EE4930 Advanced Embedded Systems

Section 011, Winter 2019/20

Draven Schilling

1-25-20

HW1 – Bare metal vs RTOS webinar

# Summary

The webinar started off with a brief description of what bare metal means. Suggesting that you are essentially are given a blank flash space where it’s up to you to write everything from scratch. Bare metal seems to be useful for extremely low latency applications that are simple and/or only provide one or few functions. It’s only when you have many applications that your microcontroller starts running into problems with bare metal not scaling as well.

Next, he defined what an RTOS is. He described it as a minimal latency solution which uses task scheduling and task management to allow concurrency and a microcontroller to scale better with many functions. An RTOS helps manage shared resources by offering mutexes/semaphores and pipes/message queues. The RTOS emphasizes preemption for allowing higher priority tasks to meet their deadlines. Other features an RTOS provides is that it is generally portable (unless in contains assembly code) such that it can run on different processor architectures. It also efficiently uses the processor such that work is done for a short bursts of time very intensely then puts the processor back to sleep. Given the current memory technology, the size of a basic RTOS is very minimal so it can be easily implemented in almost any system, even the very small and limited ones. Finally, one last important fact about an RTOS is that at its core, it only provides a kernel, scheduler, and synchronization techniques. It’s up to the programmer and/or external libraries to provide device drivers, file systems, debugging techniques, and networking capability.

In the final part, he reviews some of the common implementation techniques of an RTOS. We learn that the way tasks are setup are similar to minimal separate main programs each with their own main loop and private stack such that context switches can pick up directly where a task left off. Also. We learn tasks are queued based on their state (ready, running, and blocked). Finally, there was some discussion about basic scheduling algorithms such as round robin and priority based for picking which task could run next.

# Significant Thing I learned

Having taken the CE operating systems class and having experienced some development of a round robin scheduler, a lot of the ideas an RTOS uses seems familiar. One new thing I did learn was just how minimal an RTOS is. He suggested it run with about 4kb of space and only includes the kernel, scheduler, and synchronization techniques. This is quite a downgrade from say base linux which seems to have at least some basic device driver support, a built in file system, and some networking capabilities. I guess in an embedded case it makes sense that these features have been pruned in order to make the RTOS as fast and efficient as possible with knowledge that these core features of “primary OS’s” may not be as necessary.