# Environment Setup

1. Install vscode extension “PlatformIO IDE” and it will automatically install all tools you need including C/C++ extension, gcc arm toolchain, openocd and stm32cube official libraries. All you can find in the folder:

C:\Users\USERNAME\.platformio

1. Due a mistake of PlatformIO, you need to change the name of a board support package to make project include it. The original folder is:

C:\Users\USERNAME\.platformio\packages\framework-stm32cube\f4 \Drivers\BSP\STM32F429I-Discovery

And after change, it will be:

C:\Users\USERNAME\.platformio\packages\ framework-stm32cube\f4 \Drivers\BSP\DISCO\_F429ZI

1. Copy “ili9341.c” and “ili9341.h” files from the folder:

C:\Users\USERNAME\.platformio\packages\framework-stm32cube\f4 \Drivers\BSP \Components\ili9341

And paste them in DISCO\_F429ZI.

It is LCD driver file and actually the include path in the BSP is correct but something goes wrong when linking and I found copy them into the BSP can solve this problem.

1. To be able to use ST-link as UART link, you need to install virtual COM port driver from the site:

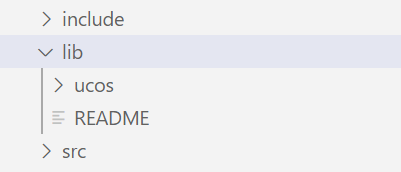
<https://www.st.com/en/development-tools/stsw-stm32102.html>

This driver is only for Windows. Some people say it is ok to use other virtual COM port driver on Linux so you will see.

1. When create a project, you simply click the “alient” in your side bar

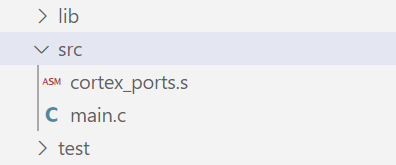
 and click “open” under “PIO Home” and you will see the homepage of PlatformIO. Then choose “+ New Project”, type a name and choose the board “ST 32F429IDISCOVERY”. The Frameword should be “STM32CUBE” and select a proper location.

1. Back to your explorer, first copy “ucos” folder into your lib:



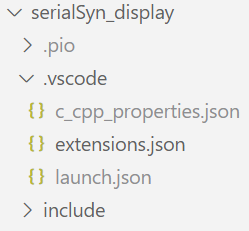
The folder contains everything of uCOS-III kernel and port files.

Copy “cortex\_ports.s” file into your src:



The assembly file will replace “pendsv\_handler” and “systick\_handler” in startup file to enable uCOS multitasking.

Now you have an environment for uCOS exercises. I had all code in a single “main.c” file to make life easier. Since you don’t want to use STM32cubeMX, you could simply copy my initialization functions. As uCOS recommended, I perform all initializations in APPSTART task after OSStart(). Most of uCOS modules are enabled by default but whenever you find a uCOS function is undefined, you could check if the module is enabled in “os\_cfg.h”. The file is used to tailor your program. Sometimes I had strange “include error” from C/C++ extension even though my program can be successfully build and upload then I will delete files under “.vscode” and reopen vscode and project.



As you suggested, I use source control now and pushed all exercises to github. The URL is:

https://github.com/BoliGao/uCOSIII-DISCO-F429ZI.git

# Exercises

1. Preemptive

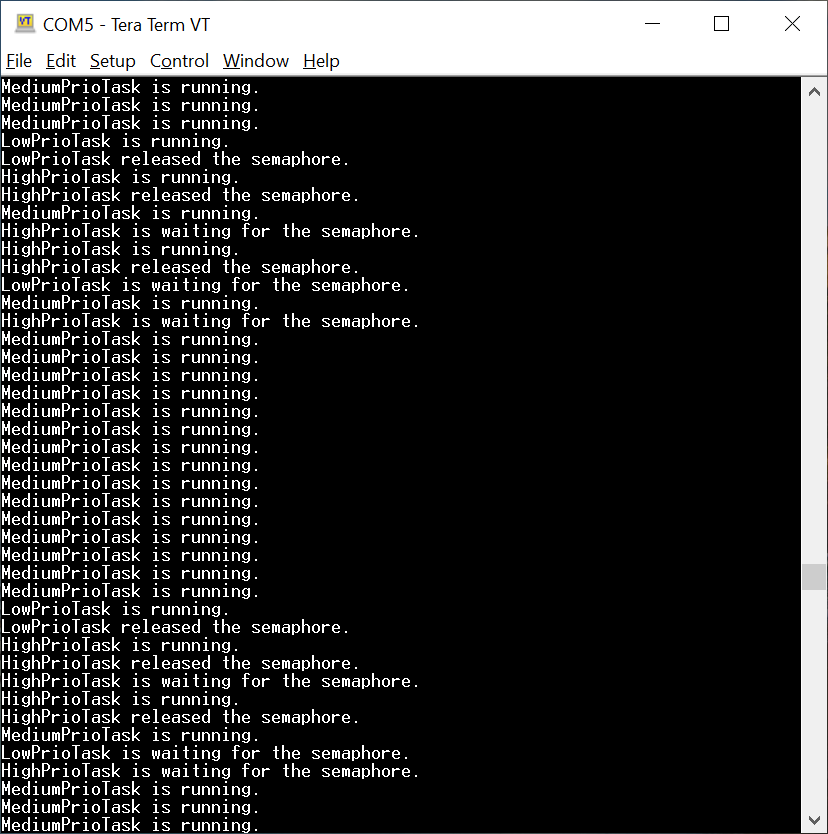
The first exercise demonstrates what is preemptive kernel. There are two tasks, LED3 and 4. The priority of LED3 is higher than 4. Both LEDs at the beginning toggle at a second interval. If you comment the delay function in LED3 task, you will see LED3 always on while LED4 never light up but if you only comment the delay function in LED4 task, you will see LED4 always on while LED3 still toggles at a second. That’s because LED3 has a higher priority and in the first situation, it runs at system clock frequency which means it will be ready to run at every system tick interrupt then LED4 has no chance to run and in the second situation, though LED4 is running at a system clock frequency, the scheduler caused by tick interrupt will still pick up a ready task with higher priority to run first so LED3 behaves as normal.

1. Task Synchronization

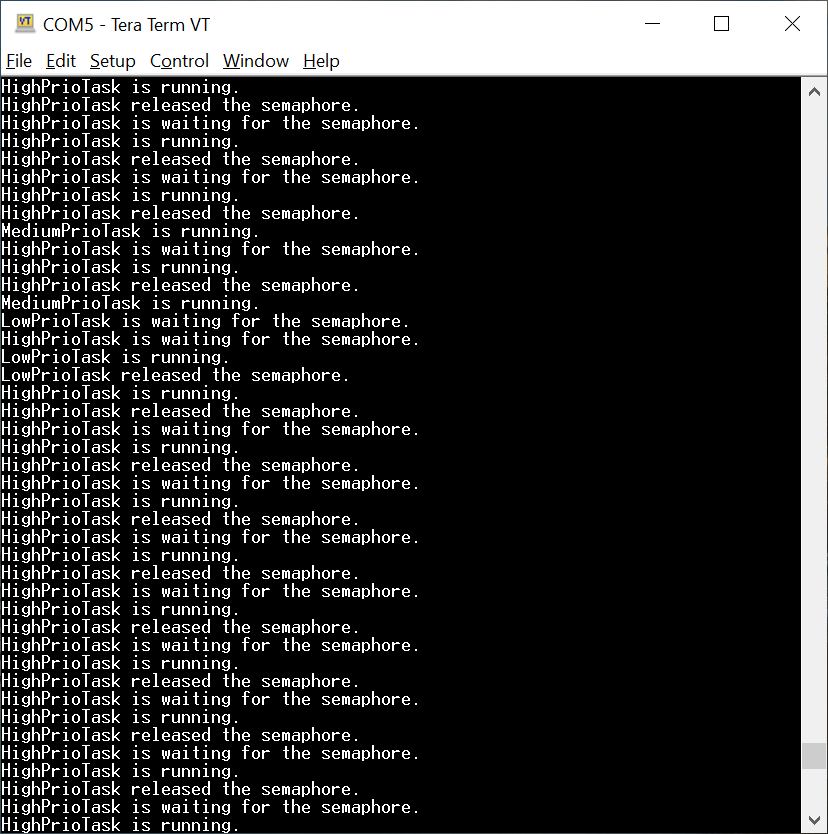
Three exercises demonstrate three ways to perform task synchronization: semaphore, task semaphore and event flag. In the semaphore exercise, LED4 follows LED3 by waiting for a semaphore sent from LED3 and in the task semaphore exercise, the sequence is LED3 > LED4 > UART so people don’t have to set a same time interval to synchronize tasks. Finally, In the event flag exercise, UART task only performs when both LED3 and LED4 have run so it is a situation that a task needs to synchronize with the occurrence of multiple events.

1. Priority Inversion and solution

Two exercises demonstrate why it is better to use mutual exclusion semaphore instead of semaphore to protect a share resource. I made three tasks here with low, medium and high priorities. Both low and high priority tasks want to use a shared integer and I simulated a situation that the shared resource is being held by low priority task while scheduling happens and we could see priority inversion when use semaphore from the picture below:



And when we change nothing else except using mutex semaphore instead of semaphore in our program. We could see:



So, whenever the high priority task wants the shared resource held by low priority task, the system will temporarily lift the priority of low priority task and make it release the resource soon.

1. Serial Synchronization

Two exercises demonstrate basic use of message queue and a combination of semaphore, task semaphore and message queue. In the sendback exercise, UART interrupt transfer a received data to transmit task with message queue and transmit task send it back to pc.

The display exercise is so far the most difficult one to do. I found I don’t have to initialize those peripherals myself because BSP will do that but the setting of system clock is crucial. In vscode environment, using HSE (high speed external) clock resource will result UART not working properly so I used HSI (high speed internal) clock resource but then I have to raise the clock frequency high enough to meet the requirement of LCD display. Also, I found my method to perform transmission with unknown size will cause problem here so make sure you know the size of file before send it. The logic of the exercise is not difficult to understand. Whenever a UART receives the file, the interrupt signals a receive task by semaphore and receive task slices the whole file into many messages and after transmit task displayed a message sent by receive task through message queue, it will tell receive task to send next one. After the whole file has been displayed, LED3 will toggle to indicate a completion and activate the UART interrupt again to wait for next file transmission. The video below is a demonstration.



1. Task communication

Two exercises demonstrate basic use of message queue and task message queue. There are two tasks, A and B, and they will tell each other how many times they have run by messages.