

# Lab: FreeRTOS Tasks

## Objective

1. Load firmware onto the SJ board
2. Observe the RTOS round-robin scheduler in effect
3. Provide hands-on experience with the UART character output timing

## Part 0a. Change UART speed

We will be working with an assumption for this lab, so we will need to change the UART speed. In Visual Studio Code IDE, hit `Ctrl+P` and open `peripherals_init.c`. Then modify the UART speed to 38400. After doing so, make sure you open your serial terminal or Telemetry web terminal and change the port speed to also 38400.

```
static void peripherals_init_uart0_init(void) {
    // Do not do any buffering for standard input otherwise getchar(), scanf() may not work
    setvbuf(stdin, 0, _IONBF, 0);
    // Note: PIN functions are initialized by board_io_initialize() for P0.2(Tx) and P0.3(Rx)
    uart_init(UART_0, clock_get_peripheral_clock_hz(), 38400); // CHANGE FROM 115200 to 38400

    // ...}
```

The `peripherals_init_uart0_init()` is executed before your `main()` function. When you are finished with this lab, you can choose to change this back to 115200bps for faster UART speed.

## Part 0b. Create Task Skeleton

A task in an RTOS or FreeRTOS is nothing but a forever loop, however unless you sleep the task, it will consume 100% of the CPU. For this part, study existing `main.c` and create two additional tasks for yourself.

```
#include "FreeRTOS.h"
#include "task.h"
static void task_one(void * task_parameter);
static void task_two(void * task_parameter);
int main(void) {
```

```

// ...
}
static void task_one(void * task_parameter) {
    while (true) {
        // Read existing main.c regarding when we should use fprintf(stderr...) in place of printf()
        // For this lab, we will use fprintf(stderr, ...)
        fprintf(stderr, "AAAAAAAAAAAA");

        // Sleep for 100ms
        vTaskDelay(100);
    }
}
static void task_two(void * task_parameter) {
    while (true) {
        fprintf(stderr, "bbbbbbbbbbbb");
        vTaskDelay(100);
    }
}
}
}

```

## Part 1: Create RTOS tasks

1. Fill out the `xTaskCreate()` method parameters.
  - See the FreeRTOS+Tasks document or checkout the [FreeRTOS xTaskCreate API website](#)
  - Recommended stack size is: `4096 / sizeof(void*)`
2. Note that you want to make sure you use `fprintf(stderr, ...)` in place of `printf(...)`
  - `fprintf(stderr, ...)` is slower and eats up CPU, but it is useful during debugging
  - `printf(...)` is faster (and efficient), but it queues the data to be "sent later"
3. Observe the output
  - After you flash your program, check the output of the serial console

```

#include "FreeRTOS.h"
#include "task.h"
static void task_one(void * task_parameter);
static void task_two(void * task_parameter);
int main(void) {
    /**
     * Observe and explain the following scenarios:
     *

```

```

* 1) Same Priority:      task_one = 1, task_two = 1
* 2) Different Priority: task_one = 2, task_two = 1
* 3) Different Priority: task_one = 1, task_two = 2
*
* Note: Priority levels are defined at FreeRTOSConfig.h
* Higher number = higher priority
*
* Turn in screen shots of what you observed
* as well as an explanation of what you observed
*/
xTaskCreate(task_one, /* Fill in the rest parameters for this task */ );
xTaskCreate(task_two, /* Fill in the rest parameters for this task */ );
/* Start Scheduler - This will not return, and your tasks will start to run their while(1) loop */
vTaskStartScheduler();
return 0;
}
// ...

```

## Part 2: Further Observations

Fundamentals to keep in mind:

- FreeRTOS tick rate is configured at 1Khz
  - This means that the RTOS preemptive scheduling can occur every 1ms repetitively
- Standout output (`printf`) is integrated in software to send data to your UART0
  - This is the same serial bus that is used to load a new program (or hex file)
  - The speed is defaulted to 38400bps, and since there is 10 bits of data used to send 1 byte, we can send as many as 3840 characters per second

Critical thinking questions:

- How come 4(or 3 sometimes) characters are printed from each task? Why not 2 or 5, or 6?
- Alter the priority of one of the tasks, and note down the observations. Note down WHAT you see and WHY.

**Hint:** You have to relate the speed of the RTOS round-robin scheduler with the speed of the UART to answer the questions above

?

## Part 3. Change the priority levels

Now that you have the code running with identical priority levels, try the following:

1. Change the priority of the two tasks

\* Same Priority: `task_one = 1, task_two = 1`

\* Different Priority: `task_one = 2, task_two = 1`

\* Different Priority: `task_one = 1, task_two = 2`

2. Take a screenshot of what you see from the console

3. Write an explanation of why you think the output came out the way it did using your knowledge about RTOS

Optional: If you have Tracealyzer program installed, we encourage you to [load this file](#) and inspect the trace.

---

## What to turn in:

1. Relevant code
2. Your observation and explanation
3. Snapshot of the output for all scenarios

If your class requires you to turn in the assignment as a Gitlab link, you should:

- Use [this article](#) to get started
  - Submit a link to Gitlab "Merge Request"
  - Be sure to ensure that your Merge Request is only the new code, and not a very large diff
- ?

---

Revision #11

Created 6 years ago by [Preet Kang](#)

Updated 3 years ago by [Preet Kang](#)