EE489 Real-Time Embedded Systems

Labs 4-6 (ST-IOT Board B-L475E-IOT01A0)

Marcus Corbin

3/5/2020

Introduction

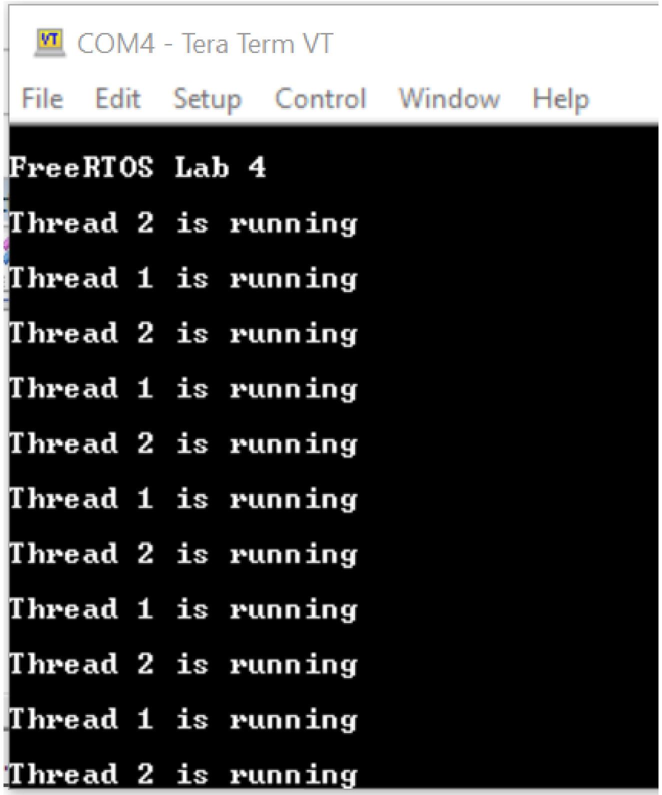
The purpose of these three labs is to see different behaviors of delays in FreeRTOS using CMSIS functions. In Lab4, an API will be used instead of using a for loop in previous labs. Lab 5 also uses an osDelay but calls for an OsDelayUntil which waits until the systick function. Lastly, in Lab 6 a continuous function will be called along with a period function that calls the osDelayUntil in Lab 5.

<u>LAB 4</u>

1. 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

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



```
FreeRTOS Lab 4
Thread 2 is running
Thread 1 is running
Thread 2 is running
Thread 1 is running
Thread 2 is running
Thread 1 is running
Thread 2 is running
Thread 1 is running
Thread 2 is running
Thread 1 is running
Thread 2 is running
```

Figure 1: Lab 4 Tera Term Output

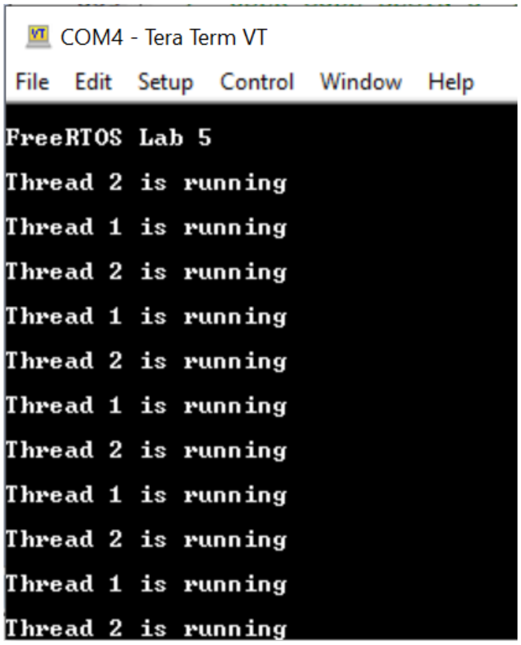
On the previous screenshot, a similar result is shown as if the for loop is used to create a delay. However, in this case the call to osDelay (1000) gives a 1 second delay instead. This allows Thread 2 to run first, wait one second, then run Thread 1. This then repeats forever in the ThreadFunc loop.

LAB 5

1. 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.

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



```
FreeRTOS Lab 5
Thread 2 is running
Thread 1 is running
Thread 2 is running
Thread 1 is running
Thread 2 is running
Thread 1 is running
Thread 2 is running
Thread 1 is running
Thread 2 is running
Thread 1 is running
Thread 2 is running
```

Figure 2: Lab 5 Tera Term Output

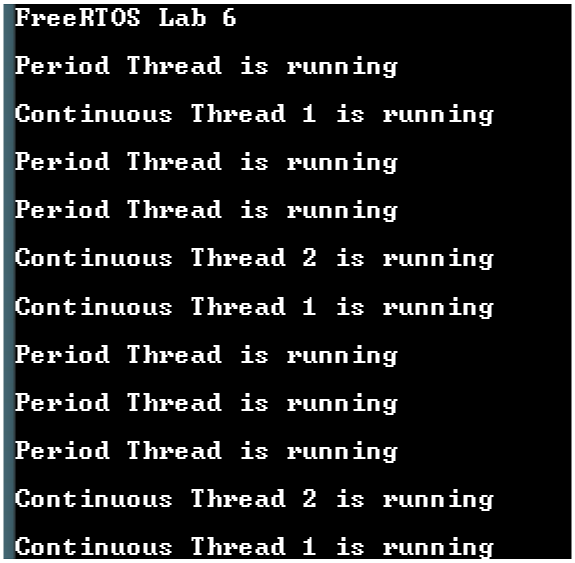
On the previous screenshot, a similar result from using osDelay is shown. However, this shows the delay using the delay until the systick is called.

LAB 6

3. 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.

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



```
FreeRTOS Lab 6
Period Thread is running
Continuous Thread 1 is running
Period Thread is running
Period Thread is running
Continuous Thread 2 is running
Continuous Thread 1 is running
Period Thread is running
Period Thread is running
Period Thread is running
Continuous Thread 2 is running
Continuous Thread 1 is running
```

Figure 3: Lab 6 Tera Term Output

On the previous screenshot, the period thread runs first along with a continuous thread 1. Shortly after, the period thread runs a couple times and then a loop of the continuous thread 2 and continuous thread 1 run.

Conclusion

These three labs had good insight onto how delays can be used with threads. An osDelay function is great to use when a specified delay time is required. However, the osDelayUntil can be used in various ways. This could be used in a single thread or in multiple threads. Altogether, different types of delays can be used in threads and other cmsis_os APIs as well.

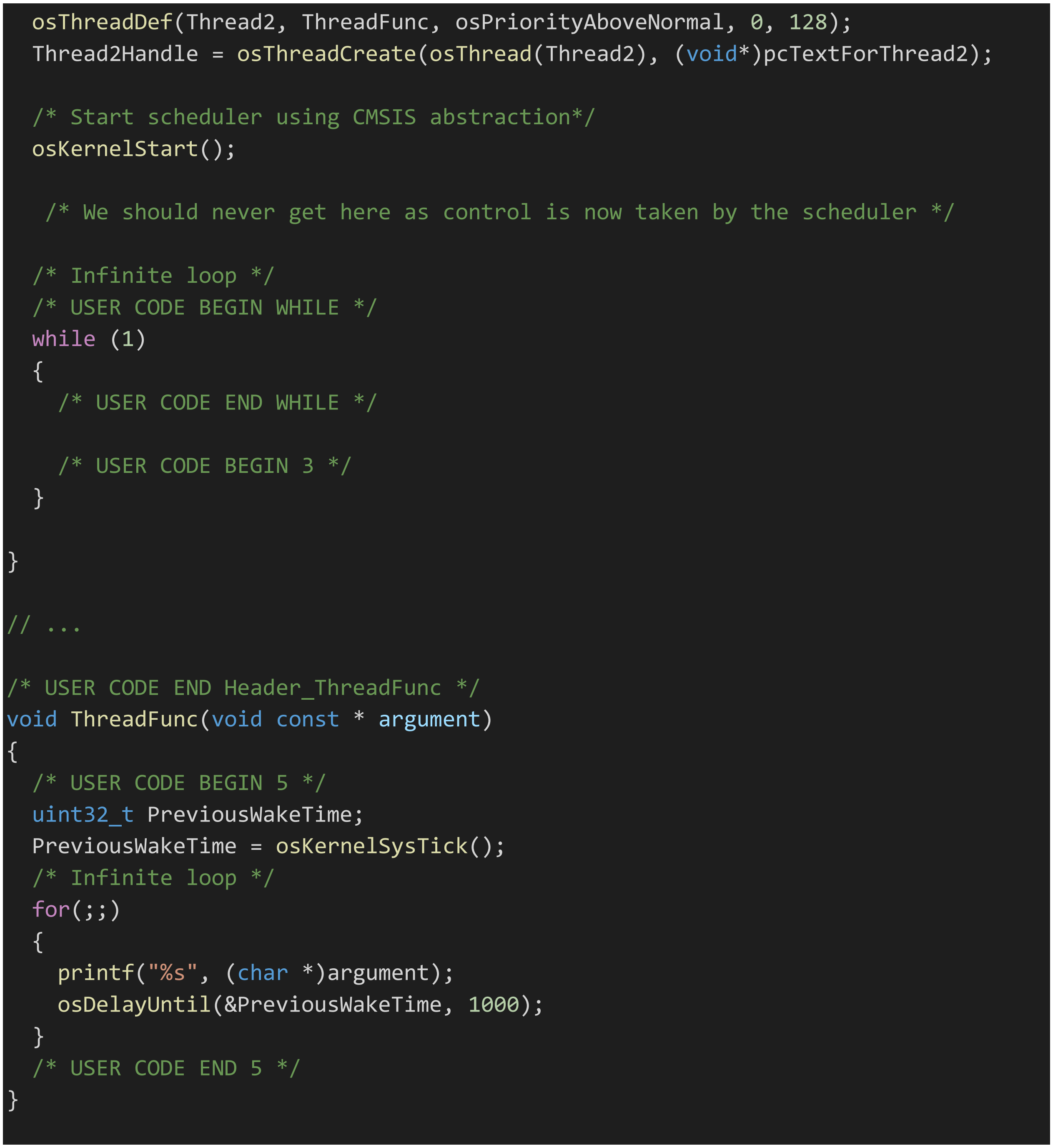
Appendix: The edited source code.

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 */
}
```

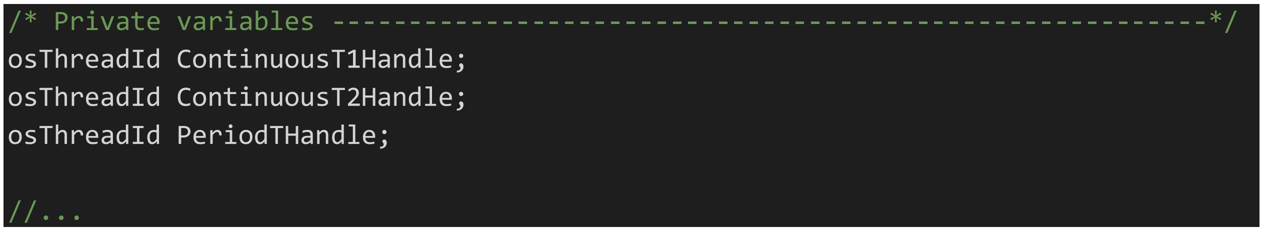
LAB 5

```
/* 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 */
}
```

<u>LAB 6</u>



```
/* Private variables ---------------------------------------------------------*/
osThreadId ContinuousT1Handle;
osThreadId ContinuousT2Handle;
osThreadId PeriodTHandle;

//...
```

```c
/* 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 */
}
```