Laboratory assignment 1

Custom Scheduler Design with ARM-based Mote.

First, the team will be reviewing and evaluating performance of a legacy code. Next, you need to propose and implement changes that will improve performance of the example real-time system. Your custom RT scheduler needs to achieve sub-millisecond scheduling resolution. Interact with your "customer" (instructor) to obtain essential information.

Objectives:

1) Familiarize with the programming environment and ARM processor (STM32F4 series core -> ARM Cortex M-4 CPU core)
2) Understand example code and its basic "scheduling algorithm
3) Setup code to measure RT performance
4) Define real-time requirements and parameters for existing tasks and jobs
5) Design a framework for RT Scheduler on ARM processor
6) Support execution of periodic tasks, sporadic jobs, and aperiodic jobs
7) Demonstrate performance improvements

Tasks:

1) Setup IDE environment:
    a. IDE: CubeIDE for STM32 (**ver. 1.2.0** is setup in EECH 210 lab)
        i. https://www.st.com/en/development-tools/stm32cubeide.html
    b. STLink/Virtual COM:
        i. https://www.st.com/en/development-tools/stsw-link009.html (main one!!)
        ii. https://www.st.com/en/development-tools/stsw-stm32102.html (possibly needed for Win7/8)
        iii. https://www.st.com/en/development-tools/stsw-link007.html (debugger firmware upgrade if needed)
2) Compile and run the code in debugger (IDE has built-in, graphical debugger)
    a. Open project in your IDE, build the project, debug using step-in/over
    b. Utilize info about the dev board at: https://www.st.com/en/evaluation-tools/nucleo-f446re.html

3) Study the "sch_basic.c" code and related header files (e.g. "rts\labs\lab_1\Drivers\STM32F4xx_HAL_Driver\Inc\stm32f4xx_hal_uart.h" and "rts\labs\lab_1\Drivers\STM32F4xx_HAL_Driver\Src\stm32f4xx_hal_uart.c" for serial/UART library)

4) Setup one or more of the pins (e.g. Port A pin 0) to provide timing signals for measurements. Then, toggle the pin(s) at specific locations in the scheduler code. Measure the following:
    a. Execution time for each loop function (inside the FOR loop in scheduler) and the entire FOR loop (inside the "void sch_loop(void)" function). Observe differences in execution time of each function, and jitter in their execution.
    b. Time between subsequent starts of the "void sch_loop(void)" function (entire period of the main loop). Measure average, and variation of the period.

5) Devise and implement method of evaluating the accuracy of time-based execution by scheduler (i.e. WHILE loop inside "void sch_loop(void)" function).
    a. Measure how much the actual execution START time differs from the intended start time – in 1ms ticks.
    b. Determine if any "sch_callback_funcs" starts or ends later then 1ms after its desired (release) time?

6) Describe and discuss: what scheduling algorithm is used? What are possible issues with the current scheduler? What has to be done to make it more robust and predictable from RT perspective?
    a. Identify any other changes that might be needed to improve performance and responsiveness of the code.
    b. Propose changes to the code such that the scheduler can implement one of standard clock driven or priority-driven schemes – e.g. EDF, LST.

7) Define real-time requirements and parameters for existing tasks and jobs
    a. Based on earlier measurements, propose set of tasks and jobs with their real-time (RT) properties (e.g. laxity, deadlines, periods, execution time, priority)
    b. Verify with customer that your understanding of the system's RT requirements and properties is correct

8) Propose overall architecture of your RT scheduler (e.g. what and how are you going to use timers, interrupts
    a. Use CubeMX and relevant documentation for STM32 processor and associated libraries (HAL) – which timers are already used? How to setup the timer for sub-millisecond (sub-ms) resolution?
    b. Define how periodic and sporadic jobs will be executed to guarantee deadlines (e.g. how you plan to use interrupts and their priorities, what analysis is needed)

9) Design and implement necessary data structures to keep track of tasks and jobs in the system and support RT scheduling
    a. Consider that you may need to add new tasks/jobs in the future
    b. You can assume non-preemptive tasks/jobs/system

10) Evaluate performance you achieved and compare with the initial results from Task 5
    a. Demonstrate what improvements you achieved (measure and present appropriate metrics)
    b. Identify 3 major benefits of your solution over original code

Deliverables:

1) Group roster (CANVAS)
2) 1st mid- update report/lab meeting (*at least tasks 1-3*)
3) 2nd mid-update slides and in-class presentation (tasks 4-8)
4) Final Report on CANVAS and in-class presentation

Source Code:

Source code is available on gitlab.com website:

- https://gitlab.com/mzawodniok/rts/-/tree/lab1_2021/labs/lab_1
- https://gitlab.com/mzawodniok/rts.git (branch "lab1_2021")

It is strongly suggested to branch from that GIT repository – keep a link to the repository to be able to "pull" updates.

Additional references:

- https://os.mbed.com/platforms/ST-Nucleo-F446RE/
- https://electronics.stackexchange.com/questions/331996/stm32-hal-implementing-uart-receive-interrupt
- https://letanphuc.net/2015/02/stm32f0-tutorial-gpio-blinking-led-cubemx-keil-source-insight/
- https://tortoisegit.org/