Jackson Paull

JHP2539

14 February 2025

ECE 381L - RTOS

Lab 2 Report

1. **OBJECTIVES**

Lab 2 was focused on the development of the RTOS Kernel, its scheduler and thread synchronization techniques. The goal is to minimize the overhead for switching between a variable number of concurrent threads, some of which are periodic, some of which are triggered by an aperiodic source (e.g. a switch on a GPIO pin), and some of which are continuous.

I implemented a simple round robin scheduler (no priority) and blocking semaphores.

To support blocking semaphores and a round robin scheduler, I implemented a set of general linked-list functions to handle the structures maintained by the OS.

1. **HARDWARE DESIGN**

None for this lab

1. **SOFTWARE DESIGN** - *Note: See GitHub, tag ‘lab2-release’ for source code files.*

Spinlock Implementation

Void OS\_Wait(Sema4Type \*semaPt){

DisableInterrupts();

while(semaPt->Value <= 0) {

EnableInterrupts();

ContextSwitch();

DisableInterrupts();

}

semaPt->Value -= 1;

EnableInterrupts();

}

void OS\_Signal(Sema4Type \*semaPt){

int i = StartCritical();

semaPt->Value += 1;

EndCritical(i);

};

The spinlock works by continuously testing the semaphore value and cooperatively switching if the thread can’t acquire it. This leads to a lot of starvation.

Round-Robin Implementation

I implemented scheduler.c which maintains its own listing of TCBs and can control how it wants to schedule things separately from how the OS will run them. Currently, the scheduler maintains a doubly linked list and simply swaps to the next thread in line (Sketches 2 and 3). Threads can be added to this list either immediately (scheduler\_schedule\_immediate(thread)) or fairly (scheduler\_schedule(thread)). The immediate scheduling is for cases where a high priority (interrupt) task might launch several time-sensitive yet lower priority tasks.

1. **MEASUREMENT DATA**

A diagram of a diagram

AI-generated content may be incorrect.

Sketch 1. TCB list before and after OS\_Launch

A diagram of a diagram

AI-generated content may be incorrect.

Sketch 2 TCB list before context switch Sketch 3 TCB list after context switch

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| FIFOSize | TIMESLICE *ms* | DataLost | Jitter *µs* | PIDWork |
| 4 | 2 | 0 | 0.9 | 8986 |
| 8 | 2 | 0 | 0.9 | 8983 |
| 32 | 2 | 0 | 0.9 | 8982 |
| 32 | 1 | 0 | 1 | 8959 |
| 32 | 10 | 0 | 0.9 | 9009 |

Table 1. Performance data with varying system specs

|  |  |  |  |
| --- | --- | --- | --- |
|  | DataLost | Jitter *µs* | PIDWork |
| With debug | 0 | 0.9 | 8986 |
| Without debug | 0 | 0.9 | 9019 |

Table 2. Performance data with and without debugging instruments

**A screenshot of a computer

AI-generated content may be incorrect.**

Figure 2.1 Cooperative thread switching

**A screenshot of a computer

AI-generated content may be incorrect.**

Figure 2.2 Preemptive thread switching (zoomed in)

**A screenshot of a computer

AI-generated content may be incorrect.**

Figure 2.3 Preemptive thread switching (zoomed out)

**A screenshot of a computer

AI-generated content may be incorrect.**

Figure 2.4 Context switch time

Here in figure 2.4, we see a context switch time of roughly 2 microseconds.

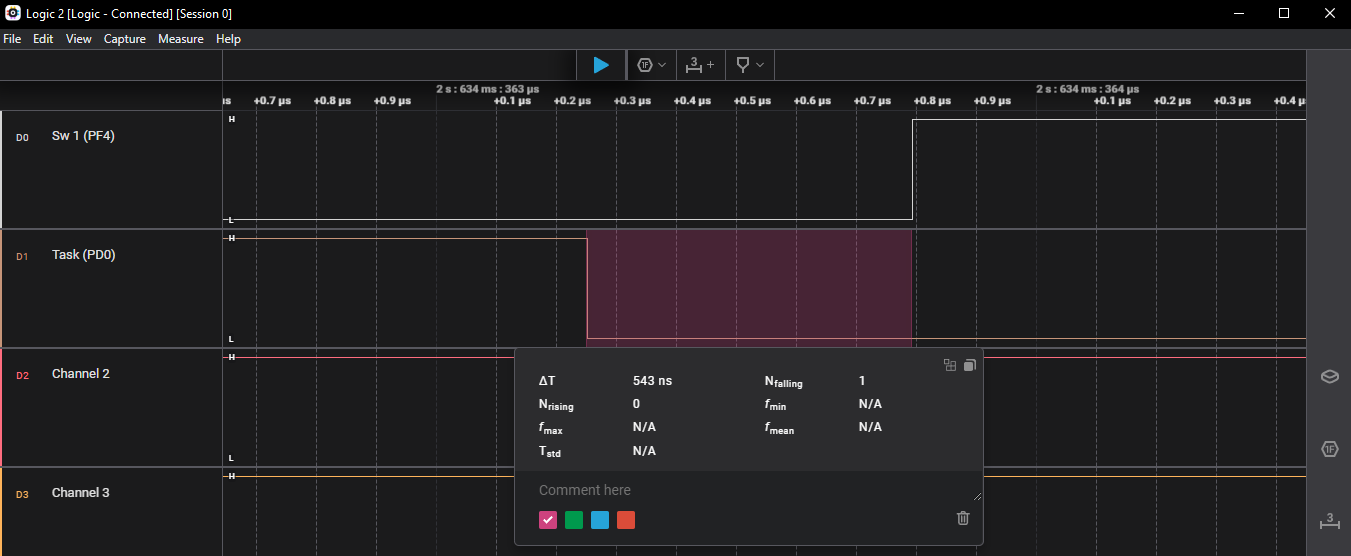


Figure 2.6 Context switch time

Here in figure 2.6, we see roughly 550ns of delay from the switch being pressed to the task executing.

1. **ANALYSIS AND DISCUSSION**
2. *Why did the time jitter in my solution jump from 4 to 6 μs when interpreter I/O occurred?*
   1. If the interpreter UART interrupts (for its fifo copying hardware to software and vice versa) have a higher priority or equal priority to the periodic DAS, then the “worst case” jitter would be one where the DAS needs to run once the UART interrupt has just started. In this case the jitter would be the normal jitter plus the computation time needed for the UART. 2 microseconds is a reasonable amount of time for the UART to move hardware FIFOs to software, hence the increased jitter.
3. *Justify why Task 3 has no time jitter on its ADC sampling.*
   1. The ADC sampling is triggered through a separate hardware timer which is not impacted by any context switches or other threads. After the data is ready, an interrupt is triggered and user code can be run, which in this case just puts the data into the software FIFO for the consumer to read.
4. *There are four (or more) interrupts in this system DAS, ADC, Select, and SysTick (thread switch). Justify your choice of hardware priorities in the NVIC?*
   1. Systick should almost be the lowest priority (with PendSV even lower than Systick) so that a context switch never returns to an interrupt context.
   2. DAS should be triggered periodically, so as to minimize jitter it should be able to preempt other tasks and therefore should have a high priority (though not necessarily the highest if another task is more critical. In this case there is no such task).
   3. The ADC task just puts data into the software fifo, so long as we place the data into the FIFO before the next ADC data is ready, we will be fine, hence it can have a lower priority than DAS.
   4. The select button only needs to be as fast as the human eye can view changes to the screen, hence it can have the lowest priority aside from SysTick.
   5. The final priorities are DAS > ADC > Select > SysTick > PendSV
5. *Explain what happens if your stack size is too small. How could you detect stack overflow? How could you prevent stack overflow from crashing the OS?*
   1. A stack which is too small is at risk of one thread writing into another threads stack space.
   2. Stack overflow can be detected by maintaining a MAGIC value at the top/base of the stack. If the value is modified, it must be that another thread pushed onto a full stack directly beneath the corrupted stack (MAGIC at bottom), or that the current thread pushed too much onto its own stack (MAGIC at top).
   3. If a MAGIC value is maintained at the top of a stack, threads can be unscheduled if they have overwritten their magic value. When a context switch triggers, the OS can audit the stack to ensure no corruption, and dump errors / unscheduled offending threads.
6. *Both Consumer and Display have an OS\_Kill() at the end. Do these OS\_Kills always execute, sometime execute, or never execute? Explain.*
   1. These OS\_Kill instructions will always execute, except for a few edge cases. This is because the threads have a body with a fixed number of instructions (bounded while loop) and hence, given enough time, will eventually complete their computation.
   2. The cases in which the OS\_Kill will not execute include
      1. Bugs in Mailbox (semaphore) implementations
      2. Crashed/stalled producer task
7. *The interaction between the producer and consumer is deterministic. What does deterministic mean? Assume for this question that if the OS\_Fifo has 5 elements data is lost, but if it has 6 elements no data is lost. What does this tell you about the timing of the consumer plus display?*
   1. Deterministic means that a fixed input sequence will lead to a fixed output sequence (even if the computation of that output sequence is intractable apriori!)
   2. If a 6 element FIFO loses no data, and a 5 element FIFO loses some data, then we know that the producer thread will produce at most 6 elements of data every period, and that the consumer will consume at least 6 elements of data every period.
8. *Without going back and actually measuring it, do you think the Consumer ever waits when it calls OS\_MailBox\_Send? Explain.*
   1. No, the Consumer and Display threads will be scheduled in a round robin. The number of threads initially being run is 8, hence the period of these threads can be estimated with 8\*2=16ms, with some of those (namely the switch tasks) being blocked. Assuming that the Display thread can move through an iteration of its while loop within its 2ms timeslice, then it will be scheduled roughly 20ms later (or sooner) and wait on the next piece of data. Since the consumer sends data every 160ms, we can assume that the Display will always read the data before the consumer sends a new piece of data.