ECEN 3002

Real Time Operating Systems Lab #7

Shared Resources

Spring 2020

Objective: Explore the available mechanisms available in the MicriumOS (such as Semaphores, Mutexes, and Event Flags) for protecting data that is shared between tasks and between a task and an ISR, as well as synchronization mechanisms for notifying tasks of updated data.

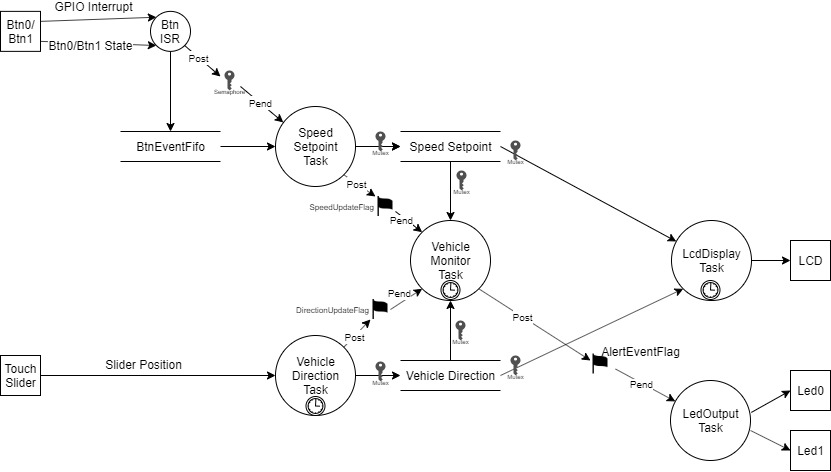


Figure 1: Vehicle Control System

In this lab, a simulated vehicle control system will be constructed, as depicted in the diagram above. Details of the system design are described in the Requirements section below.

References:

1. MicriumOS Documentation: <https://doc.micrium.com/display/OSUM50700/Micrium+OS+User+Manual>
   1. <https://doc.micrium.com/display/TECHOV/Resource+Management+Mechanisms>
   2. <https://doc.micrium.com/display/OSUM50700/Kernel+Mutex+API>
2. Review from Lab6:
   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/Software+Timers>
   5. <https://doc.micrium.com/display/OSUM50700/Kernel+Interrupt+Services>
   6. <https://doc.micrium.com/display/OSUM50700/Using+Event+Flags>
   7. <https://doc.micrium.com/display/OSUM50700/Using+the+Kernel+Timers>
   8. <https://doc.micrium.com/display/OSUM50700/Kernel+Event+Flag+API>
   9. <https://doc.micrium.com/display/OSUM50700/Kernel+Semaphore+API>
   10. <https://doc.micrium.com/display/OSUM50700/Kernel+Timer+API>

Preparation:

1. Review the MicriumOS Documentation cited above.
2. Download and run the SLSTK3402A\_textdisplay Software Example to become familiar with API for displaying text on the LCD display. Review the pertinent sections of source code to learn how to display text on the display.
3. (Optional) Download and run the SLSTK3402A\_clock Software Example to become familiar with the display of graphics on the LCD display. Review the pertinent sections of source code to learn how to draw a line on the display and how to control the font.

Design Requirements:

The following hardware will be utilized:

1. Push Buttons
   1. Button 0 will be used to increment the vehicle’s speed by 5 mph per button press/release cycle (Speed Increment Button).
   2. Button 1 will be used to decrement the vehicle’s speed by 5 mph per button press/release cycle (Speed Decrement Button).
   3. Speed should only change when a single button is pressed at one time.
2. The Capacitive Touch Slider will used to control the vehicle’s direction.
   1. Movement to the far left will signal a hard left turn
   2. Movement to the far right will signal a hard right turn
   3. Movement to the near left (of center) will signal a gradual left turn
   4. Movement to the near right (of center) will signal a gradual right turn
3. The LEDs will be used to signal alerts to the driver, for the following conditions:
   1. Speed Violation – Light LED 0, for the following warnings:
      1. Over limit, regardless of direction. Suggested limit is 75 mph.
      2. Over limit, when making a turn. Suggested limit is 45 mph.
   2. Direction Violation – Light LED 1, for the following warnings:
      1. Potential collision alert for holding the same direction for greater than a predefined time limit. Suggested limit is 5 seconds.
4. The LCD Display will be used to display the current speed setpoint in MPH and current direction (hard left, slight left, hard right, slight right) in ASCII text form. For extra credit, use a graphical display of the direction using a line to show the vehicle’s direction.

The following tasks will be created:

1. Speed Setpoint Task – Responsible for updating the Speed Setpoint data store upon user activation of the Speed Increment and Speed Decrement buttons. It is awakened by a semaphore that is posted by the Button ISR(s) when either button changes state. It notifies the Vehicle Monitor task of any change in speed using an event flag.
2. Vehicle Direction Task – Responsible for updating the Vehicle Direction data store upon any change in vehicle direction. It is awakened periodically to sample the position of the Capacitive Touch Slider. It notifies the Vehicle Monitor task of any change in direction using an event flag.
3. Vehicle Monitor Task – Responsible for checking for speed violations and direction violations and notifying the LED Output task of any change in LED state. It is awakened by an event flag set by the Speed Setpoint task and the Vehicle Direction task. It notifies the Led Output task of a change in LED state using an event flag.
4. LED Output Task – Responsible for updating the state of each LED upon notification of a change in the desired state from the event flag. It is awakened by the event flag posted by the Vehicle Monitor task.
5. LCD Display Task – Responsible for periodically updating the current speed and direction on the LCD display.

The following shared data structures will be created:

1. Button Event FIFO - data structure and accessor functions to store the state transitions of each button
   1. Upon a change in state of the GPIO, each button ISR will write the state of its respective button into the FIFO using a “write” accessor function.
   2. The Speed Setpoint task will read from the FIFO using a “read” accessor function.
   3. Each button state transition should be stored as a separate entry in the FIFO.
   4. The FIFO accessor functions should be kept in a separate file to facilitate unit testing.
2. Speed Setpoint Data - This data structure should store the current speed, as well as the count of speed increments and decrements.
3. Vehicle Direction Data - This data structure should store the current direction (left, hard left, right, or hard right), the amount of time that the current direction has been held constant, and a count of the number of left turns and right turns.

The following OS inter-task communication constructs will be utilized:

1. Semaphore
   1. A semaphore will be used to notify the Speed Setpoint task that an entry has been added to the FIFO.
2. Mutex
   1. A Mutex will be used to guarantee exclusive access to each of the shared data structures that are shared across tasks. A separate mutex should be utilized for each shared data structure (Speed Setpoint Data and Vehicle Direction Data).
3. Event Flag
   1. Event Flags will be used to notify the Vehicle Monitor Task and the LED Output Task to take action.

Procedure:

Part I – Implementation

1. The same hardware and many of the same OS constructs utilized in Lab 6 will be used for this lab. A good starting point may be to make a copy of the project RTOS\_Lab6\_Tasks and rename it to RTOS\_Lab7\_SharedResources, however you may create the project however you wish.
2. Data Structure Implementation
   1. Create the FIFO data structure and accessor functions to store the state transitions of each button. Instantiate a FIFO data structure for each of the buttons.
   2. Create the data structure to store the Vehicle Speed Data. Instantiate one instance.
   3. Create a data structure to store the Vehicle Direction Data. Instantiate one instance.
3. Task Creation
   1. Create each of the required tasks.
   2. (Optional) This step is optional in order to verify that tasks have been created and started. Temporarily awaken each task on a periodic basis using OSTimeDly(). Carefully select the value passed in the second parameter to ensure a constant task period. Verify with the debugger and/or uc-Probe or SystemView that each task is running at the desired period.
4. OS Flag definitions
   1. Declare an enumerated type that defines each of the required flag states for each of the event flags.
5. OS Object Instantiations
   1. Instantiate each of the OS resources, as required:
      1. OS\_FLAG\_GRP – 2 instances
      2. OS\_SEM – 1 instance
      3. OS\_MUTEX – 2 instances
6. IRQ Handlers
   1. Modify the IRQ handlers developed in the previous labs to put an entry into the FIFO on each interrupt event (GPIO state change).
   2. Verify with the debugger that events are added to the FIFO as expected by pressing and releasing each button and verifying the contents of the FIFO after each state change.
   3. Add the code to post to the semaphore each time that an entry is added to the FIFO.
7. Speed Setpoint Task Implementation
   1. Add the code to the Speed Setpoint task to pend on the semaphore from the button IRQ handlers (replacing any previous call to OSTimeDly() from above).
   2. Add the code to read each FIFO. Be sure to implement adequate protection measures to ensure exclusive access to the FIFO (HINT: the FIFO is accessed by an ISR and a Task).
   3. Add the code to compute the required values for the Speed Setpoint Data.
   4. Add the code to Pend and Post on the mutex to protect access to the Speed Setpoint Data.
   5. Add the code to post the event to the Vehicle Monitor Task upon change to the Speed Setpoint Data.
   6. Verify the operation of the Speed Setpoint task using the debugger
      1. Check that the task is awakened as items are added to the FIFO.
      2. Check that the task generates the correct response to each type of FIFO entry
      3. Check that the semaphore, mutex, and event flags are working as expected.
8. Vehicle Direction Task Implementation
   1. Add the OS call to awaken this task periodically.
   2. Add the call to read the current position of the Capacitive Touch Slider
   3. Add the code to compute and store each of the values in the Vehicle Direction Data.
   4. Add the code to Pend and Post on the mutex to protect access to the Vehicle Direction Data.
   5. Add the code to post the event to the Vehicle Monitor Task upon change to the Vehicle Direction Data.
9. Vehicle Monitor Task Implementation
   1. Add the code to pend on the Event Flag designated for the Vehicle Monitor Task.
   2. Upon change to the Speed Setpoint Data:
      1. Pend/Post on the mutex to guarantee exclusive access to the Speed Setpoint Data, minimizing the amount of time that the resource is locked.
      2. Read the current value of the Speed Setpoint Data
      3. Check for speed violations, per the requirements above
   3. Upon change to the Vehicle Direction Data:
      1. Pend/Post on the mutex to guarantee exclusive access to the Vehicle Direction, minimizing the amount of time that the resource is locked.
      2. Check for direction violations, per the requirements above
   4. Upon change to any of the violations:
      1. Post the appropriate event to the LED Output Task notifying the task of the activation/de-activation of an alert.
10. LED Output Task Implementation
    1. Add the code to pend on the Event Flag group designated for the LED Output Task.
    2. Update the LED hardware accordingly based on the event(s)
11. LCD Display Task Implementation
    1. Add the OS call to awaken this task periodically.
    2. Pend/Post on the mutex to guarantee exclusive access to the Speed Setpoint Data, minimizing the amount of time that the resource is locked.
    3. Read the current value of the Speed Setpoint Data.
    4. Pend/Post on the mutex to guarantee exclusive access to the Vehicle Direction, minimizing the amount of time that the resource is locked.
    5. Read the current value of the Vehicle Direction Data
    6. Update the LCD with the speed and direction information.
12. Idle Task Implementation
    1. Use the same Idle Task implementation used in previous labs to put the system in a low power state while the Idle Task is running.

Part II – Functional Testing

1. Create a functional test plan and procedure to test all functionality, per the requirements. Only a black-box test is required (verification of internal behavior, not visible to user, is not required)
2. Refer to the format provided in Lab 2.
3. The test plan should include verification of system behavior, for the following:
   1. All combinations of button presses
   2. All recognized positions on the touch slider
   3. Activation/deactivation of all alerts, by checking the state of each LED
   4. Accurate display of data on the LCD

Part III – System Analysis

1. Micrium uC-Probe
   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. Without activating any of the inputs, verify that each of the application tasks is listed and are pending/running as expected
      1. Speed Setpoint Task – Pending on semaphore
      2. Vehicle Direction Task – Awakened periodically
      3. Vehicle Monitor Task – Pending on it’s event flag.
      4. LED Output Task – Pending on it’s event flag.
      5. LCD Display Task – Awakened periodically
   6. Press one of the pushbuttons. Which tasks are awakened? Record the number of context switches that occur.
   7. Touch one side of the touch slider. Which tasks are awakened? Record the number of context switches that occur.
   8. Do the number of context switches recorded above match your expectations? Explain.
   9. Save a screenshot of the application, showing the Task(s) tab.
2. Segger SystemView
3. Launch the Segger SystemView application
4. 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.
5. Activate each of the inputs for several seconds and then press the stop button (red box) to stop the recording.
6. Check the scheduling of each task. Is each task scheduled as expected? Explain.
7. Record a screenshot of the events leading up to the scheduling of each task.

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. Implementation (10 pts: 1/2 point for each sub-heading below)
   1. IRQ Handlers
      1. FIFO is updated when, and only when, button state changes
      2. Proper use of semaphore to awaken the Speed Setpoint task.
   2. Speed Setpoint Task Implementation
      1. Proper use of semaphore to control task scheduling
      2. Interrupts are disabled when accessing the FIFO. Proper use of mutex to protect access to the Speed Setpoint Data
      3. Speed Setpoint Data is updated, as required.
      4. Proper use of Event Flag to notify the Vehicle Monitor Task of a change to the Speed Setpoint Data.
   3. Vehicle Direction Task Implementation
      1. Proper use of OSTimeDly() to awaken the task at a constant periodic rate.
      2. Proper use of mutex to protect access to the Vehicle Direction Data
      3. Vehicle Direction Data is updated, as required.
      4. Proper use of Event Flag to notify the Vehicle Monitor Task of a change to the Vehicle Direction Data.
   4. Vehicle Monitor Task Implementation
      1. Proper use of the Event Flag to control task scheduling.
      2. Proper use of mutex to guarantee exclusive access to the Speed Setpoint Data and Vehicle Direction Data. Check that the time that the resource is locked is kept to a minimum.
      3. Accurate calculation of speed/direction violations.
      4. Proper use of Event Flag to notify the LED Output Task of a required change to the state of each LED. Check that an event is only posted when necessary.
   5. LED Output Task Implementation
      1. Proper use of Event Flag to control task scheduling.
      2. LED hardware is updated accordingly based on the event(s)
   6. LCD Display Task Implementation
      1. Proper use of OSTimeDly() to awaken the task at a constant periodic rate.
      2. Proper use of the mutex to guarantee exclusive access to the Speed Setpoint Data and Vehicle Direction Data. Check that time that resource is locked is kept to a minimum.
      3. LCD is updated with the current speed and direction information.
   7. Idle Task Implementation
      1. System is placed into low power mode when Idle Task is running
4. Functional Testing (6 pts)
   1. Detailed test procedures, following the example in Lab 2.
   2. All input combinations tested
      1. Btn0/Btn1 on/off
      2. Each of four slider positions
      3. Illegal input combinations
   3. All output states tested
      1. LED0
      2. LED1
      3. LCD Display
5. uC-Probe Measurements (2 pts)
   1. Written response to each of the questions
   2. Screenshot shows pending on OS objects as expected
6. Segger SystemView Measurements (2 pts)
   1. Written response to each of the questions
   2. Screenshots shows proper scheduling of each task
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. Create unit tests for the FIFO accessor functions, using CTest. (2 pts)
   2. Instead of displaying the current direction in text on the LCD display, draw a line that shows the current direction (e.g., 0, 45°, -45°, -90°). (2 pts)

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

Point Deductions for Late Submission: -5 points per day

"${StudioSdkPath}/platform/service/sleeptimer/config"

"${StudioSdkPath}/platform/service/sleeptimer/inc"

"${StudioSdkPath}/platform/service/sleeptimer/src"

"${StudioSdkPath}/platform/common/inc"

"${StudioSdkPath}/platform/service/sleeptimer/config"

"${StudioSdkPath}/platform/service/sleeptimer/inc"

"${StudioSdkPath}/platform/service/sleeptimer/src"

"${StudioSdkPath}/platform/common/inc"