CMPE 663 Project 3

Bank

Andrei Tumbar

Instructor: Wolfe
TA: Nitin Borhade

By submitting this report, you attest that you neither have given nor have received any assistance (including writing, collecting data, plotting figures, tables or graphs, or using previous student reports as a reference), and you further acknowledge that giving or receiving such assistance will result in a failing grade for this course.

Your Signature: _____

Analysis/Design

This project looked at implementing a simulated bank using the features provided by FreeRTOS. The bank was simulated by providing accurate and scaled timings for transactions between customers and bank tellers. Random variables that drove the probabilities of certain actions were generated using the hardware RNG over a uniform distrobution across given boundaries.

Simulation

The design of the bank involved four main tasks in addition to two status tasks. Each teller is given its own RTOS task and they are all waiting on the same queue. FreeRTOS's scheduler works by scheduling the task with the highest priority that is ready to execute. If multiple ready tasks have the same priority, the scheduler will alternate between them. This means that if multiple tasks with the same priority are to block due to a queue, the task that has not been serviced over the longest time interval will be scheduled next. In other words, the teller that has not seen a client for the longest time, will be given priority to pick from the queue. This allows a fair distrobution of customers to each of the three tellers.

The fourth task running on the RTOS involves adding customers to the queue. It will generate a customer in a random time interval and queue the customer. When the work day is up, this task will self suspend.

Metrics

An important aspect of this project was to provide a mechanism for reporting simulation metrics at the end of the work day. Because all metrics could be fit into either a maximum, minimum, or arithmetic average, a `Statistic` data structure was created to model any metric that was asked for. During the relevant events, the teller or the customer creation tasks would report a metric which would in turn be added to its corresponding statistic structure. When reporting the totals at the end, averages could easily be calculated from the sum and count of the statistic and min/max were kept track of as each event was reported.

Status

The two status tasks running at lower priority simply handle writing the current teller status to the uart and writing to the seven segment display. They are running at a lower priority for two reasons. First being that the simulation accuracy would be slightly impacted if they were running at the same priority as the tellers. Secondly, putting these tasks at a lower priority allows them to run during all of the idle time that the CPU had. This proved to be fast enough to provide accurate status while still maintaining accuracy.

Grad-portion

The grad-portion of this assignment involved writing to the seven segment display. Because the seven segment display works by keeping a single command in its memory, each digit in the number had to be continously written to the display to get a proper reading. The task performing this action runs a constant loop absorbing as many cycles are possible to create a clear image.

Test Plan

Testing the code was fairly straighforward. The UART status task was updated in realtime and therefore showed most of the errors on the spot. The uniform distrobution of the RNG could be verified by making sure the minimum and maximum were close to the specified boundaries as well as the average being halfway between the two. The seven segment display was tested using the given customer entry rates as well as increased rates designed to fill the queue more.

Project Results

The software behaved as expected by the problem statement. All random numbers generated expected results and the simulation ran in the expected amount of time. The metrics reported at the end of the simulation showed that the parameters controlling the simulation as well as the design of the bank were correct.

Lessons Learned

This project explored creating an embedded system on top of FreeRTOS. I learned how to use the API reference to quickly implement the components I needed in this realtime system. Task scheduling priority and preemption became clear when expirementing with task attributes during development. The most major issue that I ran into during development was a hardfault inside the teller. I created an efficient way to debug hardfaults (and other faults) by creating an assembly subroutine to extract the core registers saved when the hardfault interrupt was triggered. This allowed me to view the instruction and register values of the problematic code. The problematic code ended being related to a libc library call `vsnprintf` not being thread-safe and therefore causing issues within FreeRTOS.

Below is an example of a hardfault occuring when overwriting part of the task control block (FreeRTOS's structure for keeping track of tasks) causing the context switcher to fault out:

```
Hardfault:
  r0: 0x20018000
  r1: 0xff0a0000
  r2: 0x00000000
  r3: 0x20000c68
 r12: 0x0000000d
  lr: 0xfffffff9
  pc: 0x8009000
xpsr: 0x2100000b
Assertion failed /home/tumbar/git/CMPE663/Core/Src/fw.c:90 : 0 && "Hard-fault error pc:", 134254592
```

Figure 1: Example of hardfault handler

Looking at the program counter in Figure 1, we can find the appropriate assembly instruction in the generated ELF binary. To do this, LLVM's objdump tool was used to decode the binary.

```
08008ff8 <SVC_Handler>:
 8008ff8: 5f f8 0c 30   ldr.w   r3, [pc, #-12]
 8008ffc: 1a 68         ldr     r2, [r3]
 8008ffe: 10 68         ldr     r0, [r2]
 8009000: b0 e8 f0 4f   ldm.w   r0!, {r4, r5, r6, r7, r8, r9, r10, r11, lr}
 8009004: 80 f3 09 88   msr     psp, r0
 8009008: bf f3 6f 8f   isb     sy
 800900c: 4f f0 00 00   mov.w   r0, #0
 8009010: 80 f3 11 88   msr     basepri, r0
 8009014: 70 47         bx      lr
```

Figure 2: Decoded instruction at faulty program counter

In this particular instance, the task control block's stack pointer was moved to an invalid address therefore causing the kernel to fault out when attempting a context switch. This specific instruction will save the tasks core registers not explicitely saved by an ISR onto the tasks stack. In the future, this mechanism will be useful in quickly determining issues related to programming errors such as the one caused by non-threadsafe use of `vsnprintf` in this exercise.