CS466 Lab 3 -- Simple Queues, Producer/Consumer, Serial IO and gdb debugging
Due by Midnight Friday 2-7-2020.

!!! Must use provided lab format!!!

You may hand in a team lab per, Individual or Two members to a team only.

Overview:

- This lab is similar in thread structure to Lab 2, I suggest that you start with a working lab2 solution.
- This lab has a lightweight programming task
- This lab adds Serial debug output so that you can provide text back to a remote console.
- This lab introduces the gdb debugger so that you can halt, single-step and break your code. It allows you to examine and modify the memory and registers (while broken).
- This is not a 100% complete step-by-step instruction for the lab; you will have to bridge some gaps. If you get stuck send me an email or ask in class or lab.

Lab Preparation:

- Review your Lab 2 thread structure.
- Read over the FreeRTOS API documentation for queues (http://www.freertos.org/a00018.html). Specifically look at the documentation for xQueueCreate(), xQueueSend(), and xQueueReceive() in detail. Like semaphores you have to create the queue structure before you can call the queue api operations (Create, Send, Receive).

Objective:

- To work a very simple Queue example.
- To add Serial debugging IO to your code.
- To analyze assert failures in your code.

Lab Work

1. □ Copy your Lab02 code/files to this lab03 directory. Name the main program producerConsumer.c. You will need to modify the Makefile to adjust the main code file name. Delete all the interrupt and semaphore code that dealt with switches, keep the green-led task.

2. □ Rename your green thread to 'heartbeat' and have it run at idle priority. This thread should always run and keep the green LED blinking at around 1 Hz.

➢ Lab-report-questions-1,2: How much effort did it take to get this basic heartbeat-only task working? As the first project that was not provided as a compiling sample how would you summarize the effort?

3. □ Look at my example code serialStubs.c and move all the #ifdef USB_SERIAL_OUTPUT sections to your code and see if you can get the emulated serial working. This will require a program on your host that can talk to serial and may require sudo. Once working this will give you a method to report data. You should be able to receive data sent with UARTprintf over the virtual serial port.

a) In order to get the serial communtications to build and link you will need to add #include "utils/uartstdio.h" to your includes as well as add the file uartstudio.c to your build.

b) The uartstudio.c file lives in the TivaDriver directory tree but by modifying VPATH in the Makefile (look how the freertos queue.o module is referenced).

c) You can use any serial communications device to receive serial output from the tiva board. On Linux the device shows up as /dev/ttyACM0 (that's ACM<zero>), I use Kermit with the following config file below;

d) There are confusing interactions about when the serial communications disconnect and they are different on various platforms.

e) I use Kermit on Linux and OSx and putty on Windows.

```
miller@xw8400 [~]
$ kermit k_ACM0
?SET SPEED has no effect without prior SET LINE
Connecting to /dev/ttyACM0, speed 115200
 Escape character: Ctrl-\ (ASCII 28, FS): enabled
Type the escape character followed by C to get back,
or followed by ? to see other options.
----------------------------------------------------

miller@xw8400 [~]
$ cat k_ACM0
set line /dev/ttyACM0
set speed 115200
set carrier-watch off
set flow none
connect
```

4. · Create a Queue with 20 entries before you start the scheduler.

a) For now we will just be passing a uint32_t through the queue.. Often you would use a structure as your queue entry element. If it's a large queue element you may use a pointer.

5. · Add a consumer thread and a producer thread. These are identical in structure to your heartbeat thread. Pass the handle of the queue to each thread as part of its thread parameters, do not use global variables. (to do this you will need to cast the queue handle to a (void *) and pass as pvParameters)

➢ Lab-report-question-3: What the heck, what does casting a value to be a `(void *)` do? What is it generally used for.

6. · Make the consumer thread block on queue receive so and momentarily light the blue led whenever a message is received. Use a frequency that works visually.

7. · Make your producer thread block for a random delay then send a message, If the queue is full assert(). Try to hit about 10 messages per second for a starting average rate. If you assert you are probably holding the LED on too long and overflowing the queue.

8. · Move all of your producer characterization data to a single structure definition and initialize a struct with the data in main. Instead of just passing the queue handle as you did with the consumer thread, Pass a pointer to the struct to the producer.

9. · Add a second producer that will also insert messages into the queue. When the consumer thread receives a consumer2 message light the red LED momentarily. Also time the random message generation rate to average about 10 Hz. Assert if the queue send fails. This second producer thread should use the same function as the first.

10. · Re-arrange the priorities so that the consumer thread has a lower priority then the two producer threads. If the program does not assert after a while, increase the period of the LED indication in the consumer to slow it down. Why do I expect an assert here?

Debugging Tools (UARTPrintf, assert, and gdb).

11. · in the top of the assert.c file add a printf so that assert conditions report where the assert failed.

```
#ifdef USB_SERIAL_OUTPUT
   UARTprintf("Assertion Failed: %s at %s::%d\n", assertion, file,
   line);
#endif
```

12. · After the UARTprintf Verify ort add the two lines in your assert code.

a) `taskENTER_CRITICAL();`
b) `taskDISABLE_INTERRUPTS();`

13. · Add some code so you fail an assert() in a producer task. In theory any other threads should have been stopped as well by the two lines added above.

a) Do the other Threads stop executing as well?

b) Why are the other threads stopping

Try with the two lines above in the code or commented out. Is there any difference between the to cases.

14. · Take time to review the file `CS466_lab03_Build_and_Debug_Basics.pdf`.

15. · Open an additional window and type the largish openocd command line to get the server running.

16. · Create the .gdbinit file documented in the .pdf file. You will also need to create the global .gdbinit file in your $HOME directory,

17. · Start GDB as in the .pdf file.

a) Take a look at the GDB cheat sheet file and try some commands that don't actually execute any code

1. Display the first 16 32-bit values stored in ROM (at address 0x00000000)
2. Display the processor registers
3. Display what breakpoints are set

b) What data is in the first 8 bytes of memory?

18. · Note that if you make and try to flash the image from the Linux command line the flash will fail because openocd has reserved the ICDI USB interface to the Tiva.

a) Make a small modification to your program

b) At the (gdb) prompt type the command s

```
(gdb) make
(gdb) reload
(gdb) c
```

c) Describe what happened.. (assuming this instruction is correct)

19. · What do the following commands do?

```
a) (gdb) reload
b) (gdb) I b
c) (gdb) I r
d) (gdb) l
e) (gdb) mr
f) p <variable>
g) p {<var>, <var>, <var>}
```

h) `s`

i) `n`

20. · Take some time to set some breakpoints and step through your code.