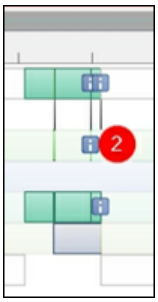
- `vTaskDelay(SemRand(10,30));` // in the book, pg 196
- Also, that description is inconsistent with the Timeline, as the Timeline does not show `TaskA` returning the mutex and immediately taking it.
  - Instead, the Timeline shows about a 5ms delay between when `TaskA` returns and takes the mutex.
  - The Timeline shows that `TaskA` runs briefly at approximately -35ms, just before point (2). Point (2) is at approximately -30ms. A screenshot is below.
  - It appears `TaskA` returns the semaphore at -35ms. This is apparent because `TaskC` is waiting on the semaphore then, and at that point, `TaskC`'s row is shaded grey. The grey shading indicates `TaskC` is marked Ready for execution. The mutex it's waiting for is now not taken.
  - However, at (2), `TaskA` takes the mutex. And, at (2) `TaskC` is no longer marked Ready for execution (`TaskC` is no longer shaded grey). This is because the mutex `TaskC` is waiting for is taken by `TaskA`.



- **Bug** in the book, page 195
  - In the Terminal and Timeline windows, the corresponding events shown have different timestamps.
  - I'm guessing they are not the same events. For each event in the Terminal window, its corresponding event in the Timeline window has a greater timestamp. The only way I know for that to happen is by using SystemView's "reference" time-feature.
  - Regardless, using different time-frames in the two windows is confusing.

- **Additional info**, page 195
  - This additional-info shows how a mutex is allocated, when two different tasks have requested it, and one task has higher priority than the other. In that case, the higher-priority task is given the mutex, even if the lower-priority task requested the mutex first.
  - For example, task X has higher priority than task Y.
    - Task X holds the mutex, and blocks
    - Task Y requests the mutex and blocks
    - Task X releases the mutex
    - Task X requests the mutex, and it is given the mutex
  - The behavior is specified in the FreeRTOS doc, *Mastering the FreeRTOS Real Time Kernel*. It's in the section, "Mutexes and Task Scheduling".
    - *If two tasks of different priority use the same mutex, then the FreeRTOS scheduling policy makes the order in which the tasks will execute clear; the highest priority task that is able to run will be selected as the task that enters the Running state.*

3 Section: Using Software Timers (pg 200-208)

- **Typo** in the book, page 202
  - "*oneShotCallBack() will simply turn off the blue LED after 1 second has elapsed*"
  - It should state 2.2 seconds (the value calculated by: `(2200/portTICK_PERIOD_MS)`)

- **Bug** in the book and code (`mainSoftwareTimers.c`), pages 204-205
  - Problem:
    - SystemView can't record the events shown in the book, for `mainSoftwareTimers.c`
    - With this code, it's not possible to start SystemView's Record feature in time for it to record the green LED being turned-on for the first time, nor to record the blue LED being turned off.
    - The figure on page 204 shows SystemView recording those events, but I don't think it's