Chapter 9: bugs, clarifications, and tips

Chapter 9: Intertask Communication (pg 209)

- **Clarification**, page 211:
  - The queue-size is 2. However, I don't think it will ever hold more than one element, due to the algorithm and task priorities.

- **Bug** in the code (`mainQueueCompositePassByValue.c`), page 211-212:
  - It would be better if this line was in the body of the if-statement:
    - `vTaskDelay(nextCmd.msDelayTime/portTICK_PERIOD_MS);`
  - If `xQueueReceive()` fails then `nextCmd.msDelayTime` does not get set on this iteration of the loop.

- **Clarification**, page 214
  - The queue-size is 8. However, the queue-size only needs to be 1 here. It would work the same, in operating the LEDs.

- **Additional info**, page 219
  - Info about `uxMessagesWaiting`:
  - It is in the struct `QueueDefinition`, which is defined in FreeRTOS's `queue.c`:
    - `C:/projects/packtBookRTOS/Middleware/Third_Party/FreeRTOS/Source/queue.c`
  - Some additional variables in the struct:

```
List_t xTasksWaitingToSend;     /*< List of tasks that are blocked waiting to post onto this queue.  Sto
List_t xTasksWaitingToReceive;  /*< List of tasks that are blocked waiting to read from this queue.  Sto

volatile UBaseType_t uxMessagesWaiting;/*< The number of items currently in the queue. */
UBaseType_t uxLength;          /*< The length of the queue defined as the number of items it will hold,
UBaseType_t uxItemSize;        /*< The size of each items that the queue will hold. */
```

- **Bug** in the book, page 220
  - There are some incorrect statements under "*A long queue was used for commands:*".
  - The queued commands and latency are attributed to both: "*low-priority task receiving*" and "*long queue length*".
  - However, for this program, the queued commands and latency are not due to the receiving task having lower priority than the sending task. If the sending task is given higher priority than the receiving task, the commands will queue in the same way.

- **Tip**, page 221:
  - It can be beneficial to write this code one's self, before looking at the provided program.

- **Additional info**, page 221
  - Info on how data is copied to and from a queue:
  - `xQueueSend` and `xQueusReceive` just use `memcpy()`, internally.
  - The copy is in FreeRTOS's `queue.c`, in the functions: `prvCopyDataFromQueue()` and `prvCopyDataToQueue()`

- **Bug** in the book and code (`mainTaskNotifications.c`), page 227
  - Problem:
    - There's a bug in the example program `mainTaskNotifications.c`, in its use of `ulTaskNotifyTake()`.
    - `ulTaskNotifyTake` is not the correct notify function to use here, although the code does work.
    - `ulTaskNotifyTake` is intended for use as an alternative to a semaphore, but this code is not implementing a semaphore. Also, in using `ulTaskNotifyTake` here, it has to be made to do something that isn't relevant to the example program.
    - Using `ulTaskNotifyTake` is not as straight-forward and it adds a bit of unnecessary complexity.
  - Solution:
    - The intended API to use here is `xTaskNotifyWait()`:
      - From the FreeRTOS manual:

```
xTaskNotifyWait( uint32_t ulBitsToClearOnEntry,
                 uint32_t ulBitsToClearOnExit,
                 uint32_t *pulNotificationValue,
                 TickType_t xTicksToWait );
```

      - How it would be used in `mainTaskNotifications.c`

```
xTaskNotifyWait(0, 0, &notificationvalue, portMAX_DELAY);
```