Milestone 4

ESOS32

"Standing in the middle of the road is very dangerous; you get knocked down by the traffic from both sides."
– Margaret Thatcher (1925-2013)

THIS MILESTONE has you begin using ESOS to develop multitasking applications on your MCU. You will need to adjust your software build process to accomodate the ESOS tree based on the examples in the repository. Finally, you will develop several solutions to a classical thread synchronization problem.

4.1 Hardware

There are no new hardware requirements for this lab milestone.

4.2 Software

4.2.1 Software preparation

Check out the latest from the public github ESOS32 repository containing the 32-bit version of ESOS at `http://github.com/jwbruce/esos32`. Become familiar with structure of the code tree. Pay special attention to the structure of the user applications under examples/stm32l4ocm3.

Doxygen for the ESOS can be found at `http://jwbruce.info/esos32`

4.2.2 Build process and examples demonstration

Compile and run the examples: *echo*, *semaphore1*, *semaphore2*, and *swtimer* on your hardware. The standard serial terminal connection in ESOS is 57.6k/8/N/1. The TA may ask you to demonstrate one or more of these sample applications on your hardware. Feel free to modify these applications to get an understanding for ESOS and its application structure.

4.2.3 Repo preparation

From this milestone forward, you will use a github repository for your team collaboration and software development. Furthermore, your milestone submissions will be "tagged" in the repository for grading by the TA and/or Dr. Bruce.

1. Using the tree structure and the repository location provided by the TA or Dr. Bruce, start populating your team repo.
2. Configure your client to access and manage your github repo.
3. Maintain all code and documentation for the remainder of your lab in this repo. The TA will provide instructions for each milestone as to the location of the files.

4.2.4 Your first ESOS application – `m4_app`

Recreate the exact same functionality as the application `m3_app` previously developed using ESOS. Your application shall not use interrupts directly. The ESOS communications service will handle the low-level UART transfers. You should use either an user task or a software timer to control the LED. Your new application, named `m4_app` should be as robust as possible, i.e. the systems should continue to operate as close to normal in the face of increasing numbers and serverity of failures.

4.2.5 Task Sychronization App

Read the Chapters 1-3 in A.B. Downey's *The Little Book of Semaphore, 2/e*. The *semaphore1* and *semaphore2* ESOS example apps above demonstrate mutex and rendez-vous. Using these apps as an example, develop a semaphore demonstration app that demonstrates *one* other basic sychronization patterns (multiplex, barrier, reusable barrier, queue) in Downey's chapter 3. Name your ESOS application `multiplex`, `barrier`, `rbarrier`, or `queue`, as appropriate. Tasks can report their status, needs, arrival at synchronization gates, etc. via statements to the ESOS serial out communicaitons channel.

To mimic reality as much as possible, your milestones needed synchronication should be created, execute, request, etc. as appropriate at random times. Your code should be easily adjusted to change the number of tasks involved in the pattern. Furthermore, your application should flash the Nucleo LED2 every 500 ms in a task as a system "heartbeat".

4.3 In-lab Checkoff

- Demonstrate the proper operation of your program `m4_app` by first sending a paragraph of text from your PC terminal program to your Nucleo. Your terminal program should display the returned "encrypted" text. Save this text.
- Next, pressing button B1, send the encrypted text from your laptop terminal program to the Nucleo. Your terminal program should receive the decrypted text from the Nucleo. The text should match the text in the first step.
- Demonstrate the different LED flash period functionality.
- Demonstrate the proper operation of your task synchronization application. Explain how your program operation is correctly. Demonstrate a variety of number of tasks.

4.4 Submission

Create a folder in your team repo named `m4` to hold this milestone's deliverables. In the milestone folder, store your application code `app.c` along with any project-specific header files. Include the appropriate `makefile` and any unit/system test files created. Also, create a `readme.txt` to explain how to build your application[1] and simple user instructions. In the milestone folder, also include the following items:

- well written, well commented, MCU code[2] for ARM-Cortex application
- any unit test or test bench files used to verify your code created for this milestone
- bonus GUI application, if applicable
- updated design review forms and software metrics

Email the TA on or before the specified due date to "submit" your deliverables. The TA will refresh his/her local copy of your tree, build the applicaiton code(s) from your source, and run the application on his/her hardware. The TA will not modify code, adjust values or defaults, or in any way modify your code. If your tree does not build "out of the box", then your team's submissions will be graded as "not working".

[1] The TA should be able to simply "pull" from your team repository, and type `make` to fully build everything in your team submission.

[2] Your MCU firmware must adhere to the *C* language coding standards.