|  |
| --- |
|  |
| **EE489 Real-Time Embedded Systems** |
| Labs 4-6 (ST-IOT Board B-L475E-IOT01A0)  *Marcus Corbin* |
|  |
| *3/2/2020* |

|  |
| --- |
|  |

# Introduction

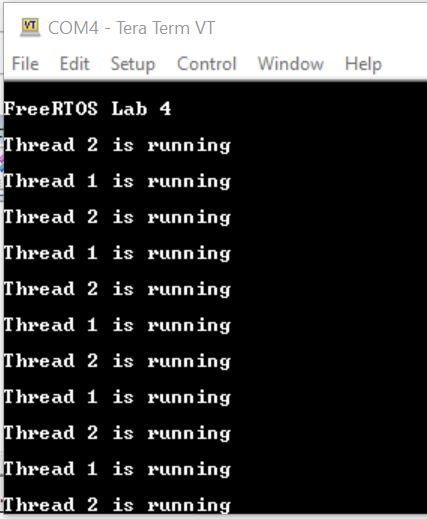
The purpose of these three labs is to see different behaviors of delays in FreeRTOS using CMSIS functions. In Lab4, an API will be used instead of using a for loop in previous labs. Lab 5

**LAB 4**

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

* osDelay ( uint32\_t millisec )
  + Wait for a specified time period in milliseconds.
    - millisec: time delay value

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

 **Figure 1: Lab 4 Tera Term Output**

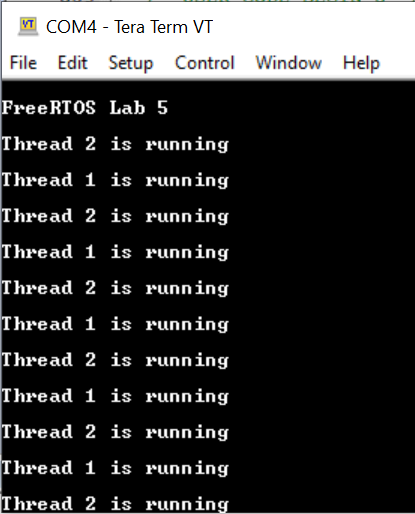
On the previous screenshot, a similar result is shown as if the for loop is used to create a delay. However, in this case the call to osDelay (1000) gives a 1 second delay instead. This allows Thread 2 to run first, wait one second, then run Thread 1. This then repeats forever in the ThreadFunc loop.

**LAB 5**

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

* osDelayUntil ( uint32\_t ticks )
  + Waits until an absolute time (specified in kernel ticks) is reached.
    - ticks: absolute time in ticks
* osKernelSysTick ( void )
  + Get the value of the Kernel SysTick timer for time comparison.

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

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

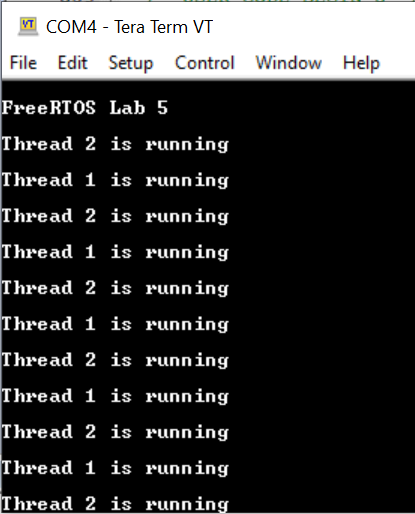
On the previous screenshot, it is show that Thread2 continuously runs and Thread1 never shows up. This is because the priority of thread2 was set higher in CubeMX and doesn’t even allow Thread1 to output in Tera Term.

**LAB 6**

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

* osDelayUntil ( uint32\_t ticks )
  + Waits until an absolute time (specified in kernel ticks) is reached.
    - ticks: absolute time in ticks
* osKernelSysTick ( void )
  + Get the value of the Kernel SysTick timer for time comparison.

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



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 4:**

/\* 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 4\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 \*/

  volatile unsigned long ul;

  /\* Infinite loop \*/

  for(;;)

  {

    printf("%s", (char\*)argument);

//    for ( ul = 0; ul < 0xFFFFFF; ul++ )

//    {

//    }

    osDelay(1000);

  }

  /\* USER CODE END 5 \*/

}

**LAB 5**

/\* 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 Lab5\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 \*/

  uint32\_t PreviousWakeTime;

  PreviousWakeTime = osKernelSysTick();

  /\* Infinite loop \*/

  for(;;)

  {

    printf("%s", (char \*)argument);

    osDelayUntil(&PreviousWakeTime, 1000);

  }

  /\* USER CODE END 5 \*/

}

**LAB 6**

/\* Private variables ---------------------------------------------------------\*/

osThreadId ContinuousT1Handle;

osThreadId ContinuousT2Handle;

osThreadId PeriodTHandle;

//...

/\* 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 Lab6\n\r");

  /\* Create the thread(s) \*/

  /\* definition and creation of Thread1 \*/

  osThreadDef(ContinuousT1, ContinuousTFunc, osPriorityNormal, 0, 128);

  ContinuousT1Handle = osThreadCreate(osThread(ContinuousT1), (void\*)pcTextForThread1);

  /\* definition and creation of ContinuousT2 \*/

  osThreadDef(ContinuousT2, ContinuousTFunc, osPriorityNormal, 0, 128);

  ContinuousT2Handle = osThreadCreate(osThread(ContinuousT2), (void\*)pcTextForThread2);

  /\* definition and creation of PeriodT \*/

  osThreadDef(PeriodT, PeriodTFunc, osPriorityAboveNormal, 0, 128);

  PeriodTHandle = osThreadCreate(osThread(PeriodT), (void \*)pcTextForThread3);

  /\* 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\_ContinuousTFunc \*/

void ContinuousTFunc(void const \* argument)

{

  /\* USER CODE BEGIN 5 \*/

  volatile unsigned long ul;

  /\* Infinite loop \*/

  for(;;)

  {

    printf("%s", (char \*)argument);

    for ( ul = 0; ul < mainDELAY\_LOOP\_COUNT; ul++)

    {

    }

  }

  /\* USER CODE END 5 \*/

}

/\* USER CODE BEGIN Header\_PeriodTFunc \*/

/\*\*

\* @brief Function implementing the PeriodT thread.

\* @param argument: Not used

\* @retval None

\*/

/\* USER CODE END Header\_PeriodTFunc \*/

void PeriodTFunc(void const \* argument)

{

  /\* USER CODE BEGIN PeriodTFunc \*/

  uint32\_t PreviousWakeTime;

  PreviousWakeTime = osKernelSysTick();

  /\* Infinite loop \*/

  for(;;)

  {

    printf("%s", (char \*)argument);

    osDelayUntil(&PreviousWakeTime, 1000);

  }

  /\* USER CODE END PeriodTFunc \*/

}