CG2271: Real-Time Operating Systems

Lab 8: Semaphores and Interrupts

In this lab, you will first explore Semaphores and how they can be used as Binary Semaphores. We will then use Semaphores to signal and synchronize between an ISR and a Task. We will explore both the Push Button Interrupt and the Serial Interrupt.

Part1: Creating a new RTOX RTX Project

Refer back to Lab 6 if you have forgotten this step. Call this project mySem. Once the project has been created copy over the code from Lab 6 with both the red_led_thread and green_led_thread and the mutex calls. Compile and download the code. Confirm that you observe that the RGB led blinks as Yellow with a 1s interval.

The code snippet 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: Adding a Binary Semaphore

We will now add a Binary Semaphore to the code to replicate the behaviour that we saw with the Mutex.

Step 1: Declare a new global sem_ID as shown.

```
osSemaphoreId_t mySem;
```

Step 2: Create a new Semaphore with an initial value of 1.

```
int main (void) {

  // System Initialization
  SystemCoreClockUpdate();
  InitGPIO();
  offRGB();
  // ...

  osKernelInitialize();                    // Initialize CMSIS-RTOS
  mySem = osSemaphoreNew(1,1,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 (;;) {}
}
```

Step 3: Use the SemAcquire() and SemRelease() as shown below.

```
void led_red_thread (void *argument) {

  // ...
  for (;;) {
    osSemaphoreAcquire(mySem, osWaitForever);

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

    osSemaphoreRelease(mySem);
  }
}
```

```
void led_green_thread (void *argument) {

  // ...
  for (;;) {
    osSemaphoreAcquire(mySem, osWaitForever);

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

    osSemaphoreRelease(mySem);
  }
}
```

When we use Semaphores in this way, we call them Binary Semaphores. They behave very similar to Mutexes. However, Semaphores have much greater uses such as Synchronization between tasks, which we will be exploring next.

Part 3: Semaphores to trigger Tasks

We will first integrate the INT code from the Switch from Lab 3 into your current project. Remember to call the initSwitch() in your main() to ensure that the GPIO is configured correctly. Your main() will now look as shown below.

```
int main (void) {

  // System Initialization
  SystemCoreClockUpdate();
  initSwitch();
  InitGPIO();
  offRGB();
  // ...

  osKernelInitialize();                 // Initialize CMSIS-RTOS
  mySem = osSemaphoreNew(1,0,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 (;;) {}
}
```

*Note that the osSemaphoreNew() also has different parameters as before.

LAB REVIEW
Q1. Explain the THREE parameters that are passed when we call osSemaphoreNew().

Compile and Download the code with the new Semaphore Parameters.

LAB REVIEW
Q2. Describe your observation. Explain why it is as such.

Modify the led_red_thread() and led_green_thread() by removing both their osSemaphoreRelease() calls. Both the threads will look like what is shown below.

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

  // ...
  for (;;) {
    osSemaphoreAcquire(mySem, osWaitForever);

    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 (;;) {
    osSemaphoreAcquire(mySem, osWaitForever);

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

Modify the Push-Button IRQ Handler to post the Semaphore as shown below. The delay() function call is to help with the switch debouncing.

```
void PORTD_IRQHandler()
{
  // Clear Pending IRQ
  NVIC_ClearPendingIRQ(PORTD_IRQn);

  delay(0x80000);
  osSemaphoreRelease(mySem);

  //Clear INT Flag
  PORTD->ISFR |= MASK(SW_POS);
}
```

Compile and Download the code.

LAB REVIEW: **Q3.** Describe your observation. Explain why it is as such.

Part 4: Semaphores for Serial

In this last part, we will be sending data from our App to control the different LED's.

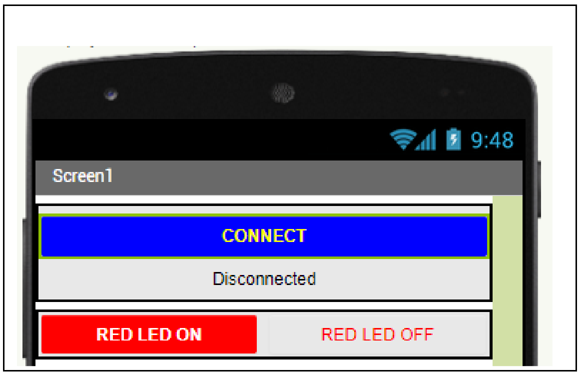
Step 1: Integrate the **UART Interrupt** code from Lab 6.

Step 2: Remove the Transmit Functionality from the UART module.

Step 3: Remove the osSemaphoreRelease() from the Push Button IRQ Handler.

Step 4: In the UART IRQ Handler, once the data packet is received, release the Semaphore.

Step 5: Launch the App with the functionality from Lab 6. It should look something like this:



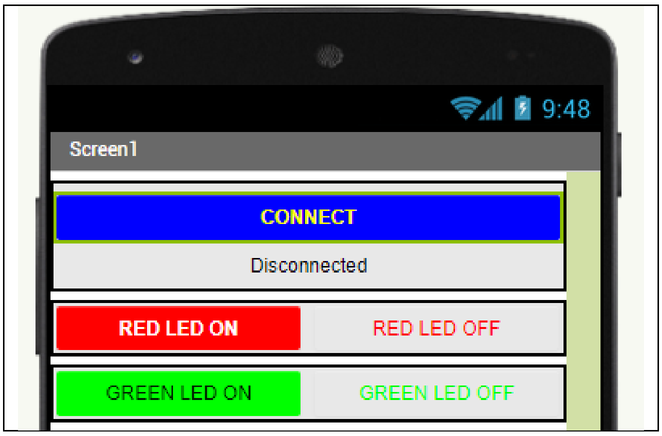
LAB REVIEW:

Q4. Provide the code for the UART ISR that handle the received data.

Q5. Explain the observation on the RGB LED everytime you press the RED_LED_ON button.

Step 6: Now add another Semaphore to your main(). The TWO different semaphores are to control the different colours depending on which button was pressed.

Step 7: Your App will now have a new row of buttons to control the Green LED.



LAB REVIEW:

Q6. Demonstrate this functionality to your Lab TA. You must be able to send different commands from the App and decode the data in the UART IRQ handler. Subsequently, the appropriate Semaphore must be released to control the specific LED only. In this demo, you don't need to implement the LED_OFF buttons. Whenever, the LED_ON button command is received, the appropriate LED must blink once.

Provide the code for your UART IRQ handler.

Summary

In this lab, you saw how we can make use of Semaphores to Trigger Tasks and how Interrupts can also be integrated into the multi-tasking environment. Using individual semaphores for the various led's may be an option for now, but its not an elegant one. In the next lab, we will explore how we can make use of Message Queues to send data from task to another and synchronize their behaviour.