CS 431-1 Embedded Systems

HW 6 (5 is skipped)

Opened: Wednesday, 29 September 2021, 12:00 AM
Due: Thursday, 23 December 2021, 11:59 PM
To do: Make a submission

This hw uses the actual hardware in digital lab (EA lab).
Get yourselves acquainted with the platform by reading the docs at https://mbed.com and playing around with the examples already there.

Your next hw assignment is to build a device that changes the brightness of the 4 LEDs on the board by receiving commands from the serial port. The assignment itself is fairly simple but you're asked to implement in a certain way. Submissions that "do what's asked" but do not meet these criteria will get 0 (e.g. if you use an RR architecture).

For starters you should go into mbed LPC1768 | Mbed and mbed NXP LPC1768 Getting Started - Handbook | Mbed for a quick start. When you grab yourselves a board, make sure your PC has the proper serial communication drivers and your mbed has the latest firmware.

Since we're working with hardware now, we should be using mbed OS 6.x (6.15 is the latest as of now) Introduction - Introduction to Mbed OS 6 | Mbed OS 6 Documentation. I've had students before who were reading the wrong documentation (e.g. mbed OS 2); make sure you're reading the right docs.

The tools that you can use are listed here, my preference is still the online version (Introduction - Quick start | Mbed OS 6 Documentation). Here's the quick start link for it: Importing and compiling the code - Quick start | Mbed OS 6 Documentation

You're asked to use RTOS architecture and hence the RTOS you'll be using is the mbed OS. An introduction to mbed OS - RTOS is here: RTOS overview - API references and tutorials | Mbed OS 6 Documentation. Following that here's the memo about the thread-safety (re-entrancy) of libraries: Thread safety - API references and tutorials | Mbed OS 6 Documentation

In this lab you're not allowed to disable and enable interrupts but use Semaphores etc. (Semaphore - API references and tutorials | Mbed OS 6 Documentation). You can use other safe data sharing mechanisms if you wish (Full list of RTOS API is here: ConditionVariable - API references and tutorials | Mbed OS 6 Documentation)

1. (15)(printf is not thread-safe. And so you shall implement a task that encapsulates printf. Task should block on a synchronisation mechanism of your choice. When there is data, it should unblock and print the received string. Ensure data is protected.
2. (10)Heartbeat task: activate a task that was blocked and print (via your printf task) a message that says "Alive for {X} seconds", X is the number of seconds since your device started. The task should activate every 5 seconds. *Hint: Set up a timer that releases a semaphore that this task is blocking on every 5 seconds.*
3. (15)Also implement a serial receiver task, which should block waiting for a single character on pc-serial (USBTX, USBRX). Upon receiving a new line, it should transfer this new line to a command task (more on this later). There are 2 ways to do it:
    1. Using BufferedSerial and using read function in blocking mode.
    2. Using UnbufferedSerial or BufferedSerial and attaching your own callback.
4. (10)Demonstrate you are able to save sequences by printing them back when enter is pressed (new-line received); do NOT print back inside ISR *(nor inside serial-rx task above depending on your implementation)*. Use the printf encapsulation task you implemented. Ensure your receive buffer is protected, ensure string-to-be-printed is protected. So, when a new line is received, the new-line should also be sent to your printf task.
5. (10)Implement an LED-controller task; this task should activate every 100ms, read 4 configuration values (e.g. from a struct) and set the LED brightnesses with respect to these values. Obviously, access to these 4 values should be protected.
6. (15)Implement a command task which blocks on reception of a line from serial receiver task. Upon unblock; this task should compare the line for predefined commands given below. Invalid parameters or commands should make the line to be completely ignored (e.g. set led 5 0.5, e.g.2. hello world). The led setting shouldn't be done directly but the values should be written to aforementioned global configuration struct. Power values are between 0.0-1.0
    1. set led $lednum:int $brightness:float
        1. sets led brightness, prints a confirmation of cmd
        2. e.g. set led 0 0.4
    1. get led $lednum:int
        1. prints (via print encapsulation) the brightness of led
        2. e.g. get led 0

2. forward $pow:float
    1. sets led1 & led3 to $pow, and led2 & led4 to 0, prints a confirmation of cmd
    2. e.g. forward 0.6
3. backward $pow:float
    1. sets led2 & led4 to $pow, and led1 & led3 to 0, prints a confirmation of cmd
    2. e.g. backward 0.7
4. left $pow:float
    1. sets led1 to $pow, led3 to $pow*0.5, and led2 & led4 to 0, prints a confirmation of cmd
    2. e.g. left 0.6
5. right $pow:float
    1. sets led2 to $pow, led4 to $pow*0.5, and led1 & led3 to 0, prints a confirmation of cmd
    2. e.g. right 0.9

1. **No task should employ sleep, nor they should spin (while(1)) uncontrollably. All tasks should be blocking until an event occurs.**
2. **Remember that you must set proper priorities to your tasks, this is not a general-purpose operating system environment, and your tasks won't be time sharing, there will be no fairness etc.**
3. **Final hint: you should use separate structures for protecting shared-data and for signalling (as described in lectures).**

Submit different C files for each part. I really suggest uploading different files to be honest, make it easy for everyone. Remaining points will be awarded on architecture+code quality.

REMARKS: Unlike the previous hw; programming busy-waits are NOT allowed and on top using sleep is also not allowed. Please use Timers/Timeouts/Tickers or any other method of non-busy waiting. Also note that you are only allowed to use official mbed libraries and NOT the user/community libs/codes.

If you feeling extra adventurous after all this work feel free to take a look at "Events" demo which uses a function-queue-scheduling architecture and incorporate function-queue-scheduling into your RTOS tasks (i.e. hybrid architecture. Hint: each RTOS task should have its own EventQueue which is dispatched forever. This only makes sense for Heartbeat and LED-controller tasks.

If you got any questions, ask them in HW 6 Discussion (5 is skipped)

Submission status

Submission status	No attempt
Grading status	Not graded
Time remaining	2 days 15 hours
Last modified	-
Submission comments	Comments (0)

Add submission

You have not made a submission yet.

◄ Realtime Applications vs. Linux: is it feasible yet?

Jump to...

HW 6 Discussion (5 is skipped) ►