EE489 Real-Time Embedded Systems

Lab 2 & 3 (ST-IOT Board B-L475E-IOT01A0)

Marcus Corbin

1/30/2020

Introduction

The purpose of these two labs is to see a couple different behaviors of threads. In Lab2, two thread will be created, but will be given a parameter of ThreadFunc. In this case, the threads will run together in the same function giving the same behavior. In CubeMX, constants are assigned to each thread to output which one will run in Tera Term. In Lab3, thread2's priority will be set higher than thread1 resulting in a different output.

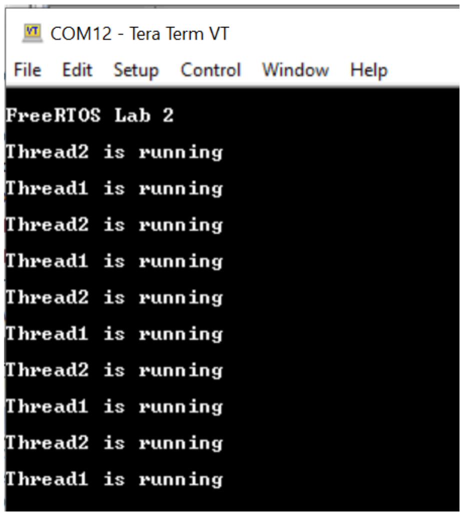
LAB 2

1. CMSIS_v1 APIs used and the corresponding FreeRTOS APIs:

- `osThreadDef( name, priority, instances, stacksz );`
  - This defines the attributes of a thread before calling osThreadCreate().
    - Name: name of the thread function
    - Priority: initial priority of the thread function
    - Instances: number of possible thread instances.
    - Stackz: stack size (in bytes) requirements for the thread function

- `osThreadCreate( const osThreadDef_t * thread_def, void * argument )`
  - Start the thread function and set it to READY state.
    - Thread_def: thread definition referenced with osThreadDefs
    - Argument: pointer that is passed to the thread function as start argument.

2. Screen shots of the program execution results (Tera Term window)



```
FreeRTOS Lab 2
Thread2 is running
Thread1 is running
Thread2 is running
Thread1 is running
Thread2 is running
Thread1 is running
Thread2 is running
Thread1 is running
Thread2 is running
Thread1 is running
```

Figure 1: Tera Term Output

On the previous screenshot, Tera Term was opened with the modified generated CubeMX code with the STM32 board connected. First, the baud rate had to be changed from 9600 to 115200. After pressing reset on the board, it first states that this is the FreeRTOS Lab1. Then, a USART1 prints out that thread2 is running. Shortly after, Thread2 completes and USART1 prints out that thread1 is running.

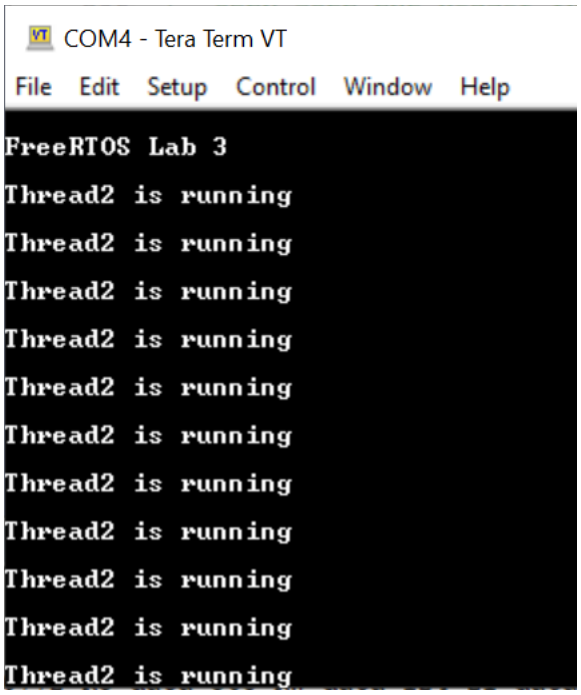
LAB 3

1. CMSIS_v1 APIs used and the corresponding FreeRTOS APIs:

- `osThreadDef( name, priority, instances, stacksz );`
    - This defines the attributes of a thread before calling osThreadCreate().
        - Name: name of the thread function
        - Priority: initial priority of the thread function
        - Instances: number of possible thread instances.
        - Stackz: stack size (in bytes) requirements for the thread function

- `osThreadCreate( const osThreadDef_t * thread_def, void * argument )`
    - Start the thread function and set it to READY state.
        - Thread_def: thread definition referenced with osThreadDefs
        - Argument: pointer that is passed to the thread function as start argument.

2. Screen shots of the program execution results (Tera Term window)



```
FreeRTOS Lab 3
Thread2 is running
Thread2 is running
Thread2 is running
Thread2 is running
Thread2 is running
Thread2 is running
Thread2 is running
Thread2 is running
Thread2 is running
Thread2 is running
Thread2 is running
```

On the previous screenshot, it is show that Thread2 continuously runs and Thread1 never shows up. This is because the priority of thread2 was set higher in CubeMX and doesn't even allow Thread1 to output in Tera Term.

Conclusion

These two labs were a good add-on to the first lab on creating two thread. It was cool to learn that a single thread function could be used instead of running two separate tasks. The one issue with lab 2 is that both threads are at the same priority and by running through the ThreadFunc makes Thread2 output first. It was also interesting to see that making Thread2 have a higher priority doesn't allow Thread1 to output. It does seem that Thread1 does run, but just doesn't have enough time to output to USART1.

Appendix: The edited source code.

LAB 2:

```
/* Private variables ---------------------------------------------------------*/
UART_HandleTypeDef huart1;

osThreadId Thread1Handle;
osThreadId Thread2Handle;

//...

/* USER CODE BEGIN PFP */
#ifdef __GNUC__
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else
#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif /* __GNUC__ */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
PUTCHAR_PROTOTYPE
{
  /* e.g. write a character to the USART1 and Loop until the end of transmission
*/
  HAL_UART_Transmit(&huart1, (uint8_t *)&ch, 1, 0xFFFF);

  return ch;
}

// ...

int main(void)
```

```
{

// ...

  /* USER CODE BEGIN 2 */
  printf("\n\rFreeRTOS Lab2\n\r");

  /* Create the thread(s) */
  /* definition and creation of Thread1 */
  osThreadDef(Thread1, Thread1Func, osPriorityNormal, 0, 128);
  Thread1Handle = osThreadCreate(osThread(Thread1), NULL);

  /* Start scheduler using CMSIS abstraction*/
  osKernelStart();

   /* We should never get here as control is now taken by the scheduler */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }

}

// ...

/* USER CODE END Header_ThreadFunc */
void ThreadFunc(void const * argument)
{
  /* USER CODE BEGIN 5 */
    volatile unsigned long ul;
  /* Infinite loop */
  for(;;)
  {
    printf("%s", (char*)argument);
        for ( ul = 0; ul < 0xFFFFFF; ul++ )
        {
        }
  }
  /* USER CODE END 5 */
}
```

LAB 3

```
/* Private variables ---------------------------------------------------------*/
UART_HandleTypeDef huart1;

osThreadId Thread1Handle;
osThreadId Thread2Handle;

//...

/* USER CODE BEGIN PFP */
#ifdef __GNUC__
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else
#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif /* __GNUC__ */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
PUTCHAR_PROTOTYPE
{
  /* e.g. write a character to the USART1 and Loop until the end of transmission
*/
  HAL_UART_Transmit(&huart1, (uint8_t *)&ch, 1, 0xFFFF);

  return ch;
}
// ...

int main(void)
{

// ...

  /* USER CODE BEGIN 2 */
  printf("\n\rFreeRTOS Lab3\n\r");

  /* Create the thread(s) */
  /* definition and creation of Thread1 */
  osThreadDef(Thread1, Thread1Func, osPriorityNormal, 0, 128);
  Thread1Handle = osThreadCreate(osThread(Thread1), NULL);
```

```c
  /* Start scheduler using CMSIS abstraction*/
  osKernelStart();

   /* We should never get here as control is now taken by the scheduler */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }

}

// ...

/* USER CODE END Header_ThreadFunc */
void ThreadFunc(void const * argument)
{
  /* USER CODE BEGIN 5 */
  volatile unsigned long ul;
  /* Infinite loop */
  for(;;)
  {
    printf("%s", (char*)argument);
        for ( ul = 0; ul < 0xFFFFFF; ul++ )
        {
        }

  }
  /* USER CODE END 5 */
}
```