Assignment 2

```
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "FreeRTOS.h"
#include "task.h"
/* Private includes ----------------------------------------------------------*/
```

The libraries which are included in the main.c file are basically main.h, FreeRTOS.h and task.h.

Creation of a Task:

Task functionality can be written in the function which can be defined as below:

```
void Task_Name(void * Parameter)
{
	//init code
	for(;;)
	{
		//infinite for loop or main task functionality
	}
}
```

In the above code, task name could be given to the task, which also takes a parameter, the code before the for loop is called the init or initialization code which only runs once, the main code functionality is written in the for loop.
From the assignment code, the function definitions are mentioned below:

```
void Task1(void *temp) {
	for(;;)
	{
		HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_12 | GPIO_PIN_13);
		HAL_Delay(4);
	}
}

void Task2(void *temp) {
	for(;;)
	{
		HAL_GPIO_TogglePin(GPIOD, GPIO_PIN_14 | GPIO_PIN_15);
		HAL_Delay(5);
	}
}
```

After initializing all the configured peripherals, its time to create the tasks.

Tasks are created using the xTaskCreate() API call.

```
xTaskCreate(pxTaskCode, pcName, usStackDepth, pvParameters, uxPriority, pxCreatedTask);
```

pvTaskCode: Pointer to the task entry function

pcName: A descriptive name for the task. This is mainly used to facilitate debugging, but can also be used to obtain a task handle.

usStackDepth: The number of words (not bytes) to allocate for use as the task's stack. For example, if the stack is 16-bits wide and usStackDepth is 100, then 200 bytes will be allocated for use as the task's stack.

pvParameters: A value that is passed as the paramater to the created task.

uxPriority: The priority at which the created task will execute.

pxCreatedTask: Used to pass a handle to the created task out of the xTaskCreate() function. pxCreatedTask is optional and can be set to NULL.

eg: From the assignment.

```
xTaskCreate(Task1, "Task1", 200, NULL, 1, NULL);
xTaskCreate(Task2, "Task2", 200, NULL, 2, NULL);
```

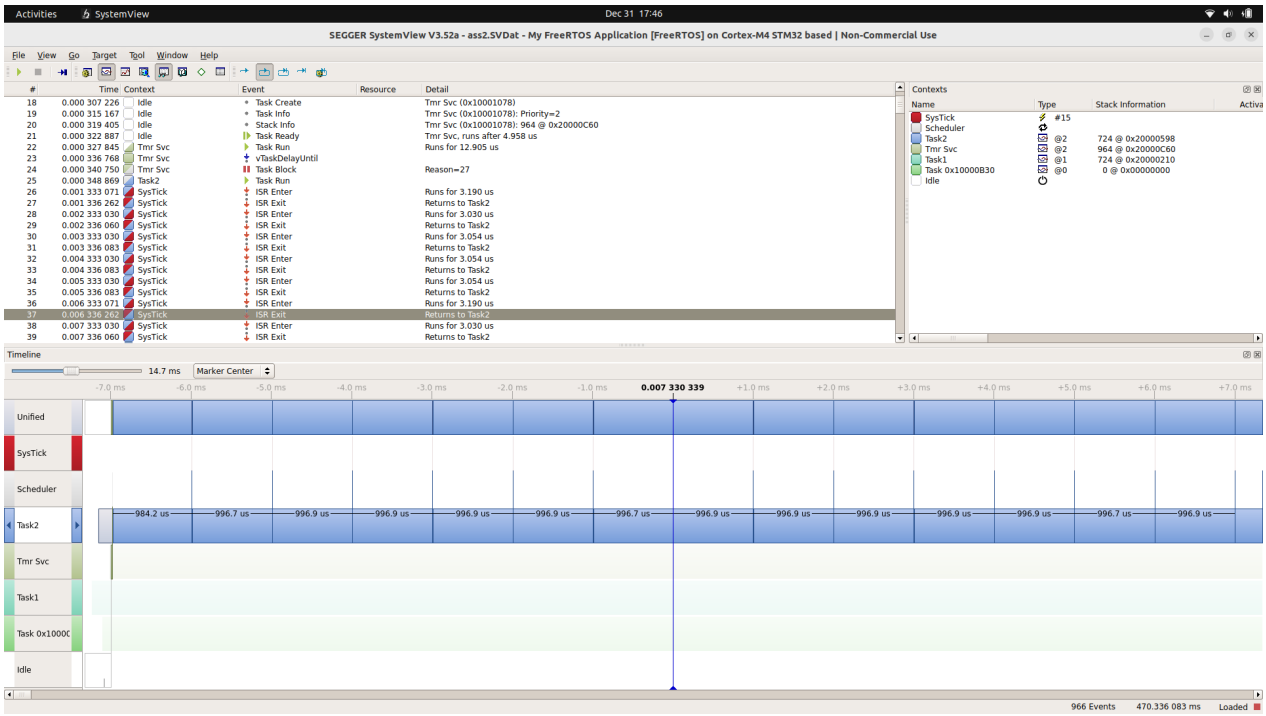
Start Scheduler:

After creating the tasks, the main step is to start the task scheduler, which is done by the API vTaskStartScheduler().

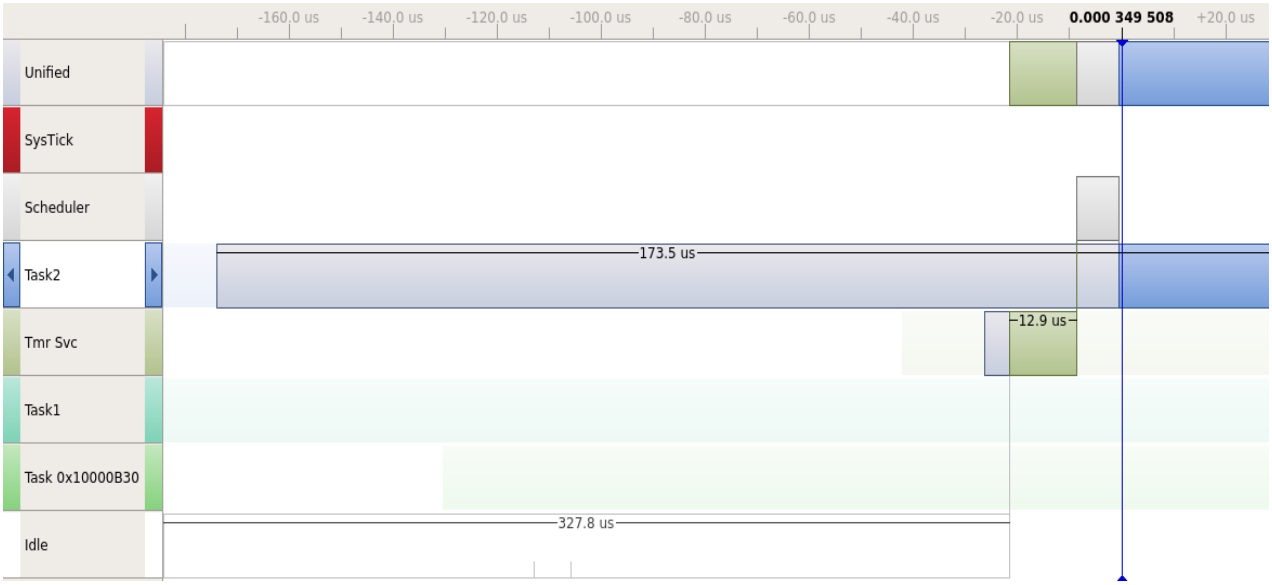
```
xTaskCreate(Task1, "Task1", 200, NULL, 1, NULL);
xTaskCreate(Task2, "Task2", 200, NULL, 2, NULL);

vTaskStartScheduler();
```

While observing the trace of the code, on the SEGGER SYSTEM VIEWER, we see something like this.

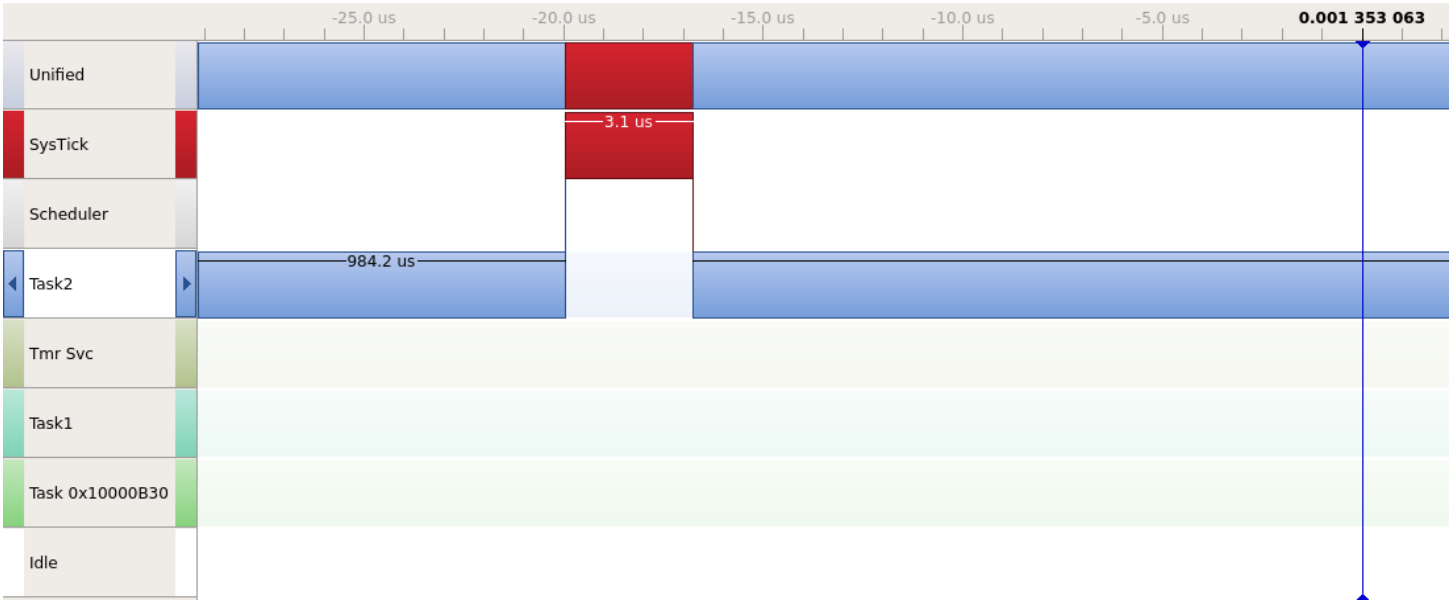


If we observe the trace carefully, we see that, the first task is always scheduled by the SVC. i.e Supervisory Call



While we observe, we can see that, the task 1 and task 2 were created and were in Ready State.

As the priority of Task 2 was higher, the Supervisory Call invokes the scheduler, and we can see that the task 2 enters the running state, and the task 2 is executing.



After a tick, we can see that systick is invoked and it looks for a task for higher priority, as the default scheduling in FreeRTOS is Priority-Based Pre-Emptive Scheduling.

As the systick doesn't find doesn't find any task with higher priority, it doesn't invoke the scheduler and thus it can be said that systick acts as a tick counter and PendSV acts as the Scheduler in ARM Cortex M4. (as we are working on ARM Cortex M4). It varies on the architecture, if we consider other architectures.