

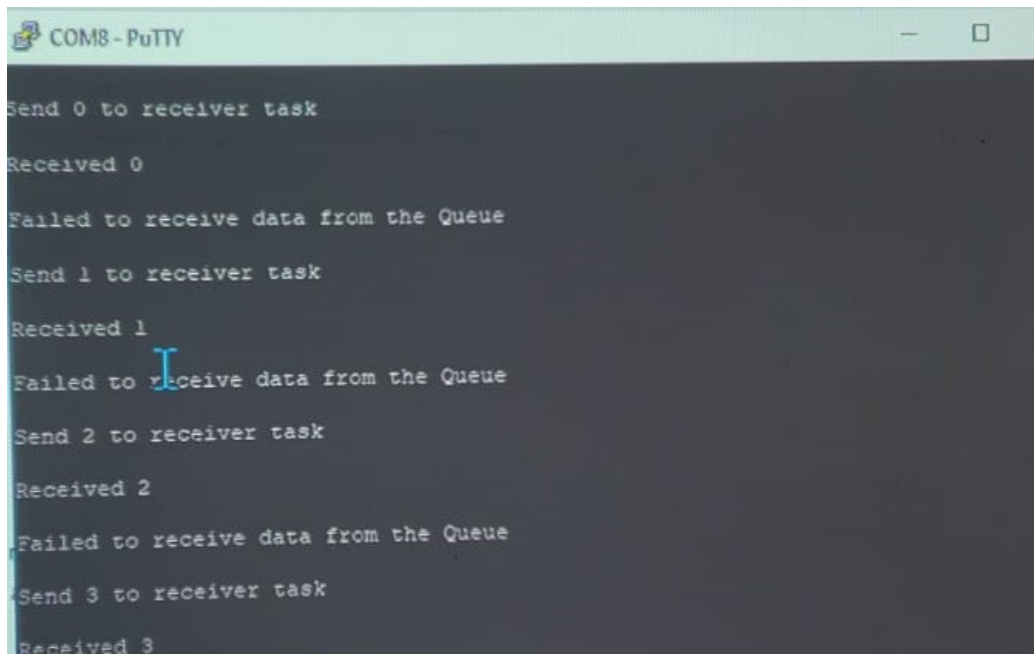
Screenshots of message on the Serial Port terminal:



```
COM8-PuTTY

Send 0 to receiver task
Received 0
Failed to receive data from the Queue
Send 1 to receiver task
Received 1
Failed to receive data from the Queue
Send 2 to receiver task
Received 2
Failed to receive data from the Queue
Send 3 to receiver task
Received 3
```

Figure 2: Send number to receiver task



```
COM8-PuTTY

Send 0 to receiver task
Received 0
Failed to receive data from the Queue
Send 1 to receiver task
Received 1
Failed to receive data from the Queue
Send 2 to receiver task
Received 2
Failed to receive data from the Queue
Send 3 to receiver task
Received 3
```

Figure 3: Send changing number to receiver task

## 2.3 Part III

Screenshot of message on the serial port Terminal showing the pressed buttons:

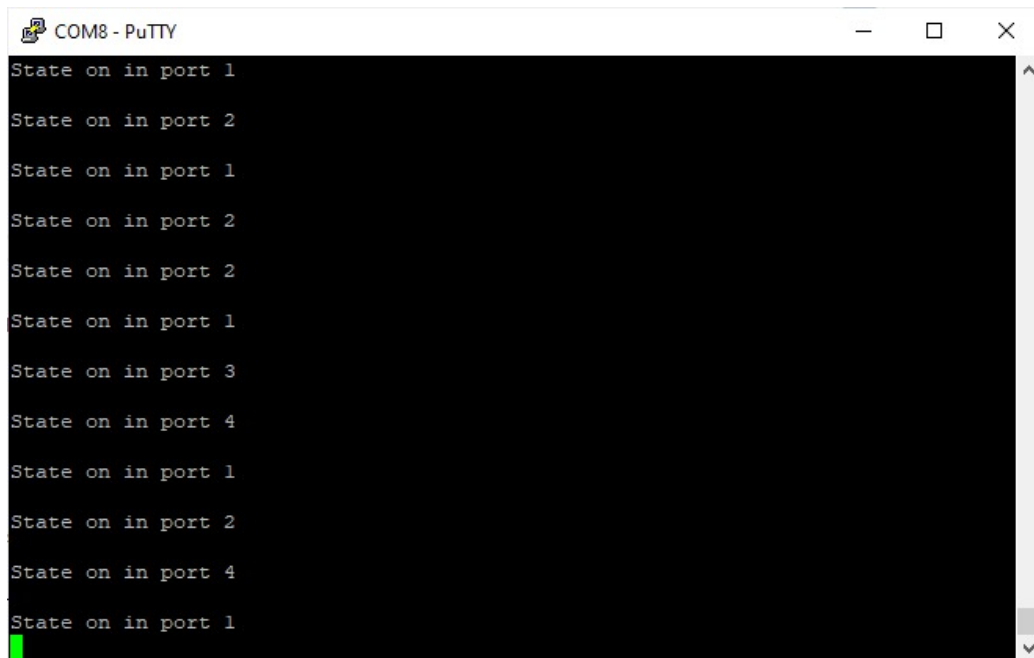


Figure 4: Switch state on terminal

Picture of the working hardware:

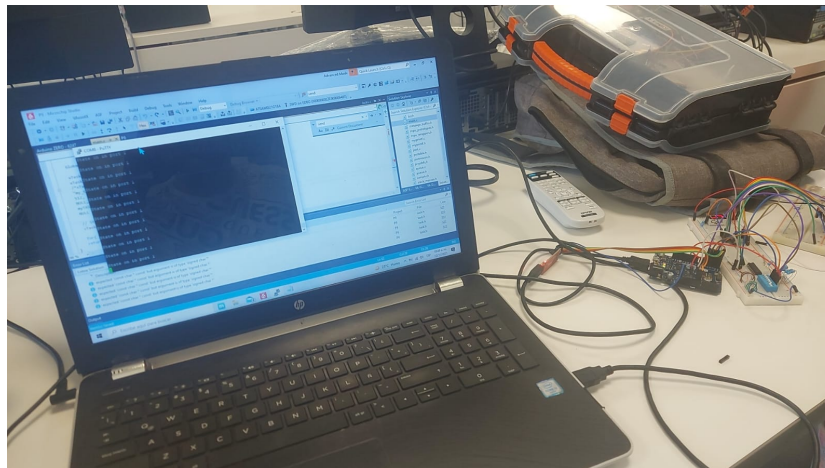


Figure 5: Working microcontroller

The message "Failed to receive data from queue" was displayed because the receiver task received nothing (cero) when the routine xQueueReceive received

no data or acknowledge from the Global\_Queue\_Handle because no task were executed.

### **3 Conclusions**

During the lab we worked together and view that a operating system could be implemented into a microcontroller and with a single core like an AT-MEGA. The applications with this technology could be using the same microprocessor to do different tasks that require to read multiple registers at once or proccess multiple instruction sets at once. The errors where to synchronize the task to run at certain time, these errors were fixed with different delays.

### **4 Bibliography**

1. <https://github.com/matias-vazquez/SistemasEmbebidos>
2. [https://github.com/javiermomc/Sistemas\\_Embebidos/tree/main/L09](https://github.com/javiermomc/Sistemas_Embebidos/tree/main/L09)