|  |
| --- |
|  |
| **EE489 Real-Time Embedded Systems** |
| Labs 7-9 (ST-IOT Board B-L475E-IOT01A0)  *Marcus Corbin* |
|  |
| *3/2/2020* |

|  |
| --- |
|  |

# Introduction

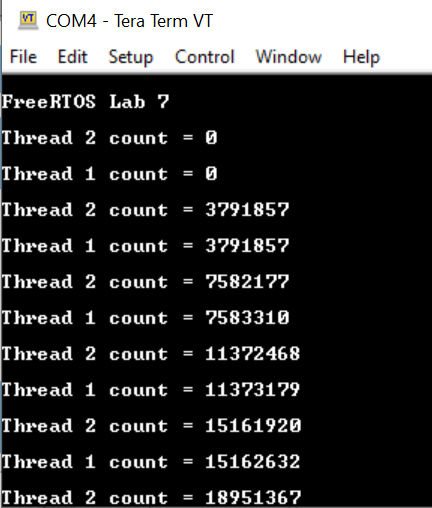
The purpose of these three labs is to see different behaviors of delays in FreeRTOS using CMSIS functions. In Lab4, an API will be used instead of using a for loop in previous labs. Lab 5

**LAB 7**

# CMSIS\_v1 APIs used and the corresponding FreeRTOS APIs:

* osThreadSuspendAll ( )
  + Suspend all the current threads.
* osThreadSuspendAll ( )
  + Resume all the threads that were previously suspended.

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

 **Figure 1: Lab 7 Tera Term Output**

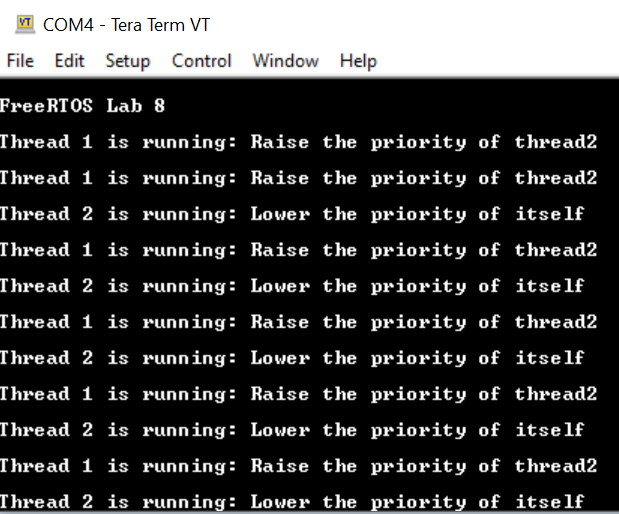
On the previous screenshot, the thread counts for thread 1 and 2 begin at zero. Once vPrintStringAndNumber is called, the tasks are previously running with a counter in the background but first get suspended. Tera term then prints out the new count value. After being printed, both threads resume again.

**LAB 8**

# 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.

# Screenshot of the program execution results (Tera Term window)

  
**Figure 2: Lab 5 Tera Term Output**

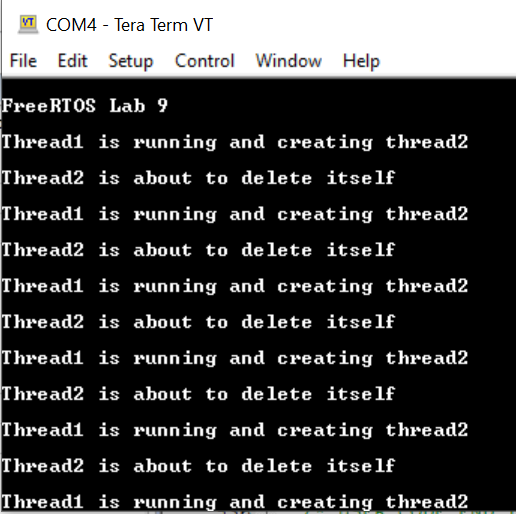
On the previous screenshot, thread one runs and then calls to have thread 2 have a higher priority. Once thread 2 is printed, it will then lower the priority of itself. This then calls for thread 1 again which loops the pervious function of raising the priority of thread 2.

**LAB 9**

# 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.

# Screenshot of the program execution results (Tera Term window)



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 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 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\_\_

#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 \*/

}