**Quiz 1 Review**

* Non-determinism vs determinism
* Latency – worst case response to an asynchronous event
* Traditional OS – large, monolithic, pre-configured, non-deterministic
* RTOS – compact, modular, configurable, deterministic
* Bare metal versus RTOS decision – use of a communications stack, complex application
* Bare metal embedded programming is efficient and responsive
* RTOS building blocks – threads, semaphores, queues, mutexes, timers
* Scheduler – manages threads, bare metal equivalent is the while (TRUE) loop
* The bare metal equivalent to a thread is a state machine function
* Preemptive versus non-preemptive/cooperative
* Is macOS, Linux, or Windows suitable for embedded applications?

**RTOS Concepts**

Here is a recap of the main RTOS concepts and constructs. In the descriptions of semaphores and queues, “inter-thread communication” can mean communication between two thread and communication between a thread and an ISR.

* **Threads** – Threads are similar to the state machines in bare-metal embedded software development. Threads can control peripherals and threads can implement application code. The advantage of threads over state machines is that you can **prioritize** threads and you don’t have to break down your algorithms into state machines. However, you should write your code to be **event driven**, to ensure that high priority threads occasionally yield control to lower priority threads.
* **Semaphores** – Semaphores are a simple form of inter-thread communication. They allow a thread to notify another thread about an event that it needs to react to.
* **Queues** – A queue is a first-in, first-out (FIFO) buffer for inter-thread communication of a stream of data. If a thread tries to read an empty queue, it will suspend until something is placed in the queue. A timeout can be associated with reading or writing to a queue, such that an attempt to read from an empty queue or an attempt to write to a full queue can unblock after the timeout period, even if the state of the queue doesn’t change.
* **Mutexes** – A mutex (from “mutual exclusion”) is a mechanism that allows a thread to monopolize the use of a shared peripheral, such as a serial port. For example, if a thread needs to print out a command prompt on the serial port, it does not want another thread to print characters on the same serial port in the middle of the command prompt. If a thread locks a peripheral with a mutex and another thread requests a mutex for that same peripheral, the scheduler idles the second thread until the first thread releases the lock on the peripheral.
* **Timers** – A timer allows a function to be scheduled to run after a specified period. The timer can be restarted to allow the function to be called periodically, such as having a thermostat take the room temperature every minute.

The RTOS maintains a list of threads that are running, idle, or halted. It also prioritizes which thread will run next. The part of the RTOS that performs these functions is called the **scheduler**.

There are two main methods for managing the threads:

* **Preemptive** – The CPU controls which thread is running and can start and halt threads without the threads having to yield control. This requires a CPU with protected modes to run the scheduler, so that it can stop a thread by itself (preempting the execution of the threads).
* **Non-preemptive or Cooperative** – Each thread must be a “good citizen” and yield control back to the RTOS. This is similar to the state machines in bare-metal programing, in that they should execute quickly and return (thus, yielding to the while (TRUE) loop, allowing other state machines to run).

**Zephyr Project Overview Video**

<https://www.youtube.com/watch?v=JEpY_ETJ_jE>

**Nordic Semiconductor Zephyr Introduction**

<https://developer.nordicsemi.com/nRF_Connect_SDK/doc/latest/zephyr/introduction/index.html>

**Class Notes**