Home / Study Guide Comments

4 Section: A buffer-based driver (page 255)

- **Bug in the code** (`mainUartInterruptBuff.c`), page 256
  - There are bugs in the interactions between the interrupt-handler and the task `uartPrintOutTask()`. This includes bugs in the mutual-exclusion scheme for shared data, and a bug in the sequencing of events.
  - **Problem #1**:
    - `startReceiveInt()` alters data that that can be concurrently referenced and altered by the interrupt handler and by the USART. These concurrent operations are problematic and appear to have one or more bugs. At the least, it is difficult to determine that these concurrent operations do not have bugs.
    - Two variables are used to specify the state of the byte-receive process. They are both set in `startReceiveInt()`, and setting the pair of them should be an atomic operation (relative to the interrupt handler). However, it's not:

```
rxInProgress = true;
rxItr = 0;
```

      - The interrupt handler could be run after the first assignment statement, and before the second one. A potential consequence is that `rxItr` can be incremented in the interrupt handler, then set to zero in `startReceiveInt()`. This can cause a byte to be lost.
      - If `xSemaphoreTake()` times-out, there can potentially be an interrupt-handler call that sets `rxItr` to `expectedLen+1`, and `rxData[expectedLen]` to a non-zero value. This call would occur after `rxInProgress` is set to true, and before `rxItr` is set to zero. It would add an extra unintended character to the output string. That extra character will appear in all future calls to `SEGGER_SYSVIEW_Print((char*)rxData)`.
      - Each of the assignment-statements themselves might not be atomic. If an assignment is partially completed when an interrupt occurs, the altered variable's value would effectively be undefined then.
    - In `startReceiveInt()`, four statements are used to set-up the USART and interrupt-handler. There might be two problems with those statements, though it's speculation:
      - The USART and interrupt-handler is set-up the first time `startReceiveInt()` is called. However, those four statements are run again for each iteration of the `while{}` loop. It's not clear what the effects are of running those instructions for an interrupt-handler and USART that is already set-up and running.
      - Also, each of the four statements might not be atomic. However, they can be run while the USART is receiving data. For each statement, if altered data is partially altered when an interrupt occurs, the data's value would effectively be undefined.
  - **Problem #2:**
    - The interrupt-handler can issue `xSemaphoreGiveFromISR()` between when `xSemaphoreTake()` times-out, and when `startReceiveInt()` is run. If the full buffer was received, the message issued would be incorrect. The message would state zero bytes were received.
  - **Solution**:
    - I didn't see an easy fix for these problems, and it appeared refactoring was needed.
    - A fixed version of `mainUartInterruptBuff.c` is here:

https://github.com/jimyuill/embedded-systems-projects-01/blob/main/book--Hands-On-RTOS/chapter-10--mainUartInterruptBuff--fixed.c

    - The fix uses two semaphores, instead of one, to control the interactions between the interrupt-handler and the task `uardtPrintOutTask()`.

- **Bug in the code** (`mainUartInterruptBuff.c`), page 258
  - **Problem:**
    - `SEGGER_SYSVIEW_Print((char*)rxData)` does not display the whole buffer if the buffer contains a null-terminator before the end of the buffer.
    - For example, the string transmitted by UART4 could be: `"data from uart4\0"`
    - When the transmitted-string is received by `USART2_IRQHandler()`, the first character put in the buffer might not be first byte in the transmitted-string. For example the full buffer could be like this:
      - `" uart4\0data from \0"`
    - The string provided to `SEGGER_SYSVIEW_Print()` is displayed in the SystemView app. However, the app will only display characters up to the first null-terminator in the provided string.
  - **Solution:**
    - One solution is for `USART2_IRQHandler()` to translate received zero-bytes to a printable character that is not in the transmitted string, e.g., "!"
    - This is done in the fixed version of `mainUartInterruptBuff.c`, cited earlier.

- **Clarification**, page 261
  - The version of `mainUartInterruptBuff.c` from the repo is different than what is shown in the book. The code from the repo includes the task `wastefulTask()`.
  - When running under SystemView, `wastefulTask()` shows the effect of `xHigherPriorityTaskWoken` on the scheduler.