EE489 Real-Time Embedded Systems

Labs 7-9 (ST-IOT Board B-L475E-IOT01A0)

Marcus Corbin

3/5/2020

Introduction

The purpose of these three labs is to see different behaviors in threads including suspending, resuming, changing priority, and deletion. In Lab 7, a counter will be initialized along with a total thread suspension. Once suspended, the count value will be printed out and then the threads will resume. In Lab 8, Thread 1 will be created and then raise the priority of Thread 2. Then, Thread 2 will lower its own priority. Finally, in Lab 9, a thread 2 will be created in thread 1 and then thread 2 will delete itself when ran.

LAB 7

1. CMSIS_v1 APIs used and the corresponding FreeRTOS APIs:

- `osThreadSuspendAll ( )`
  - Suspend all the current threads.
- `osThreadResumeAll ( )`
  - Resume all the threads that were previously suspended.

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

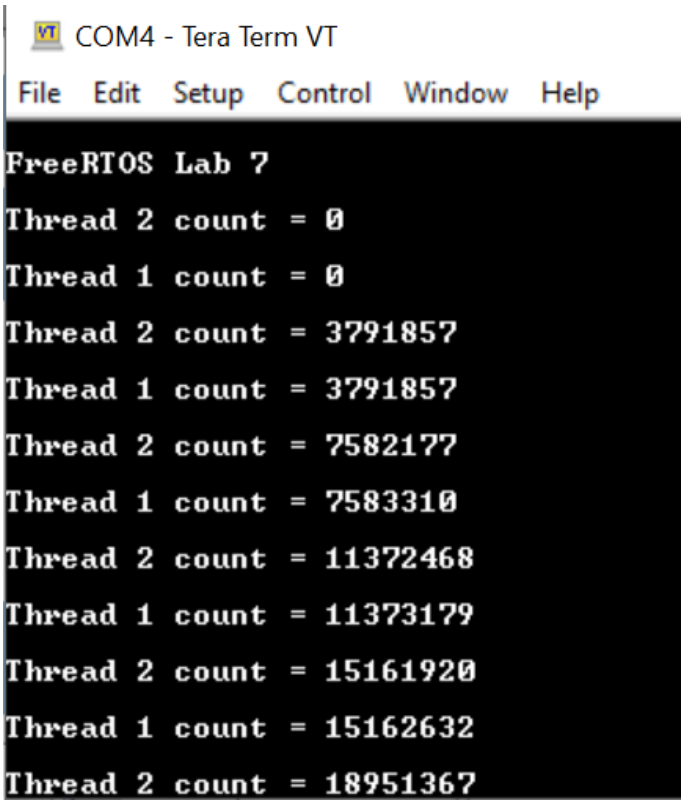

Figure 1: Lab 7 Tera Term Output

On the previous screenshot, the thread counts for thread 1 and 2 begin at zero. Once vPrintStringAndNumber is called, the tasks are previously running with a counter in the

background but first get suspended. Tera term then prints out the new count value. After being printed, both threads resume.

LAB 8

1. CMSIS_v1 APIs used and the corresponding FreeRTOS APIs:

- `osThreadGetPriority ( osThreadId thread_id )`
  - Get the priority of an active thread.
    - thread_id: thread ID obtained by osThreadCreate or osThreadGetId.
- `osThreadSetPriority ( osThreadId thread_id, osPriority priority )`
  - Change the priority of an active thread.

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

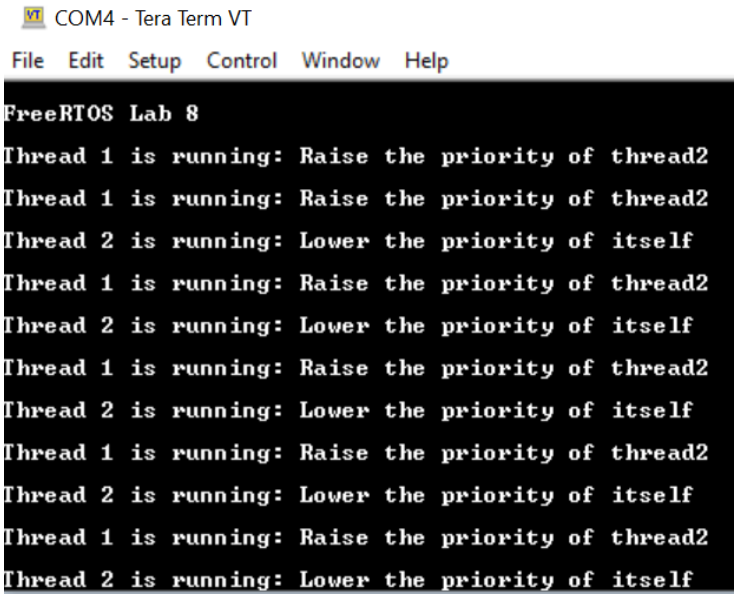

Figure 2: Lab 8 Tera Term Output

On the previous screenshot, thread one runs and then calls to have thread 2 have a higher priority. Once thread 2 is printed, it will then lower the priority of itself. This then calls for thread 1 again which loops the pervious function of raising the priority of thread 2.

LAB 9

3. CMSIS_v1 APIs used and the corresponding FreeRTOS APIs:

- `osThreadTerminate ( osThreadId thread_id )`
  - Remove the thread function from the active thread list. If thread is running, the execution will stop.
    - thread_id: thread ID obtained by osThreadCreate or osThreadGetId.

4. Screenshot of the program execution results (Tera Term window)

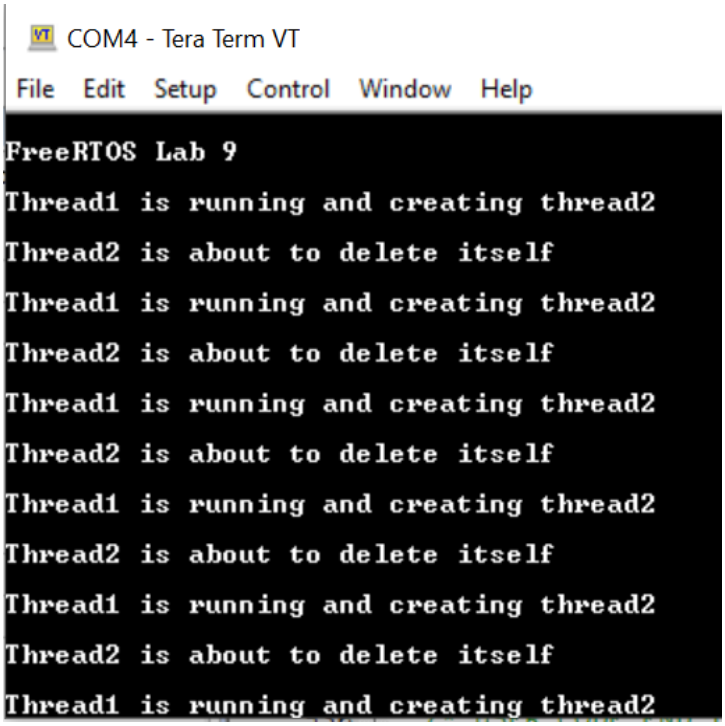

Figure 3: Lab 9 Tera Term Output

On the previous screenshot, thread 1 is initially created and then creates thread 2. Once thread 2 runs, it will delete itself. However, since thread 1 always is creating thread 2, it creates a loop. This is a simple function that can end a task even if it is already running (as mentioned in the API definition).

Conclusion

These few labs are a good view of how tasks can be utilized in different ways. It was interesting to see how large the count number got in lab 7 once all the tasks were suspended. This shows how fast the microprocessor runs task 1 and 2. For lab 8, it was good to learn how a task can call a cmsis_os API to change the thread of itself or another thread. It can also delete a thread or the thread that is currently running. Altogether, these were good thread exercises to learn that can be used when creating and using threads.

Appendix: The edited source code.

LAB 7:

```
/* 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;
}

void vApplicationIdleHook()
{
  /* This hook function does nothing but increment a counter. */
  ulIdleCycleCount++;
}

void vPrintStringAndNumber(char const *pcString, unsigned long ulValue)
{
  /* Print the string, suspending the scheduler as method of mutual
  exclusion. */
  osThreadSuspendAll();
  {
    printf(pcString, ulValue);
```

```
  }
  osThreadResumeAll();
}


// ...

int main(void)
{

// ...

  /* USER CODE BEGIN 2 */
  printf("\n\rFreeRTOS Lab 7\n\r");

  /* Create the thread(s) */
  /* definition and creation of Thread1 */
  osThreadDef(thread1, ThreadFunc, osPriorityNormal, 0, 128);
  thread1Handle = osThreadCreate(osThread(thread1), (void*)pcTextForThread1);

  /* definition and creation of thread2 */
  osThreadDef(thread2, ThreadFunc, osPriorityAboveNormal, 0, 128);
  thread2Handle = osThreadCreate(osThread(thread2), (void*)pcTextForThread2);


  /* 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 */
  /* Infinite loop */
  for(;;)
  {
    vPrintStringAndNumber((char *)argument, ulIdleCycleCount);
    osDelay(1000);
  }
  /* USER CODE END 5 */
}
```

LAB 8

```
/* 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 Lab 8\n\r");

  /* Create the thread(s) */
  /* definition and creation of Thread1 */
  osThreadDef(Thread1, ThreadFunc, osPriorityNormal, 0, 128);
  Thread1Handle = osThreadCreate(osThread(Thread1), (void*)pcTextForThread1);

  /* definition and creation of Thread2 */
  osThreadDef(Thread2, ThreadFunc, osPriorityAboveNormal, 0, 128);
  Thread2Handle = osThreadCreate(osThread(Thread2), (void*)pcTextForThread2);

  /* 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_Thread1Func */

void Thread1Func(void const * argument)
{
  /* USER CODE BEGIN 5 */
  osPriority uxPriority;
  volatile unsigned long ul;
  /* Infinite loop */
  for(;;)
  {
    printf("\n\rThread 1 is running: Raise the priority of thread2\n\r");
    for( ul = 0; ul < mainDELAY_LOOP_COUNT; ul++)
    {
```

```
      //Delay implementation
    }
    osThreadSetPriority(Thread2Handle, uxPriority + 1);
  }
  /* USER CODE END 5 */
}

/* USER CODE BEGIN Header_Thread2Func */
/**
* @brief Function implementing the Thread2 thread.
* @param argument: Not used
* @retval None
*/
/* USER CODE END Header_Thread2Func */
void Thread2Func(void const * argument)
{
  /* USER CODE BEGIN Thread2Func */
  osPriority uxPriority;
  volatile unsigned long ul;
  /* Infinite loop */
  for(;;)
  {
    printf("\n\rThread 2 is running: Lower the priority of itself\n\r");
    for( ul = 0; ul < mainDELAY_LOOP_COUNT; ul++)
    {
      //Delay implementation
    }
    osThreadSetPriority(Thread2Handle, uxPriority - 1);
  }
  /* USER CODE END Thread2Func */
}
```

LAB 9

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

osThreadId Thread1Handle;
osThreadId Thread2Handle;

//...

/* USER CODE BEGIN PFP */
#ifdef __GNUC__
```

```c
#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 Lab 9\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_Thread1Func */
void Thread1Func(void const * argument)
{
  /* USER CODE BEGIN 5 */
  /* Infinite loop */
  for(;;)
  {
    printf("\n\rThread1 is running and creating thread2\n\r");

    /* definition and creation of Thread2 */
    osThreadDef(Thread2, Thread2Func, osPriorityAboveNormal, 0, 128);
    Thread2Handle = osThreadCreate(osThread(Thread2), NULL);

    osDelay(1000);
  }
  /* USER CODE END 5 */
}

/* USER CODE BEGIN Header_Thread2Func */
/**
* @brief Function implementing the Thread2 thread.
* @param argument: Not used
* @retval None
*/
/* USER CODE END Header_Thread2Func */
void Thread2Func(void const * argument)
{
  /* USER CODE BEGIN Thread2Func */
  /* Infinite loop */
  printf("\n\rThread2 is about to delete itself\n\r");

  osThreadTerminate(Thread2Handle);

  /* USER CODE END Thread2Func */
}
```