

RTOS Development – 23EC361

Student Name: DIKSHANT MAHAJAN

Roll Number: 2210993527

Class: BE-ECE X1 (4th Sem)

23EC-361: RTOS Development**Assignment 2****Introduction to RTOS**

A real-time operating system (RTOS) is an OS that guarantees real-time applications a certain capability within a specified deadline. RTOS's uses tasks, semaphores and mutex as the tool to do inter-task communication during run time for designing of critical system using RTOS. This signalling scheme offers multiple ways to interact with system resources to share data among various tasks.

Software & Hardware Required

Microsoft Word, STM Cube IDE, STM32L446RG Nucleo board

Pre-requisites: You must do the following before this task.

1. Go through the lecture slides of week 1 to 4.
2. Read the contents of chapter 1 to 3 from reference book 1. (Hands-on RTOS with Microcontroller by Brian Amos, Packt Publishing, First Edition, 2020.)

Objective

The objective of this task is to understand the fundamental concepts of Tasks in RTOS development. Create three tasks (Task 1, Task 2, Task 3) and assign them different priority levels to implement task scheduling algorithm in RTOS development.

Submission Details

1. Prepare a step-by-step tutorial to design and develop RTOS for controlling three tasks (Task 1, Task 2, and Task 3) using STM32 development board.
2. What is the effect of different priority levels on the tasks during the execution of three tasks using RTOS?
3. Write down a reflection summary in 150 words, describing how RTOS has improved the performance of controlling three tasks in a critical system development environment.

RTOS Development – 23EC361

ANS 1 :

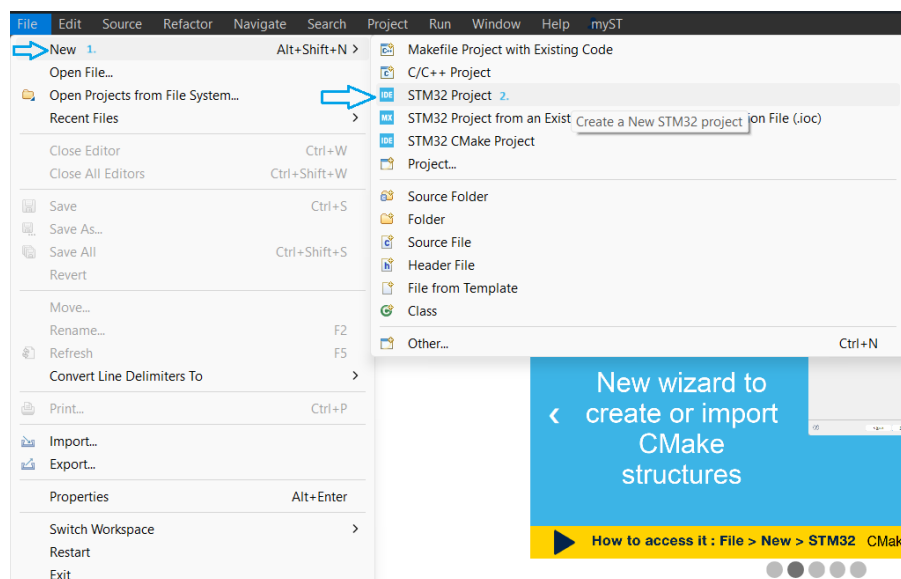
Step-1 – The first step is to download and install the **STM 32 Cube IDE** on your respective computers or laptops from their own website. Link for downloading the software is : <https://www.st.com/en/development-tools/stm32cubeide.html#get-software>

Get Software

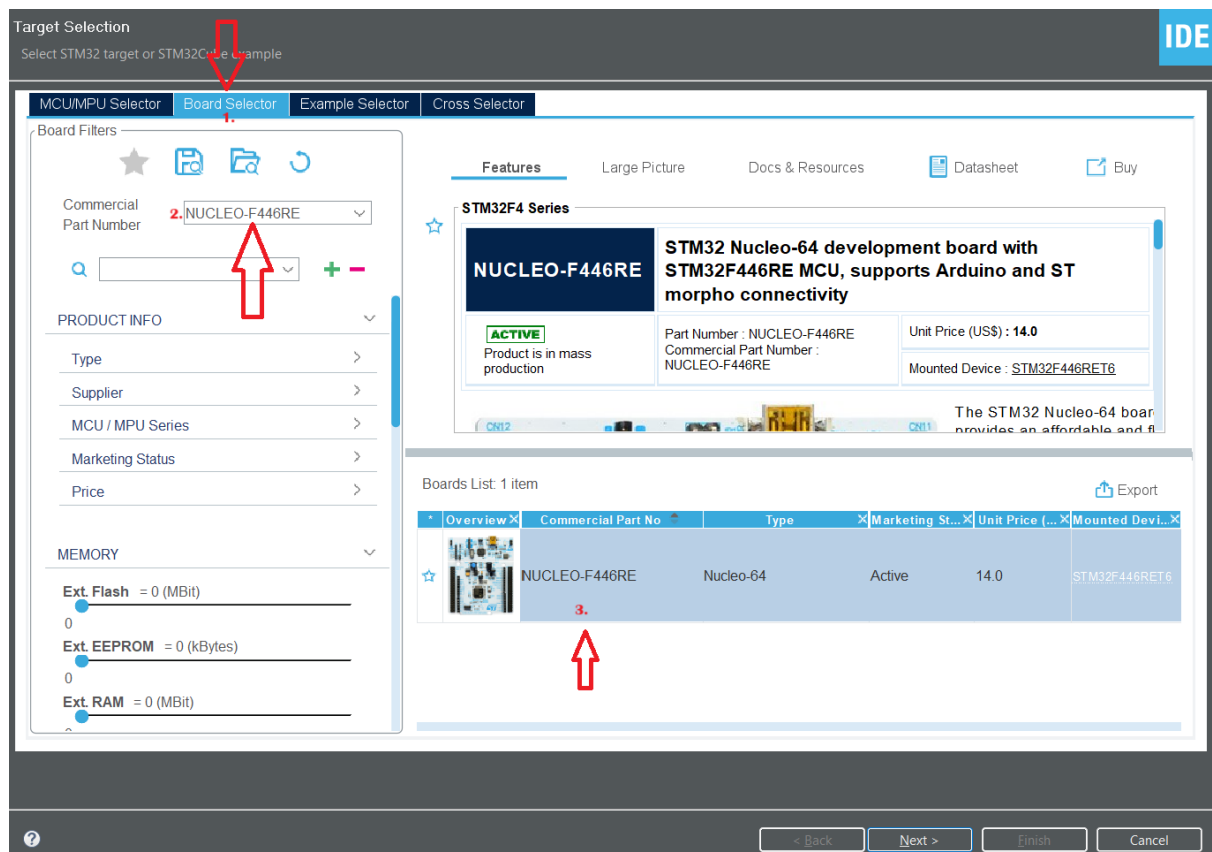
Part Number	General Description	Latest version	Download	All versions
STM32CubeIDE-DEB	STM32CubeIDE Debian Linux Installer	1.14.0	Get latest	Select version ▾
STM32CubeIDE-Lnx	STM32CubeIDE Generic Linux Installer	1.14.0	Get latest	Select version ▾
STM32CubeIDE-Mac	STM32CubeIDE macOS Installer	1.14.0	Get latest	Select version ▾
STM32CubeIDE-RPM	STM32CubeIDE RPM Linux Installer	1.14.0	Get latest	Select version ▾
STM32CubeIDE-Win	STM32CubeIDE Windows Installer	1.14.0	Get latest	Select version ▾

Tap the **Get Latest** button as shown and now a zip file will start downloading once done extract that file and from the extracted folder you'll see the **setup.exe** file install that file and finally your software will be installed on your device.

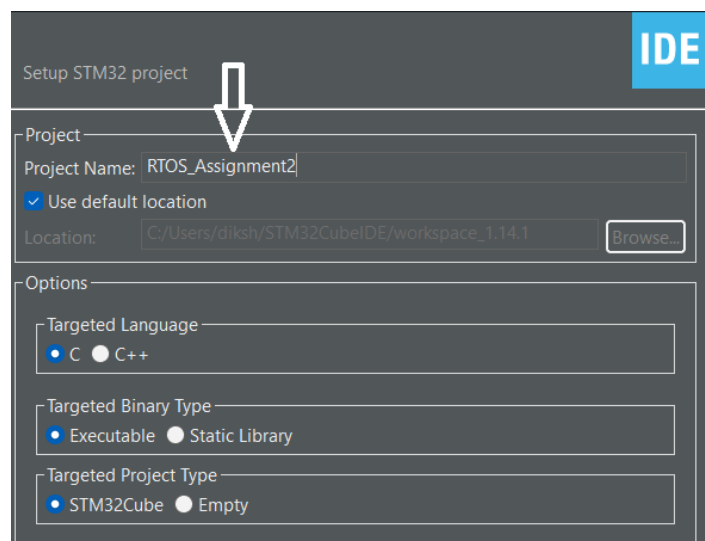
Step-2 – Now Open the STM32 Cube IDE. Go to **File > New > STM32 Project**



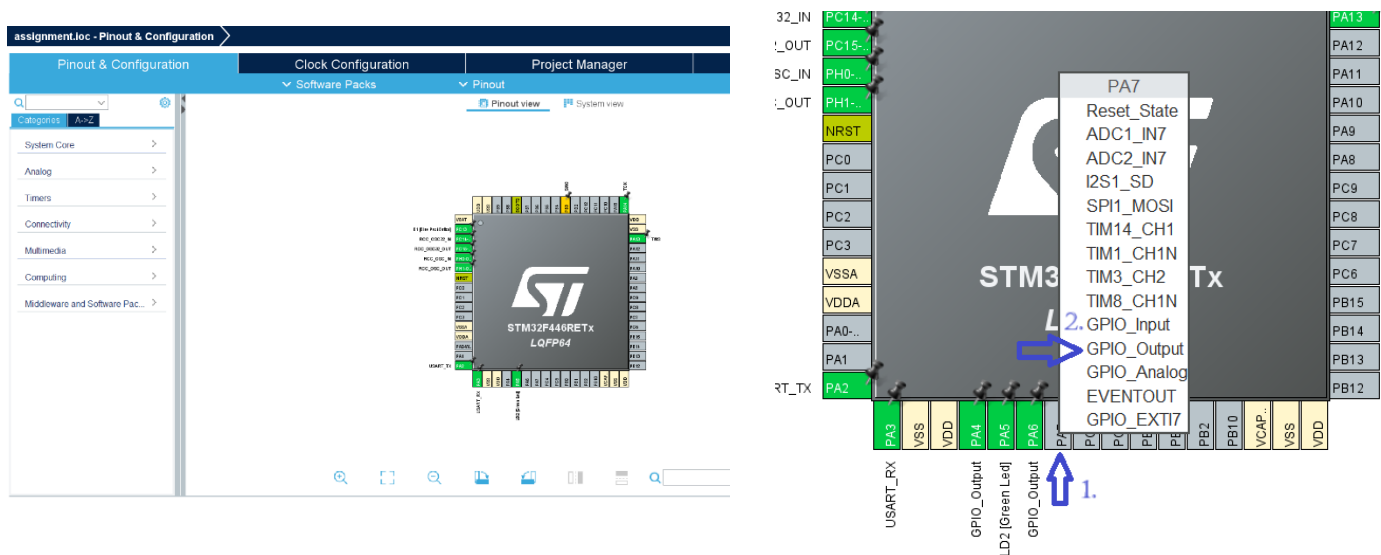
Step-3 – Then a **Target Selection** window appears. In this window First click on **Board Selector**. In the **commercial Part Number** box search : **NUCLEO-F446RE** and select that board.



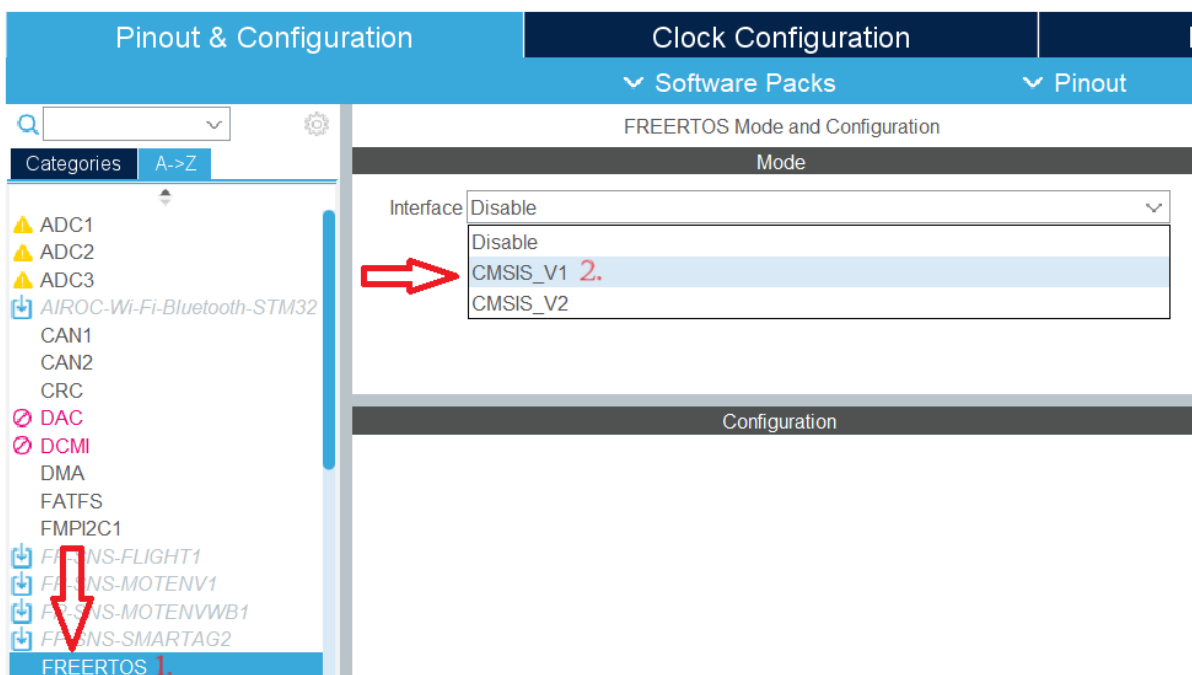
Step-4 – After clicking on **Next** in the previous step, A **setup STM32 Project** window pops up. In that window give a **Name** to your project and click on next.



Step-5 – Now, the **Pinout & configuration window** appears. Select the **Output pins** on which we are going to get the output. I am taking **O/P as PA4, PA6 & PA7**.



Step-6 – Now **tap on A-Z option**, from there scroll down and you can see a **FREERTOS** option, select it and a sub menu will open. In the interface select **CMSIS_V1**.



RTOS Development – 23EC361

Step-7 – Now below the interface menu you can see **configuration tab**, there click on **Tasks and Queues**. Now click on **add button**, here you can add the number of tasks that you need with different priority levels. In our case 3 tasks are created with following priorities :

Task1 – osPriorityLow ; Task2 – osPriorityNormal ; Task3 – osPriorityHigh

Tasks and Queues Configuration

Task Na...	Priority	Stack Si...	Entry Fu...	Code G...	Parameter	Allocation	Buffer N...	Control ...
Task1	osPriorit...	128	StartTas...	Default	NULL	Dynamic	NULL	NULL
Task2	osPriorit...	128	StartTas...	Default	NULL	Dynamic	NULL	NULL
Task3	osPriorit...	128	StartTas...	Default	NULL	Dynamic	NULL	NULL

Edit Task (Task1)

- Task Name: Task1
- Priority: osPriorityLow
- Stack Size (Words): 128
- Entry Function: StartTask1
- Code Generation Option: Default
- Parameter: NULL
- Allocation: Dynamic
- Buffer Name: NULL
- Control Block Name: NULL

New Task (Task2)

- Task Name: Task2
- Priority: osPriorityNormal
- Stack Size (Words): 128
- Entry Function: StartTask02
- Code Generation Option: Default
- Parameter: NULL
- Allocation: Dynamic
- Buffer Name: NULL
- Control Block Name: NULL

New Task (Task3)

- Task Name: Task3
- Priority: osPriorityHigh
- Stack Size (Words): 128
- Entry Function: StartTask03
- Code Generation Option: Default
- Parameter: NULL
- Allocation: Dynamic
- Buffer Name: NULL
- Control Block Name: NULL

Step-8 – Further scroll down and you will find **SYS button** click on it and from **Timebase Source** change from **SYSTICK** to **TIM1**

SYS Mode and Configuration

Mode

Debug: Serial Wire

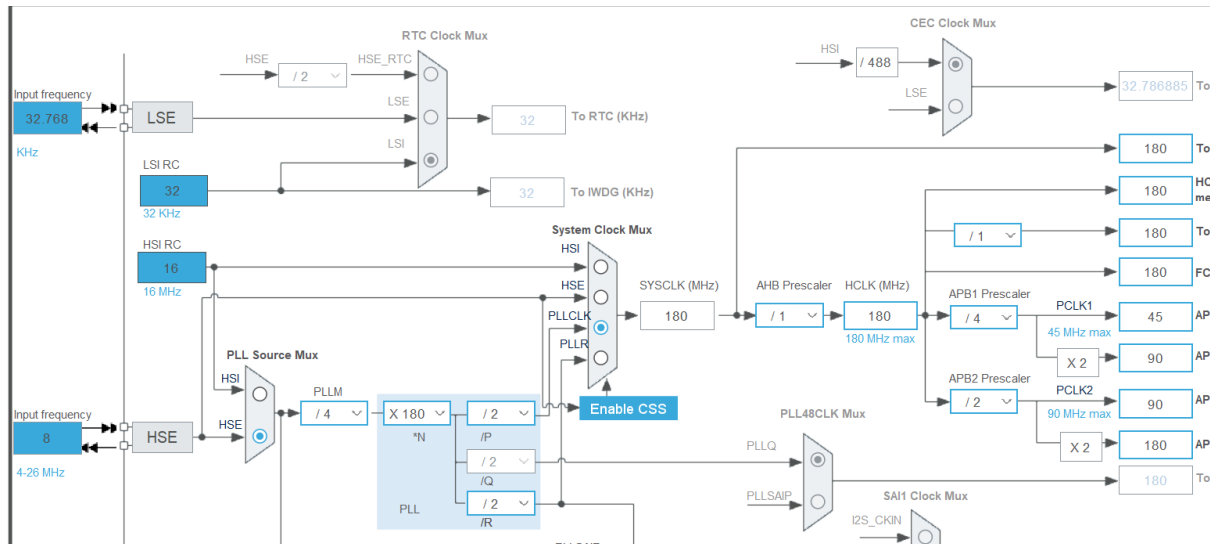
☐ System Wake-Up 0

☒ System Wake-Up 1

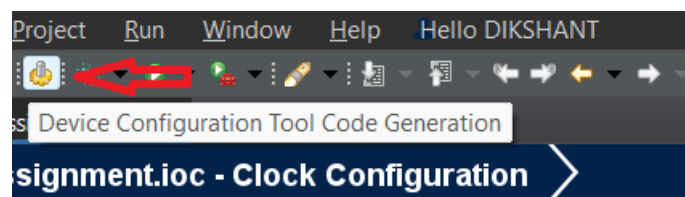
Timebase Source: TIM1

RTOS Development – 23EC361

Step-9 – Now, click on **Clock Configuration Tab** and do the changes in clock setting according to the figure below. Finally, our configuration part is done here.



Step-10 – Click on **Code Generation Tool**  option in taskbar menu.



Step-11 – Write the following code snippets in your own code as according to the given figure :

```
66 /* USER CODE BEGIN 0 */
67 char* task1_data = "Task 1 is Running!\r\n";
68 char* task2_data = "Task 2 is Running!\r\n";
69 char* task3_data = "Task 3 is Running!\r\n";
70
71 /* USER CODE END 0 */
72
```

```
288 void StartTask1(void const * argument)
289 {
290     /* USER CODE BEGIN 5 */
291     /* Infinite loop */
292     for(;;)
293     {
294         HAL_UART_Transmit(&huart2, (uint8_t*) task1_data, strlen(task1_data), HAL_MAX_DELAY);
295         HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_4);
296         osDelay(2000);
297     }
298     /* USER CODE END 5 */

```

RTOS Development – 23EC361

```

308 void StartTask02(void const * argument)
309 {
310     /* USER CODE BEGIN StartTask02 */
311     /* Infinite loop */
312     for(;;)
313     {
314         HAL_UART_Transmit(&huart2, (uint8_t*) task2_data, strlen(task2_data), HAL_MAX_DELAY);
315         HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_6);
316         osDelay(2000);
317     }
318     /* USER CODE END StartTask02 */

```

```

327 /* USER CODE END Header_StartTask03 */
328 void StartTask03(void const * argument)
329 {
330     /* USER CODE BEGIN StartTask03 */
331     /* Infinite loop */
332     for(;;)
333     {
334         HAL_UART_Transmit(&huart2, (uint8_t*) task3_data, strlen(task3_data), HAL_MAX_DELAY);
335         HAL_GPIO_TogglePin(GPIOA, GPIO_PIN_7);
336         osDelay(2000);
337     }

```

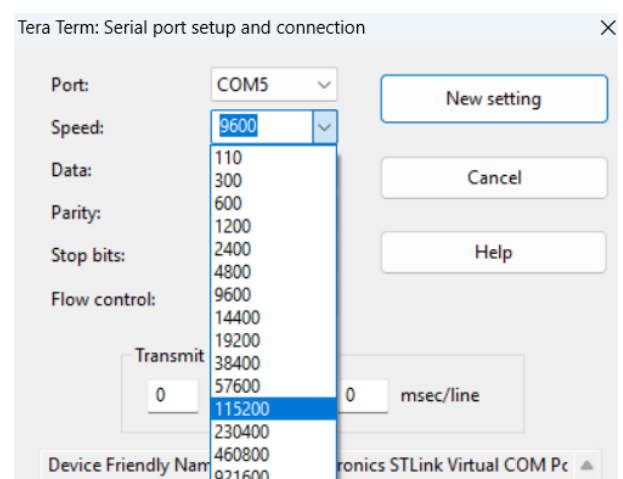
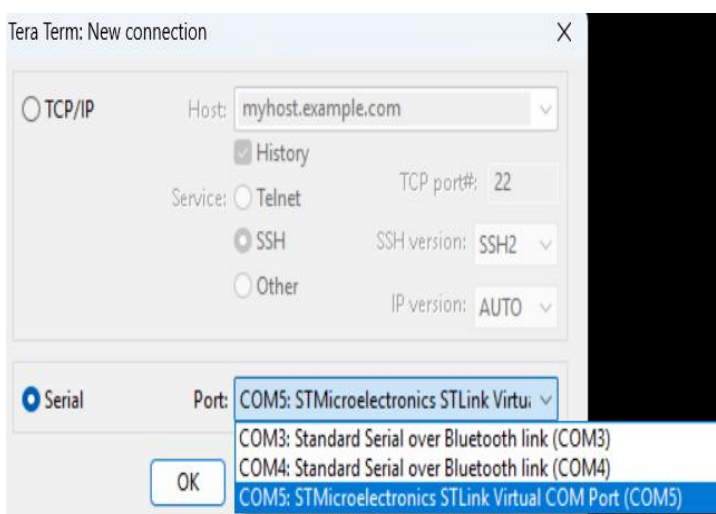
Here, your coding part is also done! Now, just connect your hardware and run it to verify.

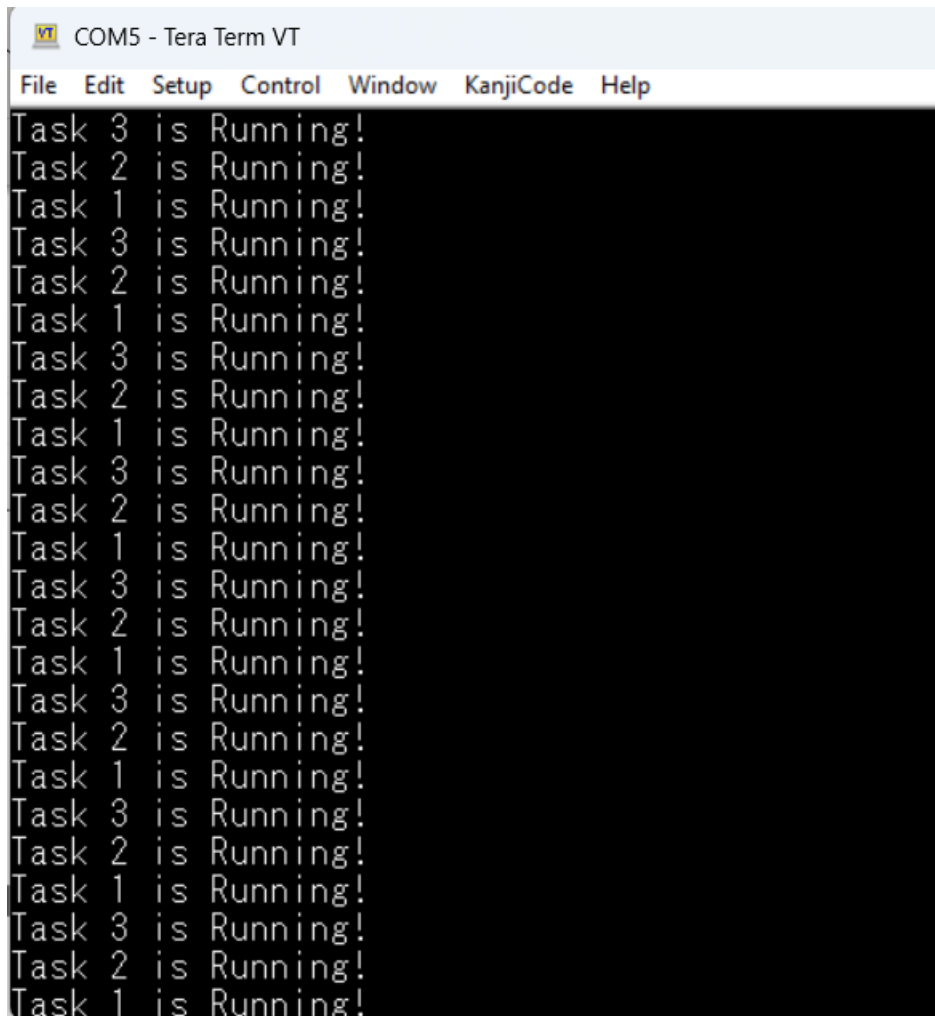
Step-12 – For checking the output we have to verify it on the serial monitor **Tera Term**. For downloading it refer the following link :

<https://tera-term.en.softonic.com/download>

Now, open Tera Term. Select **Serial** option and select the Port in which your Nucleo board is connected. Click OK.

In the **Setup** sub menu in taskbar, a pop-up window appears. Set the **Baud rate** in the **Speed** option and click **New setting**. We have selected the Baud Rate to **115200 bps**.



Result :

```
COM5 - Tera Term VT
File Edit Setup Control Window KanjiCode Help
Task 3 is Running!
Task 2 is Running!
Task 1 is Running!
Task 3 is Running!
Task 2 is Running!
Task 1 is Running!
Task 3 is Running!
Task 2 is Running!
Task 1 is Running!
Task 3 is Running!
Task 2 is Running!
Task 1 is Running!
Task 3 is Running!
Task 2 is Running!
Task 1 is Running!
Task 3 is Running!
Task 2 is Running!
Task 1 is Running!
Task 3 is Running!
Task 2 is Running!
Task 1 is Running!
```

ANS 2 : The sequence in which tasks are executed in a Real-Time Operating System (RTOS) is determined by their priority levels. Greater priority tasks—like Task3 (osPriorityHigh)—take precedence over tasks with lesser priority—like Task1 (osPriorityLow). Setting priorities helps to guarantee that important jobs are completed on time, meeting deadlines, and attending to urgent situations. Preemption highlights the importance of priority levels by allowing higher-priority actions to interrupt lower-priority ones.

Task3 is performed first in the code that is provided, and then Task2 and Task1, demonstrating the ensuing effect that priority levels have on the order in which tasks are completed.

ANS 3 : Through the implementation of three different job priorities in the RTOS, a deep comprehension of task scheduling algorithms was shown. Based on criticality, task prioritization ensures that urgent jobs are completed on time, supporting system dependability and compliance with real-time requirements. Task execution, memory allocation, and context switching are all effectively handled by the RTOS. All things considered, the performance and dependability of task control in a critical system development environment have greatly increased with the incorporation of RTOS. This has resulted to an optimized and effective embedded system by streamlining resource use, improving job management, and enabling the system to meet strict time limitations. The RTOS plays a critical role in coordinating task execution to improve system efficiency, as seen by the organized prioritization it enables.

GITHUB Link : <https://github.com/DIKSH4NT0615/RTOS>