    - In the Terminal window, the events labeled "2" and "3" do not correspond with the events labeled "2" and "3" in the Timeline window. For each window, the time between those events is very different.
        - In the Terminal window, the time between events "2" and "3" is about 800ms.
        - In the Timeline window, the time between events "2" and "3" is about 300ms.

- **Bug** in the book, page 186
    - There's a minor error in this sentence:
        - *"Marker 1 indicates TaskB didn't receive the semaphore within 500 ms. Notice there is no follow-up execution from TaskB – it immediately went back to taking the semaphore again."*
    - This part of the sentence seems incorrect: "*there is no follow-up execution from TaskB*".
    - There is "follow-up execution from TaskB". When TaskB doesn't receive the semaphore within 500 ms, it sets the red LED on.

- **Bug** in the book and code (`mainSemPriorityInversion.c`), page 190-193
    - Problem
        - The program `mainSemPriorityInversion.c` does not result in "priority inversion", as intended.
        - `TaskB` is intended to spin so much that `TaskA` is often prevented from getting the semaphore in time. However, `TaskB` doesn't spin enough for that to happen. `TaskA` never times-out.
    - Analysis
        - Two tests were run that show this:
            - The program was run for 7.5 minutes, and the SystemView log was saved to a CSV file.
                - `TaskA`: failed to get semaphore: 0 times; semaphore taken: 1310 times
                - `TaskC`: failed to get semaphore: 32 times; semaphore taken: 1279 times
                - `TaskB`: iterations: 15,632
            - The program was modified, so log messages contained a count of the times the semaphore was taken, and failed to be taken. The program was run for 20 minutes.
                - `TaskA`: failed to get semaphore: 0 times; semaphore taken: 3900 times
                - `TaskC`: failed to get semaphore: 56 times; semaphore taken: 3517 times
                - `TaskB`: iterations: 41,000
        - The spin-loop in `TaskB` does not run long enough to cause priority inversion:
            - In `TaskB`, for each iteration, there is a random delay between 10 and 25 ms, and a spin-loop that runs a random amount between 3 and 8 ms.
                - From my testing, `lookBusy(94000000)` is about 1 second.
                - So, `lookBusy(StmRand(250000, 750000))` would be 3 to 8 ms.
            - `TaskC` will hold the semaphore for at most 196 ms
                - Its delays are 172 ms.
                - `TaskB` could run (spin) concurrently for at most 24 ms: 8ms before TaskC's first delay, 8ms before the second delay, and 8ms after the second delay.
                - It's very unlikely that `TaskB` would run concurrently for 24ms, or even 20ms.
            - `TaskA`'s timeout for the semaphore request is 200 ms. But, it will wait on `TaskC` holding the semaphore for at most 196 ms. So the timeout never occurs.
    - Solution
        - The problem is fixed by making `TaskB`'s spin-loop run longer.
        - It was changed to run 18 to 28ms: `lookBusy(StmRand(1692000, 2632000))`
        - So, if `TaskC` has the semaphore, and it waits on `TaskB`'s spin-loop just once, it will wait 18 to 28 ms for it. And TaskC will hold the semaphore for 190 to 200 ms.
        - Testing results for 7.5 minutes are:
            - `TaskA`: failed to get semaphore: 1335 times; semaphore taken: 865 times
            - `TaskC`: failed to get semaphore: 544 times; semaphore taken: 950 times
            - `TaskB`: iterations: 7600 (rounded)

- **Clarification**, page 190-191
    - `TaskA` and `TaskC` use functions to set the red LED on and off (`RedLed.On()`, `RedLed.Off()`). For these functions to work properly here, the functions must be atomic