Lab 6-7: Bare Metal TCBs/FreeRTOS

Blake Williams

11/10/2024

**Summary**

The goal of Lab 6 and 7 is to build upon prior stop light labs. In Lab 6 the goal was to implement a bare metal version of multitasking/task scheduling without the use of a fully implemented RTOS. Lab 7 builds upon Lab 6 by introducing FreeRTOS and allowing our bare-metal task switcher to be converted into a full RTOS system. This includes features like task scheduling, mutex state transitions, and many more.

**Design and Implementation**

In Lab 7, the largest change from Lab 6 is how both handle task management and scheduling. This change comes from the difference in Lab 7s design utilizing a RTOS environment while Lab 6 relies on a Bare Metal approach to task scheduling. For example, in the bare metal code in Lab 6 a TCB is manually defined with a struct containing a function pointer, data pointer, and task Ready flag. The bare metal Lab 6 utilizes a custom scheduler to iterate over each task and check its taskReady flag. If a taskReady flag is triggered then a timer interrupt occurs and allows the task to execute.

In contrast to Lab 6, the FreeRTOS implementation utilizes the built in TCBs and scheduler. Since the TCBs are managed by the OS (freeRTOS) this allows for task creation, suspension, delay, and resuming without the need to manually toggle flags. Each task runs independently of one another, and are able to utilize self-contained timing with the vTaskDelay() function. In comparison to Lab 6s Task scheduler being a while (1) loop that iterates through all the tasks and executes based on the taskReady flag. Lab 7 can utilize the FreeRTOS kernel in order to schedule tasks. For example when all of the tasks are created, a vTaskSuspend() is called for each of the tasks except for the supervisor. Then when the kernel is started with vTaskStartScheduler(), the kernel will execute the SupervisorTask which will check the state variable, and enter the state corresponding to the value inside. The initial state is always GREEN, so inside the GREEN state a vTaskResume() is called on the Green Task Handle, which then executes the Green Task. Once a button is pressed, then the Green Task changes the state variable, suspends itself, and allows for the next task to run. All of this task scheduling being handled by the FreeRTOS kernel.

**Conclusion**

Overall both labs helped to introduce the idea of Tasks and scheduling, and different ways on how they can be implemented. Along with how Tasks and scheduling function, Lab 7 showed how FreeRTOSs kernel/scheduler can be utilized to replace the Bare metal design given in Lab 6. This design lead to less code needing to be written as a lot of components that were manually defined in Lab 6 were abstracted out into the features of FreeRTOS.