CG2271: Real-Time Operating Systems

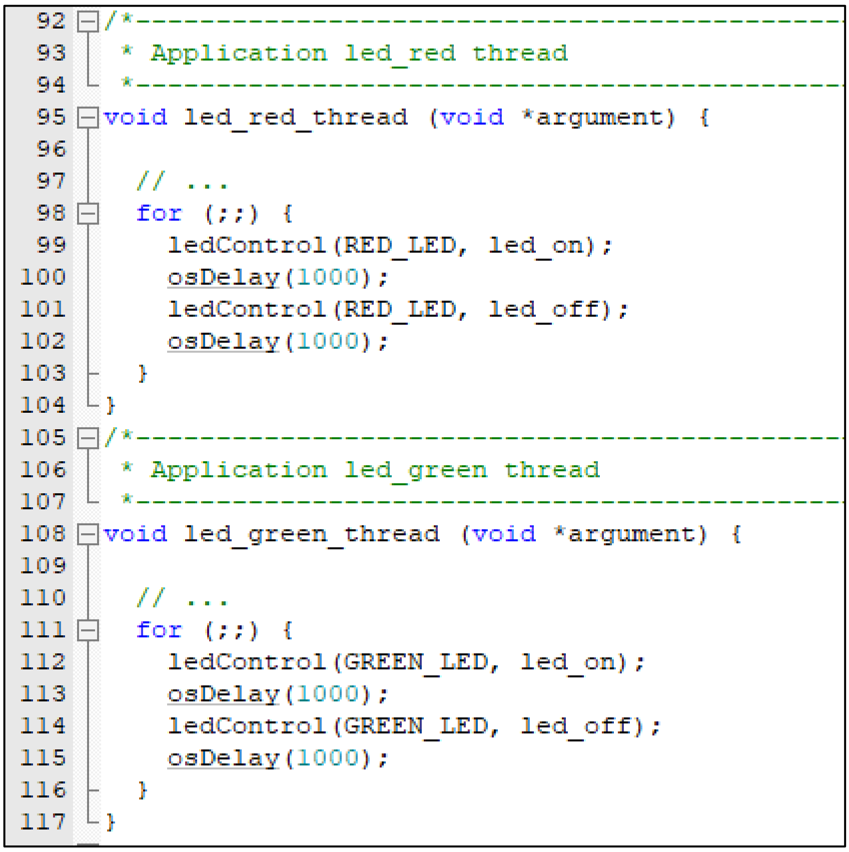
Lab 7: Priority and Mutex

In this lab, you will first get to see how priorities of a task can be changed. We will then explore how we can use Mutexes to control access to a shared resource.

Part1: Creating a new RTOX RTX Project

Refer back to Lab 6 if you have forgotten this step. Call this project myMutex. Once the project has been created copy over the code from Lab 6 with both the red_led_thread and green_led_thread. Compile and download the code. Confirm that you observe that the RGB led blinks as ‘Yellow’.

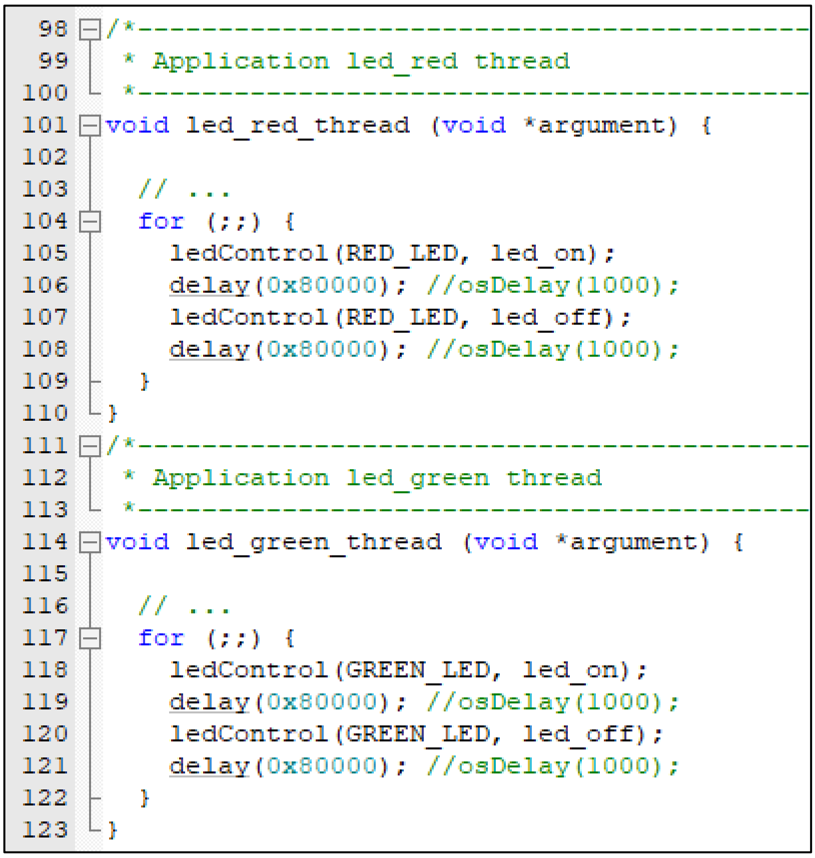
A snapshot of the code is shown below.



```
/*----------------------------------------------
 * Application led_red thread
 *----------------------------------------------
void led_red_thread (void *argument) {

  // ...
  for (;;) {
    ledControl(RED_LED, led_on);
    osDelay(1000);
    ledControl(RED_LED, led_off);
    osDelay(1000);
  }
}
/*----------------------------------------------
 * Application led_green thread
 *----------------------------------------------
void led_green_thread (void *argument) {

  // ...
  for (;;) {
    ledControl(GREEN_LED, led_on);
    osDelay(1000);
    ledControl(GREEN_LED, led_off);
    osDelay(1000);
  }
}
```

Part2: Priority Assignment

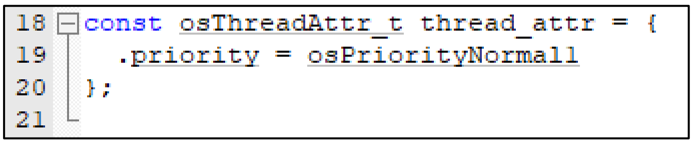
We will now change the osDelay() in the threads to the normal _delay() that you had used in the earlier studios. A snapshot is shown below.



```
/*----------------------------------------------------------------------------
 * Application led_red thread
 *---------------------------------------------------------------------------*/
void led_red_thread (void *argument) {

  // ...
  for (;;) {
    ledControl(RED_LED, led_on);
    delay(0x80000); //osDelay(1000);
    ledControl(RED_LED, led_off);
    delay(0x80000); //osDelay(1000);
  }
}
/*----------------------------------------------------------------------------
 * Application led_green thread
 *---------------------------------------------------------------------------*/
void led_green_thread (void *argument) {

  // ...
  for (;;) {
    ledControl(GREEN_LED, led_on);
    delay(0x80000); //osDelay(1000);
    ledControl(GREEN_LED, led_off);
    delay(0x80000); //osDelay(1000);
  }
}
```

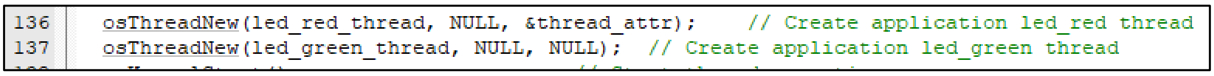
We learnt in Lab 6 that the reason for this behaviour is because RTX periodically switches between tasks in a round-robin manner for tasks with the same priority. We will first change the priority of the red_led_thread so that it is higher than the green_led_thread.

We first need to define the attributes for the thread so that it can be passed along when the osThreadNew() is called. The code snippet for the attributes definition looks as shown below.



```
const osThreadAttr_t thread_attr = {
  .priority = osPriorityNormal1
};
```

We now need to pass these attributes when we call osThreadNew(). The cod snippet looks as shown below.



```
  osThreadNew(led_red_thread, NULL, &thread_attr);    // Create application led_red thread
  osThreadNew(led_green_thread, NULL, NULL);  // Create application led_green thread
```

Only the led_red_thread has the modified attributes. The green_led_thread is still created with the default attributes.

LAB REVIEW

Q1. What is the default priority level at which the led_green_thread is created?

Q2. What are the highest and lowest priority levels that can be assigned to a task?

Compile and download the code.

LAB REVIEW

Q3. State your observation. Explain why you see such a behaviour.

Part3: Mutex for your Critical Section

Revert your delay functions back to using the osDelay().

Remove the attribute setting for the led_red_thread in osThreadNew() and replace it with NULL.

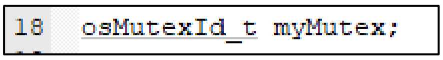
Compile and Download your code.

You should observe the Yellow Colour blinking at 1s intervals.

We will now be using Mutexes to control the access to the critical section of the code. In this case, the critical sections refers to the part of the code controlling the RGB LED.

Step 1:

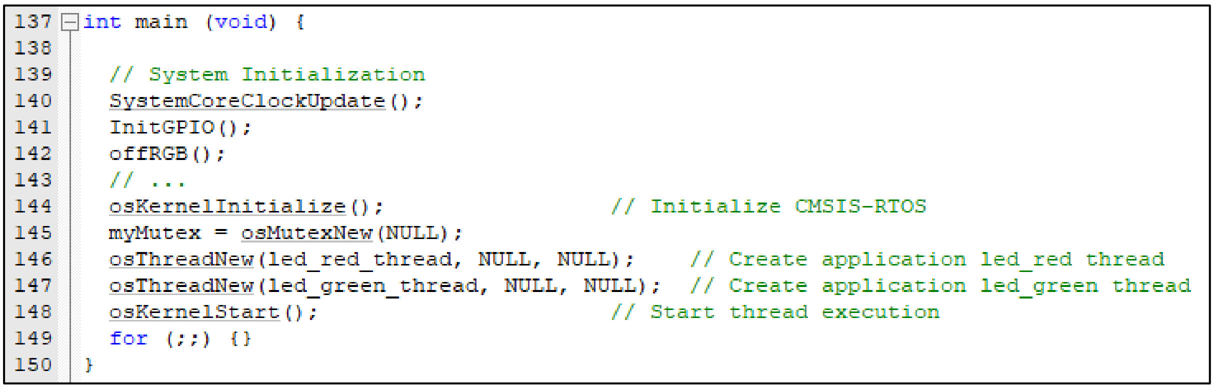
We first need to create a mutexId (globally) as shown below:



```
osMutexId_t myMutex;
```

Step2:

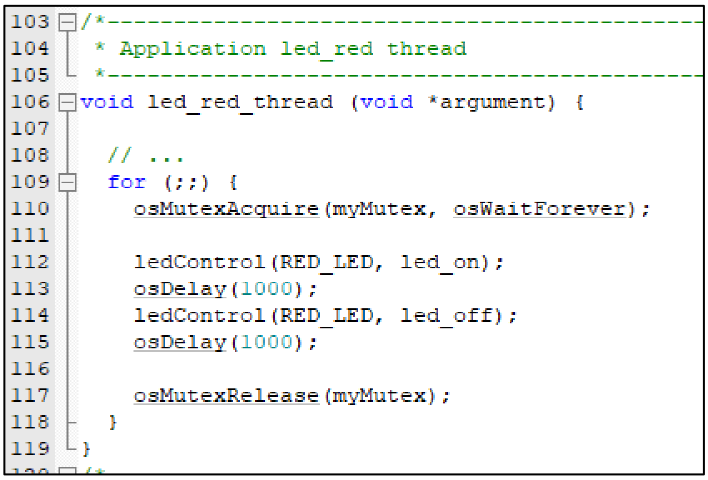
We can then create the mutex in the main().



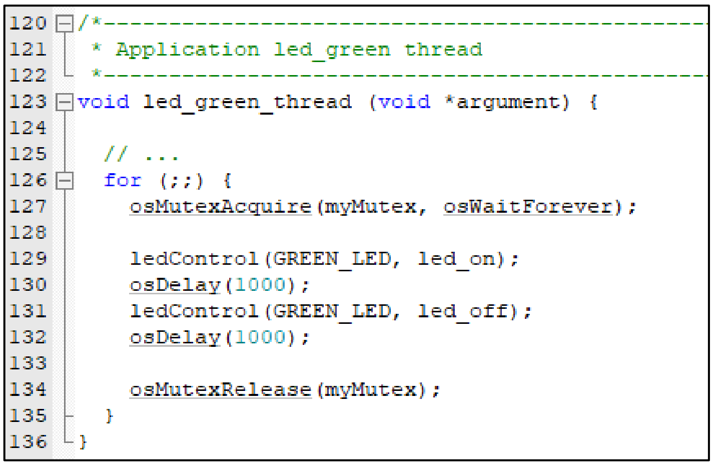
```
int main (void) {

  // System Initialization
  SystemCoreClockUpdate();
  InitGPIO();
  offRGB();
  // ...
  osKernelInitialize();                 // Initialize CMSIS-RTOS
  myMutex = osMutexNew(NULL);
  osThreadNew(led_red_thread, NULL, NULL);    // Create application led_red thread
  osThreadNew(led_green_thread, NULL, NULL);  // Create application led_green thread
  osKernelStart();                      // Start thread execution
  for (;;) {}
}
```

Step3: Use the Mutex

We can now use the mutex to protect the critical sections of the code. Modify your code as shown below for both the tasks. Compile and download the code.



```
/*----------------------------------------------------------------------------
 * Application led_red thread
 *---------------------------------------------------------------------------*/
void led_red_thread (void *argument) {

  // ...
  for (;;) {
    osMutexAcquire(myMutex, osWaitForever);

    ledControl(RED_LED, led_on);
    osDelay(1000);
    ledControl(RED_LED, led_off);
    osDelay(1000);

    osMutexRelease(myMutex);
  }
}
```



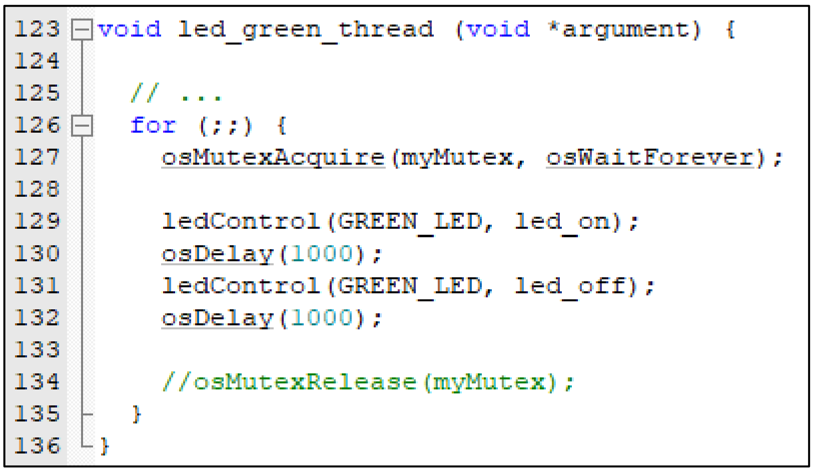
```
/*----------------------------------------------------------------------------
 * Application led_green thread
 *---------------------------------------------------------------------------*/
void led_green_thread (void *argument) {

  // ...
  for (;;) {
    osMutexAcquire(myMutex, osWaitForever);

    ledControl(GREEN_LED, led_on);
    osDelay(1000);
    ledControl(GREEN_LED, led_off);
    osDelay(1000);

    osMutexRelease(myMutex);
  }
}
```

LAB REVIEW

Q4. Describe your observation. Explain why it is as such.

Part4: Mutex Usage Rules

We will now modify the code in led_green_thread to remove the osMutexRelease(). We will still have the osMutexAcquire(). The code snippet is shown below.



```
void led_green_thread (void *argument) {

  // ...
  for (;;) {
    osMutexAcquire(myMutex, osWaitForever);

    ledControl(GREEN_LED, led_on);
    osDelay(1000);
    ledControl(GREEN_LED, led_off);
    osDelay(1000);

    //osMutexRelease(myMutex);
  }
}
```

Compile and download your code.

LAB REVIEW

Q5. Describe your observation. Explain why it is as such.

Summary

In this lab, you saw how we can make use of Mutexes to control access to a critical region of the code. At the same time, we also explored some side effects of improper usage of Mutexes. In the next lab, we will explore how tasks can synchronize with each other.