ECEN 3002

Real Time Operating Systems Lab #6

Inter-Task Communication

Spring 2020

Objective: Explore the available communication and synchronization mechanisms available in the MicriumOS, including Event Flags, Semaphores, and Message Queues.

This lab builds upon the code developed in Lab 3 where a multi-task application was created. In this lab, the communication and synchronization services provided by the kernel will be utilized in place of the simple polling with delay method that uses global variables to share information. The GPIO interrupts that were utilized in Lab 2 will be re-introduced. The interrupt handlers will communicate with the ButtonInput task using Event Flags. The built-in kernel timer will be used to sample the touch slider. The timer callback will communicate with the SliderInput task using a Semaphore. The ButtonInput task and SliderInput task will communicate with the LedOutput task using a Message Queue.

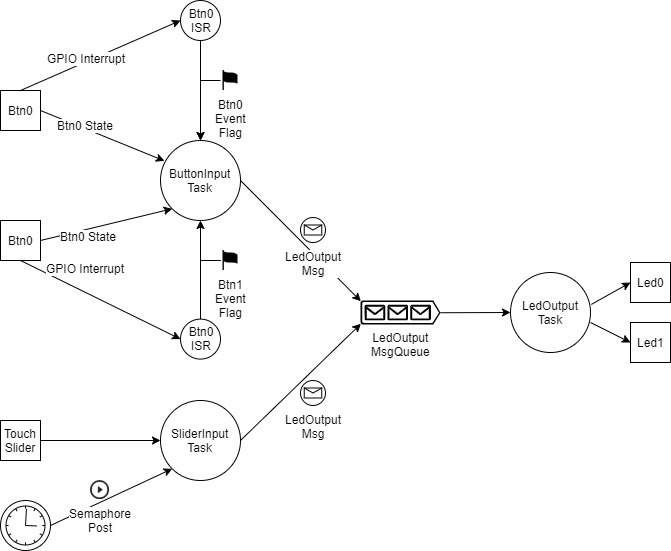


Figure 1: Inter-Task Communication

uC-Probe and SystemView will be used to analyze scheduling and system performance. The Energy Profiler will be used to compare the power consumption between the application developed for this lab and Lab 3.

References:

1. MicriumOS Documentation: <https://doc.micrium.com/display/OSUM50700/Micrium+OS+User+Manual>
   1. https://doc.micrium.com/display/TECHOV/Interrupts
   2. https://doc.micrium.com/display/TECHOV/Task+Synchronization
   3. https://doc.micrium.com/display/TECHOV/Task+Synchronization+Based+on+Events
   4. https://doc.micrium.com/display/TECHOV/Task+Message+Passing
   5. https://doc.micrium.com/display/TECHOV/Software+Timers
   6. https://doc.micrium.com/display/OSUM50700/Kernel+Interrupt+Services
   7. https://doc.micrium.com/display/OSUM50700/Using+Event+Flags
   8. https://doc.micrium.com/display/OSUM50700/Using+Message+Queues
   9. https://doc.micrium.com/display/OSUM50700/Using+the+Kernel+Timers
   10. https://doc.micrium.com/display/OSUM50700/Kernel+Event+Flag+API
   11. https://doc.micrium.com/display/OSUM50700/Kernel+Message+Queue+API
   12. https://doc.micrium.com/display/OSUM50700/Kernel+Semaphore+API
   13. https://doc.micrium.com/display/OSUM50700/Kernel+Timer+API

Preparation:

1. Review the MicriumOS Documentation cited above.

Procedure:

Part I – Interrupts, Semaphores, Event Flags, and Message Queues

1. Make a copy of the project RTOS\_Lab3\_Tasks. Rename the project to RTOS\_Lab6\_ITC.
2. Add the code from Lab2 that was placed under the compile-time switch for the interrupt-driven implementation. The compile-time switch is not needed for this lab.
3. Create the data structure for the message queue. The data structure should communicate the least amount of information at the highest level of abstraction possible for the LED Output task to make the proper decision about lighting each LED.
4. Declare an enumerated type that specifies how each bit of the event flag will be used.
5. Define static global variables for the semaphore, event flag group, and message queue.
6. Define the timer callback function.
7. Add the Micrium OS functions calls to create the periodic timer, semaphore, event flag group, and message queue resources. Note that each resource must be created prior to “posting” or “pending” on the resource.
8. Add the Micrium OS calls to “post” and “pend” on each resource (OSSemPost(), OSSemPend(), etc.). Minimizing the number of context switches should be considered when calling “post”. Each task should have a single “pend” point.
9. Add the OS call to start the periodic timer.
10. Add the code to the LedOutput task to parse the message received in the message queue and to control the LEDs.
11. Compile, link, and download the program to the board.
12. Set breakpoints to verify that all interrupts and tasks are scheduled and run as expected.
13. Perform each of the functional tests from Lab2 to verify that the external behavior of the application is the same as from Lab2.

Part II – Using Micrium µC-Probe to analyze the system

1. Launch the Micrium uC-Probe application
2. Create a new project in uC-Probe and add the Micrium OS Kernel screen to the project
3. Click Run (make sure the firmware application is also running and not stopped at a breakpoint)
4. Go to the Task(s) tab in uC-Probe
5. Verify that each of the application tasks is listed (ButtonInput, SliderInput, and LedOutput) and that each task is pending on the expected OS object.
   1. ButtonInput should be pending on the Event Flag Group
   2. SliderInput should be pending on the Semaphore
   3. LedOutput should be pending on the Message Queue.
6. Without pressing any buttons or activating the touch slider, which tasks are transitioning to/from the Read state and which tasks are blocked (hint: check Context Switch Counter)?
7. Press one of the pushbuttons. Which tasks are awakened? Record the number of context switches that occur when a button is pressed and then released.
8. Touch one side of the touch slider? Which tasks are awakened? Record the number of context switches that occur when a button is pressed and then released.
9. Do the number of context switches recorded above match your expectations? If not, check your code to determine why.
10. Save a screenshot of the application, showing the Task(s) tab.

Part III – Using Segger SystemView to analyze the system

1. Launch the Segger SystemView application

(stop the uC-Probe application if it is still running).

1. Remember to look up the address of the RTT Control Block from the generated .map file (symbol name \_SEGGER\_RTT) and enter the value in the dialog box when the record button is pressed.
2. Activate each of the inputs for several seconds and then press the stop button (red box) to stop the recording.
3. Record how often the SliderInput task is run. Does it match the timer period? Save a screenshot that shows the period.
4. Locate when the ButtonInput task is run. Note that by highlighting the task name in the timeline window, left and right arrows will appear for advancing to the previous/next location in the timeline of when the task was run. Does the scheduling of the LedOutput task appear to be synchronized with the ButtonInput task? Explain why. Save a screen shot that shows the scheduling of the ButtonInput task and the LedOutput task.

Part IV – Idle Task and Low Energy Mode

1. Start the Energy Profiler
2. Record the average power consumed at nominal conditions (no buttons pressed, slider not touched). Compare this value with the values recorded in Lab 2 and Lab 3.
3. Record the average power consumed when one and both buttons are pressed. Compare this value with the values recorded in Lab 2 and Lab 3.
4. Explain the reason for the difference, if any, in the power measurements between this lab, Lab 2, and Lab 3.

Grading:

1. 25 points equals “100%” for this lab.
2. All project files and source code must be submitted, such that results can be duplicated by the grader. ZERO score if all project files and source code are not submitted.
3. Basic Construction:
   1. Project builds without warnings and successfully executes, with each of the three application tasks running: 2 pts
   2. Proper coding and placement of OS calls to create, post, and pend on each OS object: 3 pts
   3. All functional tests from Lab 2 pass: 2 pts
4. uC-Probe Measurements:
   1. Written response to each of the questions in steps 6-10: 1 pt for each step (5 pts max.)
   2. Screenshot shows pending on OS objects as expected: 1 pt
5. Segger SystemView Measurements:
   1. Written response to each of the questions in steps 4-5: 1 pt for each step (2 pts max)
   2. Screenshot shows proper scheduling of each task in steps 4-5: 1 pt for each screenshot (2 pts max.)
6. Power Measurements:
   1. Analysis and comparison of power consumption of this lab with Lab 2 and Lab 3: 3 pts
7. Bonus points (will add to the above-listed 25 pts if the lab is turned in on time—which can push your lab#2 grade above 100%)
   1. Increase the period at which the LedOutput task services the message queue, such that multiple messages are queued at one time, as observed in Micrium uC-Probe application.
      1. Record the effect on scheduling of the LedOutput task relative to the ButtonInput task from the timeline shown in SystemView. Do the LEDs still update in the expected sequence?
   2. Return the LedOutput task period back to its original value, and gradually increase the period of the ButtonInput task. At what point are events missed (i.e., LEDs do not update in the expected sequence)?

Point Deductions for not following proper coding style: -1 points per violation

Point Deductions for Late Submission: -10 points per day