LAB 01: FREERTOS TASKS

Consider 2 tasks: *task_one* and *task_two*

Each task is assigned a priority from 0 to (configMAX_PRIORITIES – 1), where configMAX_PRIORITIES is defined within FreeRTOSConfig.h. Low priority numbers denote low priority tasks. The idle task has priority zero.

The FreeRTOS scheduler ensures that tasks in the Ready or Running state will always be given processor (CPU) time in preference to tasks of a lower priority that are also in the ready state. In other words, the task placed into the Running state is always the highest priority task that can run.

Part A: Unequal Priority

When ***task_one*** has a higher priority than ***task_two,*** the output obtained is as shown in figure1.

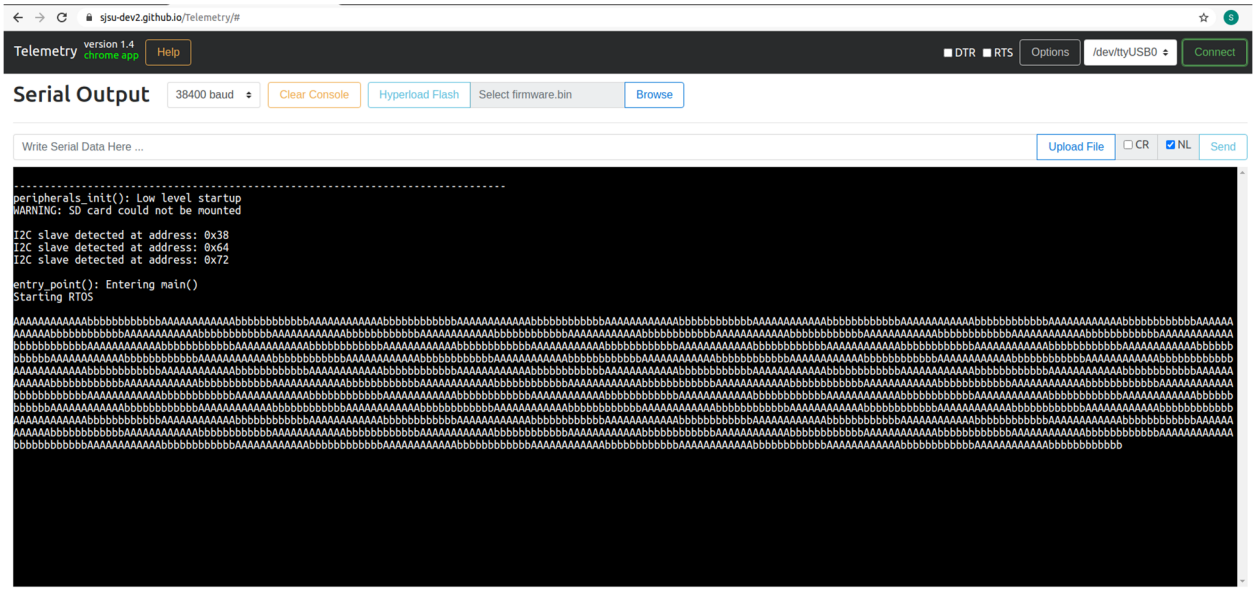

Figure1

As seen clearly from the figure since task_one has higher priority than task_two, it will move into the Running state and will be executed. Untill then task_two will be in the Ready state. Once task_one has finished executing the scheduler will move task_two to running state for execution and task_one is moved to Blocked state. The slice time is 1 ms and sleep duration given to both tasks is 100ms. After task_one executes and goes to sleep, the next 100ms is given to task_two to execute before it can be pre-empted and task_one needs to be executed again. During the sleep period the tasks would be placed in Blocked state. After 100ms, task_one is moved to the Ready state as it is ready for execution. Since it is a high priority task task_two will be preempted. Figure shown below is the sequence of execution.

task_one	task_two	task_one	task_two	task_one	task_two	task_one	task_two	task_one

Part B: Unequal Priority

When ***task_two*** has a higher priority than ***task_one*** the output obtained is as shown in figure2.

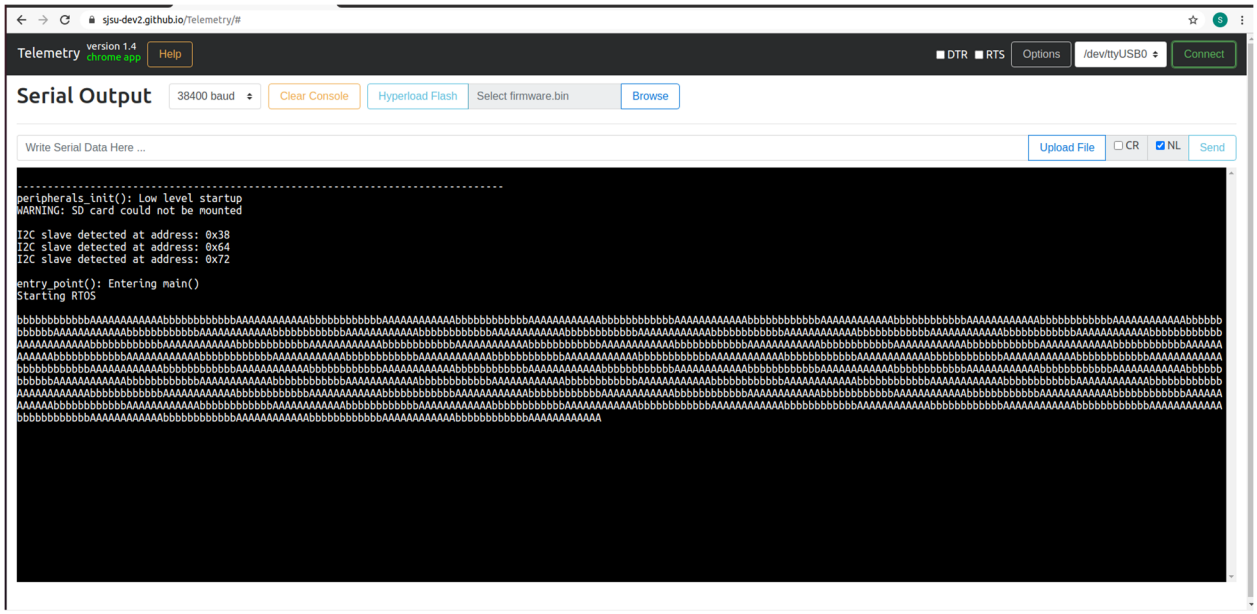

Figure2

As seen clearly from the figure since task_two has higher priority than task_one, it will move into the Running state and will be executed. Untill then task_one will be in the Ready state. Once task_two has finished executing the scheduler will move task_one to running state for execution and task_two is moved to Blocked state or Ready state depending if it needs to be executed again or it is in delay state. The slice time is 1 ms and sleep duration given to both tasks is 100ms. After task_two executes and goes to sleep, the next 100ms is given to task_one to execute before it can be pre-empted and task_two needs to be executed again. During the sleep period the tasks would be placed in Blocked state. After 100ms, task_two is moved to the Ready state as it is ready for execution. Since it is a high priority task task_one will be preempted.

task_two	task_one	task_two	task_one	task_two	task_one	task_two	task_one	task_two

The above diagram shows the sequence of execution in this case.

Part C: Equal Priority

We know that as per FreeRTOS scheduler the task placed into the Running state is always the highest priority task that can run.

In this scenario, both the tasks have equal priority. So ideally both the tasks would be running depending on whichever task is moved to running state. The slice time is 1ms and so both the tasks will have to run within 1ms.

As seen from the figure, both the tasks are running and a combination of only 3 or 4 characters is printed from each task. This happens because the UART speed is 38400. Hence as per the calculations mentioned below a total of 3.84 characters can be printed every msec from each task. Delays from both the tasks are removed to avoid any of the task moving to a block state.

UART Speed: 38400 bps

FreeRTOS Tick Rate: 1KHz

Slice Time: 1ms

Number of characters (in secs): 38400/10 = 3840 characters/sec

Number of characters (in msec): 3840*1msec / 1sec = 3.84 characters /msec

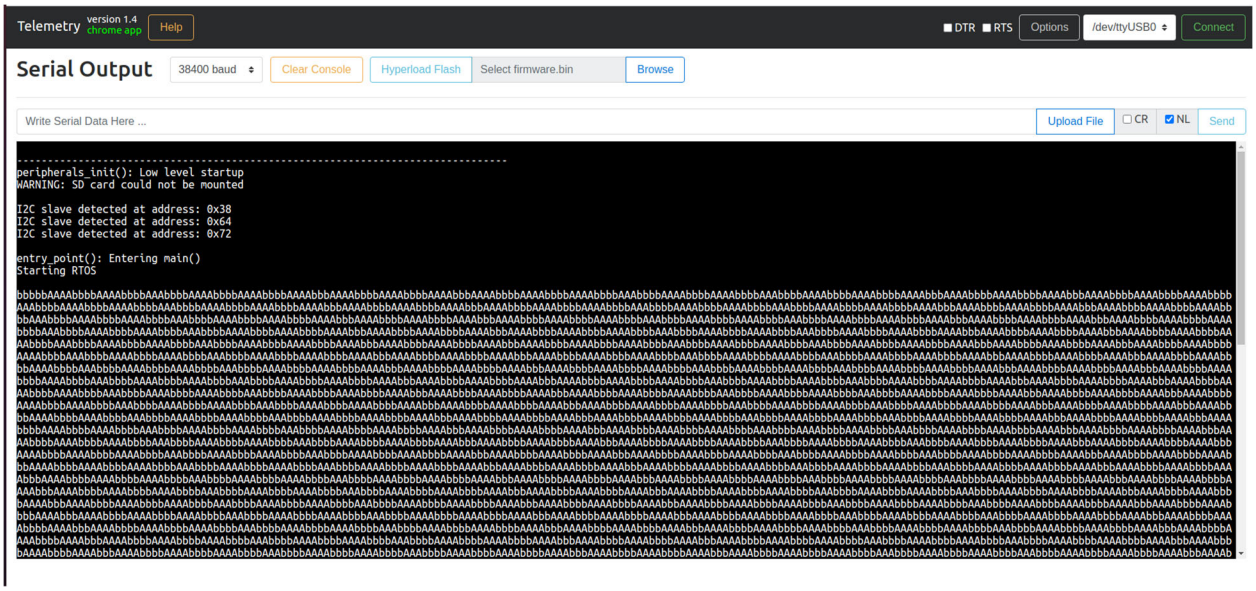

Figure3

Since both the task have equal priority, the scheduler can execute any of the two tasks depending on which task is in the ready state. As seen from the figure character b is printed which tells us that task_two is moved to the running state initially. But at the beginning of the print it is observed that there are 5 b's printed but after that there has always been a combination of 3 or 4 characters from every task. The reason for printing only 5 b's at the start is because task_one might not have been in the ready state. The

moment task_one is moved to ready state a switch occurs but by then task_two would have executed and hence got a chance to print 5 characters.