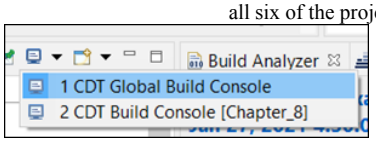
Chapter 8: bugs, clarifications, and tips

Chapter 8: Protecting Data and Synchronizing Tasks (pg 177)

1 Section: Using Semaphores (pg 178-193)

- **Tip**, page 178
  - In running `Build All`, here's a way to check for build errors:
    - Before running `Build All`, clear the console
    - After running `Build All`, check the Console messages for errors.
      - In the Console frame, click on the icon "`Display Selected Console`", then select "`CDT Global Build Console`". This will show the console messages for all six of the projects.



      - In the console, search for "errors" to check the six projects were built correctly

- **Tip**, page 181
  - I found it very helpful here to take the time to use SystemView to analyze `mainSemExample.c` and to get skill in using SystemView.
  - SystemView tips:
    - In the Timeline window, set the zoom level: `right-click : Zoom : View 200us`
    - To go-to an event of interest in the Timeline: in the Event or Terminal window, click on the event

- **Additional info**, page 181
  - This info provides analysis of how the scheduler works, and its relationship to `SysTick ISR` and the `Idle` task. This info is from analyzing `mainSemExample.c`, and from the book and Stack Exchange.

  - `SysTick ISR`:

741	20.799 055 005	BlueTaskB	Task Block	Reason = 4
742	20.799 063 435	Idle	System Idle	
743	20.799 962 134	SysTick	ISR Enter	Runs for 3.769 us
744	20.799 965 903	SysTick	ISR Exit	Returns to Idle
745	20.800 962 014	SysTick	ISR Enter	Runs for 3.000 us
746	20.800 965 014	SysTick	ISR Exit	Returns to Idle
747	20.801 962 028	SysTick	ISR Enter	Runs for 2.912 us
748	20.801 964 940	SysTick	ISR Exit	Returns to Idle

    - Every 1 ms, the `SysTick ISR` is run
    - Info from the book on processing ticks:
      - *"...the scheduler is still running at predetermined intervals. No matter what is going on in the system, the scheduler will diligently run at its predetermined tick rate."*, page 47
      - *"All of this switching [between tasks] does come at a (slight) cost – the scheduler needs to be invoked any time there is a context switch. In this example, the tasks are not explicitly calling the scheduler to run. In the case of FreeRTOS running on an ARM Cortex-M, the scheduler will be called from the SysTick interrupt...*
      - *A considerable amount of effort is put into making sure the scheduler kernel is extremely efficient and takes as little time to run as possible. However, the fact remains that it will run at some point and consume CPU cycles. On most systems, the small amount of overhead is generally not noticed (or significant), but it can become an issue in some systems."*, page 45
    - Apparently, part of what the `SysTick ISR` does is call scheduler-code, to perform scheduling functions. This can be seen in the SystemView screenshot below. This analysis involves some speculation, so it might not be fully correct:
      - The `SysTick ISR` starts running at event 1053.
      - The `SysTick ISR` runs scheduler-code, and this scheduler-code runs under the `SysTick ISR`.
      - At event 1054, the scheduler-code has detected that `BlueTaskB`'s delay has ended, and the scheduler-code moves `BlueTaskB` to the Ready state.
        - Tasks that are marked ready for execution are displayed with a light grey bar until their execution starts, e.g., as here, for `BlueTaskB`.
      - At event 1055, the scheduler-code moves `BlueTaskB to the` Running state. The `SysTick ISR` ends, and `BlueTaskB` is started.
    - How long the `SysTick ISR` runs (some speculation is involved here):
      - It appears that when the `SysTick ISR` does not have much to do, it runs for 2-4 micro-seconds. This is seen in events 1051-1052 and 1059-1060.
      - The `SysTick ISR` can run longer when it has more scheduler-code to run. This is the case from event 1053 to 1055, which is about 10 micro-seconds.
      - Since `SysTick ISR` is run every 1 ms, its system overhead appears to be around 0.2% to 1%