CSCE 432/4301: Embedded Systems
Spring 2020

Project 1 (10% - Group size: 2)

The objective of this short project is to develop a cooperative scheduler for embedded systems. This scheduler is similar to the function queue scheduler introduced in the lecture. Your implementation must comply with the following specifications/API:

1. All ready to run tasks (functions) are stored in the ready queue.
2. The ready queue is sorted based on the tasks priorities
3. A task is a just a function that returns no data and accepts no arguments.
4. To insert a task to the ready queue call `QueTask()` and pass two arguments:
    a. Pointer to the function (function name) that implements the task and
    b. Task's priority.
5. Supports 8 priority levels
6. A task can be inserted into the ready queue by
    a. Other tasks (`QueTask`),
    b. ISRs (`QueTask`), or
    c. Itself (`ReRunMe`).
7. `Dispatch()` API is used to remove the highest priority task from the queue and run it.
8. The system is initialized by a single call to the `Init()`function. This function creates and initializes all needed data structures.
9. A task my enqueue itself by calling `ReRunMe(0)`. If `ReRunMe()` is called with a non-zero argument, the task will be inserted to the ready queue after a delay specified by the argument. For example, `ReRunMe(5)` inserts the task into the ready queue after 5 ticks. To implement this functionality, another queue (delayed queue) is introduced to store the delayed tasks. This queue is sorted based on the delay/sleeping time (descending order). Once the sleeping time of the head of this queue expires, the task is removed from the delayed queue and inserted into the ready queue. The sleeping time of the tasks in the delayed queue is updated (decremented by 1) every tick.
10. The tick is 50 msec. Use the SysTick Timer to generate the tick interrupts.
11. Your code must be developed using STM32CubeMX and Keil µVision to run on the Nucleo-32 board you have.

The following code shows a simple application (has only one task implemented using the function `TaskA()`) that uses the dispatcher:

```
#define TRUE 1
void TaskA(){
        // do something here
        toggleLED();
        // Rerun again after 10 ticks (500 msec)
        ReRunMe(10);
}

int main(){
        Init(); // initialize the scheduler data structures
        QueTask(TaskA);
        while (TRUE) {
                Dispatch();
        }
        return 0;
}
```

Finally, you need to develop two small applications to demonstrate your scheduler:

- Ambient temperature monitor: Read the ambient temperature using a sensor every 30 sec. Produce an alarm (LED flashing) when the temperature exceeds a threshold. The threshold is set using a text command sent to the embedded system from a PC over an asynchronous serial link.
- Parking sensor: Produce a sound that reflects how close is the car from the nearest object. A buzzer will be used to produce beeps and the duration between the beeps reflects how far is the object. The object distance will be determined by the ultrasound sensor.

Submission:

- Before April 11, 2020 11:59 PM
- The submission will be done through a public GH repo (the url shall be submitted to BB)
- The repo must contain a comprehensive readme file (consider it the project report). The readme file should discuss the implementation (APIs details) and should walk anyone through the process of building and using the scheduler. Also, it has to document the two sample applications.
- Grade break down:
    - Scheduler implementation meeting all the requirements: 30%
    - Scheduler unit tests: 20%
    - Readme file (documentation): 20%
    - 2 demo applications: 30%
    - Inconsistent development activities (as shown by GH commits): -20%