CG2271 Real-Time Operating Systems

Tutorial 7

Q1. Look back at the code Q3 from Tutorial 6. You decide to replace the Semaphore with a Mutex. The Push Button IRQ Handler releases the mutex and the led threads acquire the mutex before proceeding with the rest of the code. The code looks as such:

```
int main (void) {

  // System Initialization
  SystemCoreClockUpdate();
  initSwitch();
  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 (;;) {}
}
```

```
void led_red_thread (void *argument) {

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

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

  }
}
```

```
void led_green_thread (void *argument) {

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

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

  }
}
```

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

  delay(0x80000);
  osMutexRelease(myMutex);

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

a) Describe the behavior of the code.
b) Explain the expected observation.
c) Look at the return values from osMutexRelease() and give a scenario on how it is possible to get the different response codes.

Q2. ***[EXAM STYLE QUESTION]***

The three led threads now have a different priority. The led_red_thread has the highest priority, the led_green_thread has the medium priority and the led_blue_thread has the lowest priority. The app_main has already created these three threads with the appropriate priority levels. The expected behavior is as such:

- At time t=0+, the GREEN LED will be ON for 1s
- At time t=1+, both the GREEN LED and the BLUE LED will be ON together for 1s
- At time t=2+, both the GREEN LED and the BLUE LED will be OFF and the RED LED will be ON for 1s.
- The above three sequences will repeat indefinitely.

Using **ONLY** Thread Flags, implement the code for the three led threads. You can assume the following thread ID's:

- redLED_ID -> led_red_thread
- greenLED_ID -> led_green_thread
- blueLED_ID -> led_blue_thread

You are not allowed to create any other threads.

The terms 0+, 1+, 2+, refer to the time just after 0, 1 or 2s.

Q3. ***[EXAM STYLE QUESTION]***

The system only has TWO Tasks, Task 1 and Task 2 that access a shared unsigned char variable "count" initialized to 0.

Task 1 Priority is equal to Task 2 Priority and there are no other tasks in the system.

Task 1 is required to increment the variable of count by 1. Whenever the count holds a value greater than or equal to 10, Task 2 will reset the variable back to 0.

Using **ONLY** Mutexes implement the code for the both the tasks.

You are not allowed to use any osDelay() calls or any other OS constructs.

Q4. ***[EXAM STYLE QUESTION]***

The system is as above in Q3 except that now **Task 1 Priority is GREATER than Task 2 Priority**.

Fulfill the same objectives as before using **ONLY** Event Flags.

You are not allowed to use any osDelay() calls or any other OS constructs.

THE END