**Mid-Term Exam Review**

* Characteristics of traditional operating systems
  + Large, monolithic, pre-configured, non-deterministic
    - Large in that they are actually very heavy in terms of memory and LOC
    - Monolithic meaning they come in one invariable size
    - Pre-configured meaning they come with many modules to support common peripherals
* Characteristics of real-time operating systems
  + Compact, modular, configurable, deterministic
    - Compact meaning they are small, enough so to run on very limited hardware
    - Modular meaning only those components necessary will be baked/built in
    - Configurable meaning the programmer can choose which modules are included in an application
* Key components of an RTOS
  + Threads
    - Congruent to state machines, control peripherals and run application code
    - Can be dynamically created/destroyed
  + Semaphores
    - Simple form of inter-thread communication (to notify of events to react to)
  + Mutexes
    - Mutual exclusion, allows thread to monopolize use of a shared peripheral
  + Queues
    - FIFO buffer used to send data between threads
    - Can be put to sleep while waiting for something in the Queue to come up
  + Timers
    - allows a function to be scheduled at a specific interval
* Determinism vs non-determinism
  + Deterministic: Known maximum response time for peripheral inputs
  + Non-deterministic: No guaranteed maximum response time for peripheral inputs
* Latency
  + The maximum time specified by an RTOS to respond to an asynchronous event (ex: interrupt by communications peripheral)
* Preemptive vs non-preemptive (cooperative) scheduling
  + Preemptive:
    - CPU controls which thread is being run and can start/halt threads without having to yield control
  + Non-Preemptive:
    - Each thread must be a “good citizen” and yield control back to the RTOS (similar to states in bare metal state machine)
* Round-robin scheduling of tasks with the same priority
  + Tasks execute in an equal fashion around the loop
* Deciding between using an RTOS and bare-metal programming
  + Use of communications stack
    - Take advantage of relatively complex code already written for using protocols of different communications stacks
  + Complexity and flexibility of application
    - Use RTOS when expected that application will continually expand/change, or that several variations will be made of both
* Embedded vs real-time
  + RTOS Embedded
    - Anything with a needed known response time and fits the other requirements/use cases of RTOSs (comms/expandability)
  + Bare Metal
    - Response time-critical applications that do not meet the specific use cases of RTOS (smaller, no comms, narrow hardware budget)
  + Traditional OS
    - Embedded applications where it’s easier to simply use a regular OS, response time is not critical, and usually best for comms
* Scheduler analog in bare-metal programming
  + the while(TRUE) loop calling each state machine
* Task analog in bare-metal programming
  + the individual state machines (functions)
* Task priority, yielding to the scheduler
  + Scheduler will run highest priority tasks first, but this means those higher priority tasks must yield in order to allow lower priority task to run and operate
* Task stack and its significance
  + The stack of a task
* Semaphores
  + Simple form of inter-thread communication (to notify of events to react to)
* Queues, FIFOs, LIFOs
  + FIFOs
    - First-In, First-Out (Queue)
    - Similar to standing in line for a movie
  + LIFOs
    - Last-In, First-Out (Stack)
    - Similar to a stack of pancakes or plates
* Mutexes
  + Mutual exclusion, allows thread to monopolize use of a shared peripheral
* Timers
  + allows a function to be scheduled at a specific interval
* Handles
  + Used to track the location and metadata of a data structure, although it is NOT a pointer
* Blocking concepts, checking vs letting the scheduler do the work
  + Blocking allows the scheduler to decide what needs to be done
* Writing the code in a task versus in a bare-metal state machine
  + The code written in a task and bare-metal must both yield control, as they each must be good citizens
  + In a task, the state machine code is written in a while(TRUE) loop, which is not the case for the state machine code
* RTOS hazards – priority inversion and deadlock
* Priority inheritance
  + When a high priority task gets blocked while waiting for some data from a low priority task, a mid priority task might temporarily gain control since higher priority than the low, thus indirectly gaining a higher priority than the high since the high remains blocked while the mid executes
  + Scheduler handles this occurrence
* Deadlock solutions
  + A set of 2 or more tasks/threads which have interdependence on one another are all blocked indefinitely waiting for the next to provide something
  + This can be prevented by cancelling, restarting, or giving a default value after timeout to the first task/thread
* HAL, BSP
  + HAL:
    - Hardware Abstraction Layer
    - allows a computer OS to interact with a hardware device at a high/general/abstract level rather than hardware level (called from OS kernel or device driver)
    - Written by chip manufacturer
  + BSP:
    - Board Support Package
    - contains hardware-specific drivers to llow an OS (RTOS) to function in certain environments (with certain peripherals)
    - Written by board manufacturer
* Linker segments
  + IDE Stats
    - Text: code and constant data that is stored on flash memory
    - Data: the initialized data (might be associated with global/extern variables), stored on RAM
    - BSS: uninitialized data, ends up in RAM as well
* Re-entrant functions
  + Functions that can be called by multiple tasks/threads without changing any global/static variables (stateless)
  + Can only modify if passed as a parameter (in other words, a helper function)
  + Generally: NO STATICS TO BE RE-ENTRANT
  + Example of NON-re-entrant: strtok()
* Review lesson notes
* Review quizzes 1-5
* Review assignments 1-5

**Final Project Definition**

ST Micro B-L475E-IOT01A development board peripherals:

* STM32L475 MCU – 80MHz Cortex-M4 core, FPU, MPU, 1MB flash, 128KB RAM
* Serial flash – 64Mb (8MB)
* WiFi – 802.11 b/g/n
* Bluetooth Low Energy (BLE) – V4.1
* Sub-GHz RF – 915MHz ISM band
* Near Field Communications (NFC) Tag – ISO/IEC 14443 Type A and NFC Forum Type 4
* Digital microphones (2)
* Temperature and relative humidity sensor
* Barometric pressure sensor
* 3D magnetometer
* 3D accelerometer + 3D gyroscope
* ToF (time-of-flight) sensor
* Green LED
* Pushbutton
* Potential feature via an Arduino-compatible shield – Graphical LCD

**Class Notes**