Functions:

void init_function(**void**):

This function enqueues all the tasks before dispatching and initializes the global variables of the task.

void dispatch(**void**):

This function calls the function pointer located in the first element of the queue of the ready tasks. In this function, we can also enqueue a task if another task is supposed to call it. It also calls another function delay_queue_check() to check on the delay queue (explained in details right below)

void delay_queue_check(**void**):

This function loop on the delay queue (which contains the tasks that will be reran their delay ticks pass) to check if there are any functions that are ready to be reran, and also decrements the timer of each task queued in delay_queue( );

void ReRunMe(**int**):

This function is responsible for queueing any functions in the “delay_queue( );” that will be re-executed again after certain number of ticks (which is passed as a parameter to the function)

void ready_enqueue(**int**):

This function is equivalent to “QueTask( )”. The function takes the number of the task that needs to be queued. The function checks which task number is received and based upon this number, the equivalent task is inserted in the ready_queue in its right order. The function checks first if the ready_queue is empty. If so, the task in inserted in the first index. Otherwise, the function loops on the busy elements in the queue and compare their priorities to the to-be-inserted task’s priority. Once it finds the right index, the elements in this index and after it are shifted by one to the right, and the new task is inserted in the right index.

void ready_dequeue(**void**):

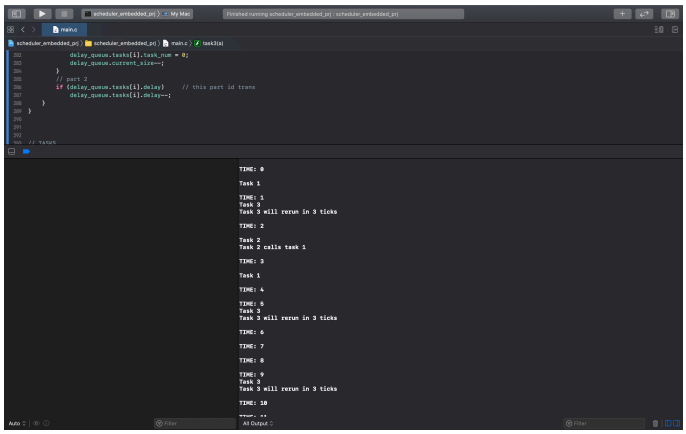
This function is responsible for dequeuing the task after executing it. The finished task is always in the beginning of the queue. Therefore, the function shifts all the elements in the ready_queue to the left by one.

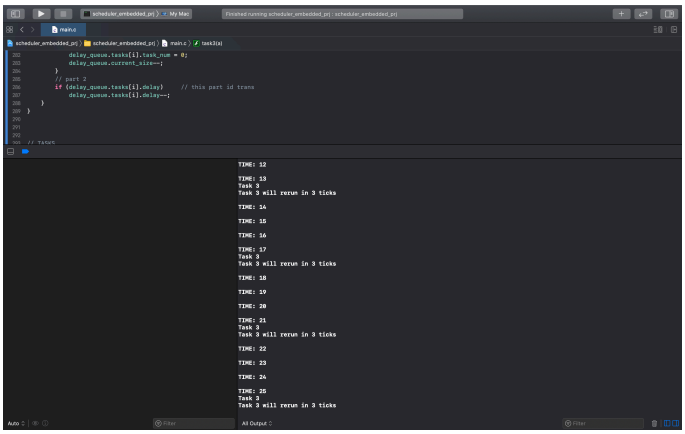
Sequence of Code:
First, the init_function is called in the main so that all the global variables are initialized and the ready_queue is filled. Then, in a loop of 30 (this is just for testing. In the applications, it is an infinite loop), the function dispatch( ) is called. This function first checks if there are any tasks that are ready for execution. If so, the function calls the function pointer of the task in the first element of the queue. If the task executed is supposed to call another task, the other task called is enqueued in the ready_queue using the ready_enqueue( ) function. The function dequeues the finished task and calls the function delay_queue_check( ) to check if there are any tasks that need to be re-executed. Then, the timer is incremented (in the applications, there is no timer)

Test Cases:

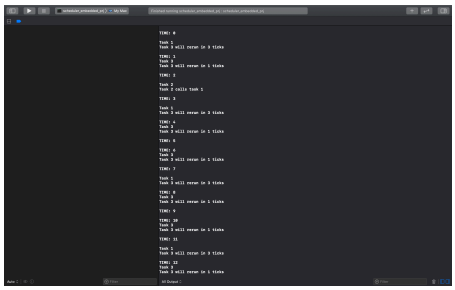
Priorities: 1 > 3 > 2

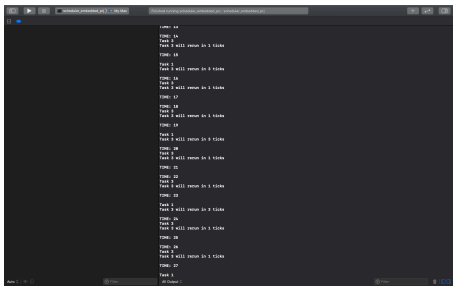
Case 1:
task 1 is a basic task
task 2 calls task 1
task 3 reruns every 3 ticks

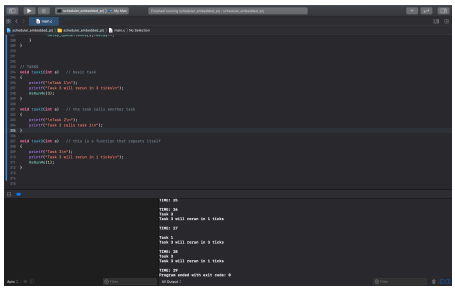




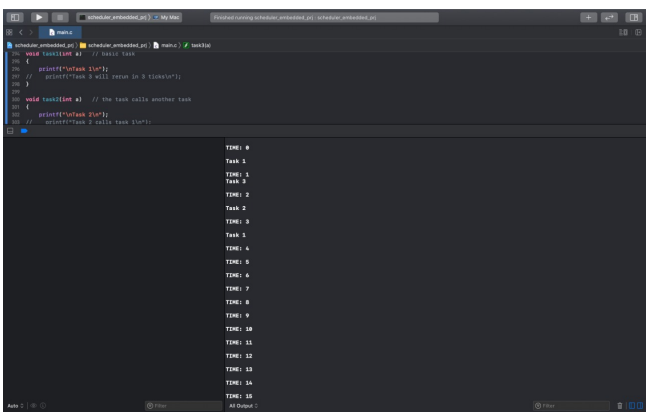
Case 2:
task 1 reruns every 3 ticks
task 2 calls task 1
task 3 reruns every 1 tick







Case 3:
task 1 is a basic task
task 2 is a basic task
task 3 is a basic task



project1EmbeddedEmanAndSalma

PROJECT 1 EXPLANATION

Abstract Scheduler Code Explanation:

---------- FIRST APPLICATION ----------

Application 1: Ambient temperature monitor

This application prompts the user to enter the threshold temperature on TeraTerm. Then, it measures and displays the temperature every 30 seconds on TeraTerm, using the DS3231. If the temperature is above the threshold temperature, the connected external LED flashes. Otherwise, it is off.

The nucleo board is first set up on STMCubeMX, where we set up the initial settings for it:
- SYS -> Serial Wire (debug mode)
- RCC -> Crystal/Ceramic Resonator
- I2C1 -> I2C: In order to order the sensor to read the temperature every 30s and to be able to read it from it to display it to the user.
- USART2 -> Asynchronus: To allow asynchronus communication in order for the user to type on TeraTerm and view the temperatures on it.
- Baud Rate -> 9600 Bits/s
- GPIO Output -> PB3: To be able to turn on/off/flash the external LED (depending on the temperature wrt the threshold)

There are structs for the tasks and queues. Every task has a function pointer, task_num (1 or 2 or 0- done task), priority and delay. The queue struct has an array of tasks, current_size (size of active tasks in the queue) and tasks_are_done_flag. The init_function function assigns the parameters of all tasks and queues. Also, it enqueues both of our tasks. There are two tasks (later on them). The dispatch function keeps going as long as there are tasks ion either the ready or delay queues (as long as the application is running, since there are rerunme task(s)). It dequeues the ready queue with the function at the head (completed) and loops over the delay queue and if there are any task whose delay has reached 0, then it is ready to be moved to the ready queue and this is done.

Task 1 sets the control buffers of the sensor in order to force it to take readings every 30s (more frequent than it would if not user-initiated). Then, this temperature is read, converted and shown to the user on TeraTerm, over USART2. If the temperature exceeds the threshold, the external LED flashes. This task is a ReRunMe(600) task, which means that it gets repeated every 600 * 50ms = 30s. This is because one SYSTICK is defined as 50ms. Task 2 toggles the LED if the flag_toggle_LED flag is true. It is a ReRunMe(1) task.

The SysTick_Handler decremnents the delay of every task in the delay queue every tick.

To use:
- Build the code
- Connect the nucleo board and the USB-TTL, which are connected together and to the rest of the circuit as shown in the pictures and video
- Enter Debug Mode
- Open TeraTerm selecting the USB serial connection (COM 5)
- Run the code
- Enter the threshold temp when prompted
- Observe the readings and the LED

Pictures:
IMG_4986.HEIC
IMG_4990.HEIC

Video:
https://youtu.be/QHPCCP-YkKk

---------- SECOND APPLICATION ----------

Application 2: Parking Sensor

This application measures the temperature between a sencor and and object. The closer the object is to the sensor, the faster the buzzers beeps. If it is a significant distance away from it, it will not beep (no risk of collision).

The nucleo board is first setup on STMCubeMX, where we set up the initial settings for it:
- SYS -> Serial Wire (debug mode)
- TIM1 -> Clock Source: Internal Clock, Channel 1: Input Capture Direct Mode, Prescaler: 8-1, Counter Period: 0xffff-1, NVIC Settings: TIM1 Capture Compare Interrupt
- Baud Rate -> 9600 Bits/s
- GPIO Output -> PB3 and PA9: To be able to turn on/off buzzer and communicate wirth sensor

The same applies as the first application and everything is the same except for the tasks and main function. Task 1 measures the distance using the sensor and determines a frequency depending on the range of distance it falls in. The greater the distance, the greater the frequency. The distance is measure using the function HCSR04_Read(), which pulls the TRIG pin high and low with a delay in between and using the timer TIM1. It is ReRunMe(10)- every 10x50ms = 0.5s. Task 2 toggles the output PB£ going to the buzzer according to the frequency which is controlled

by a simple delay function between turning it on and off. It is ReRunMe(1)- every 50ms.

The SysTick_Handler decremnents the delay of every task in the delay queue every tick.

To use:
- Build the code
- Connect the nucleo board, which is connected to the rest of the circuit as shown in the pictures and video
- Load it onto the board
- Gert any object and move it closer and farther away from the sensor and listen to the buzzer that will indicate the proximity.

Video:
https://youtu.be/9W8Lt7K0-NY

Pictures: