# Lab 3 RTOS with FIFO, blocking, and sleeping

**Goals**  • Extend the RTOS to include blocking, FIFO, and sleeping,

* Extend the RTOS to include a hardware triggered periodic timer.

# Starter files • Lab 3 files in tm4c123valvanoware folder.

• Your Lab 2 solution will be helpful as well.

**Background**

In this lab you will convert your spinlock semaphores to blocking semaphores and convert your mailboxes into first in first out (FIFO) queues. In Lab 2, you implemented two periodic tasks using a timer interrupt. In this lab, you will adjust the periodic task system to run using a separate hardware timer and add the ability to expand the number of periodic tasks based on global variables.

A thread is in the **blocked** **state** when it is waiting for some external event like input/output (serial input data available, LCD free, I/O device available.) If a thread communicates with other threads, then it can be blocked waiting for receive data or waiting for there to be room in the transmit buffer. Both types of blocking that will be implemented in the part of this lab. If a thread wishes to output to the LCD display, but another thread is currently outputting, it will block. We will use a blocking semaphore to implement the sharing of the display output among multiple threads.

All of these features can be implemented by changing **OS\_Wait**, **OS\_Signal, OS\_Sleep**, and your scheduler. One possible way to implement blocking semaphores modifies the TCB to include a blocked pointer that will point to the semaphore that is blocking it from running. When a thread calls OS\_Wait and needs to be blocked it will update its blocked status in the TCB and call OS\_Sleep to allow another thread to execute. OS\_Sleep calls OS\_Suspend which cases the systick interrupt and gives a full time slice to the next thread which is why we needed to change the periodic tasks to using a different timer than systick.

The FIFO queue we will implement in this lab will allow threads to produce and consume data in a more advanced manner than a mailbox as multiple items can be put in queue at once. This will be accomplished using an array and two pointers to different sections of the array. There will be a put pointer that points to the next place to store data and a get pointer that points to the next place to receive data from. There will also be a semaphore that helps to keep threads from overfilling the FIFO or trying to consume data when the FIFO is empty.

## Preparation

1. Review the Time Management chapter ($) in your book for background in how to implement the features of this lab as instructed.
2. Review the Board Support Package (BSP) to see if there are any useful features in there you might want to use (like timers).
3. Go through the os.c from Lab 2 and the os.c for Lab 3 and copy C code from Lab 2 to Lab 3 (do not move the entire file, just some C functions). Similarly, copy the SysTick ISR from Lab 2 osasm.s to your Lab 3 osasm.s. The Lab 2 SysTick ISR should be sufficient for Lab 3.

## Procedure

1. To get main\_step1 to run you will need to implement semaphores in os.c. Please note that while you must implement blocking this step does not test the blocking as it will be implemented and tested in step 2. Information on semaphores can be found in the book in section 4.2 you should read this section so you will know how to implement these functions.
2. Extend OS\_AddThreads to handle six threads, this should be a simple expansion much like when you extended from three to four threads in lab 2. Next you will need to add a blocked field to the TCB, this should be a pointer to the semaphore that is causing this thread to be blocked. Then you will need to rewrite the scheduler to check to see if the blocked field is set (not 0) and to skip that thread if it is. Additionally you will need to rewrite **OS\_Wait** to block the thread if the semaphore is less than zero and rewrite **OS\_Signal** to find the next thread in the TCB blocked by that semaphore and unblock it. Information on this step can be found in the same chapter as the last step.
3. Implement FIFO queues into OS.c. These will allow threads to communicate with one another and allow for greater functionality of the operating system. Information about how FIFO queues work can be found in section 4.3 in the book.
4. In this step you will implement sleeping as defined in OS.c. You will need to add a sleep parameter to the TCB and fill in the **OS\_Sleep** function. The sleep parameter should be a number of milliseconds the thread should sleep for, and it should be counted down using a hardware timer other than systick. In OS.c there is a function called **runperiodicevents** which we will put in code to decrement the sleep timers of all threads in the tcb that are above zero. You should also add code to **OS\_Init** to initialize the timer that will run **runperiodicevents** at 1000 Hertz. More information on this can be found in section 4.4 of the book.
5. In step 5 you will implement the periodic events. You will need to create a new TCB for periodic events that will contain a pointer to the function to run, the period at which it should run, and a counter that changes every millisecond to keep track of when it will run next. You should add code to manage the counter in **runperiodicevents**. More information on this section of the lab can be found in section 3.5 of the book.
6. Step 6 is the final step in which you test the actual main function and run the full lab to see if your operating system works. Common issues to arise here include problems with your blocking semaphores or your semaphore pointers.

## Checkout (show this to the TA)

1. Demonstrate the final system to the TA.

## Deliverables (exact components of the lab report)

1. Each lab member should submit a copy of the code to the lab 3 assignment under assignments on blackboard. Preferred submittal is in the form of a zip/rar/7z file.

## Hints

1. You should go through the procedure section in order running main\_step1 with step 1, main\_step2 with step 2 and so on until step 6 when you can run the actual main function. Be sure to change the names of each main function as you can only have one at a time.
2. Since each of the labs is built on top of the previous lab, time spent debugging this lab will greatly simplify subsequent labs. In other words, some students report that significant time is wasted during labs 4 5 and 6 because their Lab 3 OS has bugs.
3. Be sure to review how pointers in C work as their misuse can cause a lot of issues that can take a lot of time to sort out.