AGH	*Akademia Górniczo-Hutnicza w Krakowie*

Faculty: ESA	Academic Year 2019/2020	Year of study	Fields of science:

Student name:	**Ioannis Manousaridis**
Course:	**Real Time Operating Systems in Control Applications**
Exercise nr:	**10P**
Exercise subject:	**Exercise RTOS-10P. FreeRTOS –timers**
Lecturer:	dr inż. Grzegorz Wróbel

Date: 00.00.2020		Grade:

1 Exercise purpose

The purpose of the exercise is to learn about the possibilities of measuring time in the real-time operating systems.

2 Assumptions / Theory

Timers are really useful in operating systems. By using delays the whole programs stops and waits for the delay function to finish. By using timers, the tasks can be executed concurrently without any delays. This gives the opportunity to execute the tasks faster. The fiigure 1 helps for a better understanding of this concept.

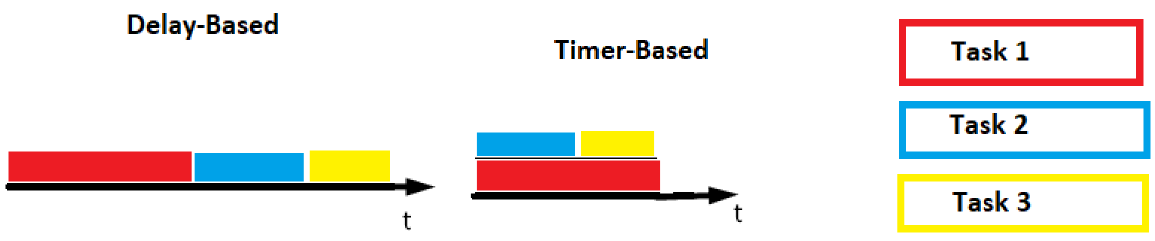


Figure 1: Difference between tasks which use delays, on the left, and timers on the right.

In FreeRTOS there is a dedicated 'Tmr Svc' (Timer Service or Deamon) Task which maintains an ordered list of software timers, with the timer to expire next in front of the list). The Timer Service task is not continuously running: from the Timer List the task knows the time when it has to wake up each time a timer in the timer list has expired. When a timer has expired, the Timer Service task calls its callback (the Timer callback).

The other concept the Timer task is using is a queue: this queue is used for inter-process communication. Using that queue, other tasks can send commands to the timer task, for example to start or stop a timer. The Timer Task will wake up when something is sent to that queue. And that way the Timer API has parameters like 'ticksToWait' to specify the waiting time if the timer queue is full.

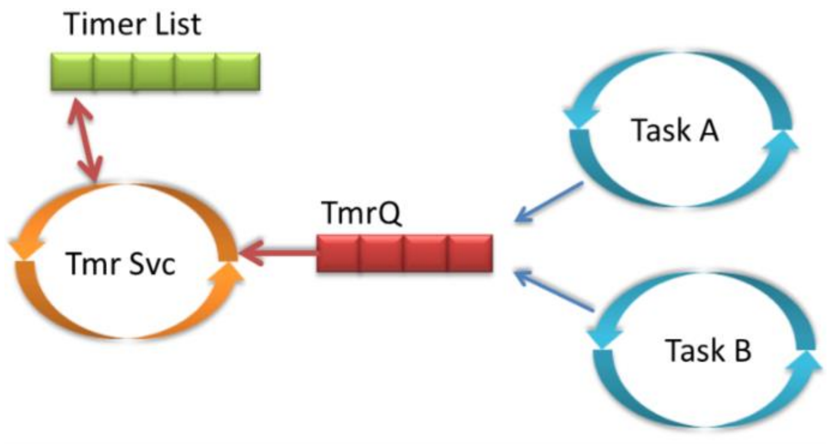


Figure 2: Diagram of how the timers and tasks work in FreeRTOS.

FreeRTOS software timers support two different software timers, configured at timer creation time:

- **One-shot timer**. Once started, a one-shot timer will execute its callback function only once. It can be manually re-started, but will not automatically re-start itself.

- **Auto-reload timer.** Conversely, once started, an auto-reload timer will automatically re-start itself after each execution of its callback function, resulting in periodic callback execution.

For the creation a timer the following function is being used:

```
TimerHandle_t xTimerCreate
                 ( const char * const pcTimerName,
                   const TickType_t xTimerPeriod,
                   const UBaseType_t uxAutoReload,
                   void * const pvTimerID,
                   TimerCallbackFunction_t pxCallbackFunction );
```

The first parameter is just the name of the timer and the second is the period of the timer. The third parameter is the uxAutoReload. If it is set to pdTRUE, then the timer will expire repeatedly with a frequency set by the xTimerPeriod parameter. If uxAutoReload is set to pdFALSE, then the timer will be a one-shot and enter the dormant state after it expires. The next parameter is an identifier that is assigned to the timer being created. Typically this would be used in the timer callback function to identify which timer expired when the same callback function is assigned to more than one timer which did not happen in this lab and thus it is not used. The last parameter is the function which is going to be called when the timer expires.

3 Description of implementation

Two files were created in the scope of this lab. In the first file, lab13_013.c, the user has three options in the begining. The first one is to create two one-shot timers with the same trigger time, the second is to create two-one shot timers but with different trigger time and the last one is to create one one-shot timer and one auto-reload timer. Regardless of the input of the user two timers will be created. The diffence is on the parameters. Each timer will have different name, period, uxAutoReload and callback function.

Also, it is important after the creation of the timer to start the timer. The creation of the timer will not make it to start ticking. For this reason, the following line of code is used:

```
if (xTimerStart(timerX_one, 10) != pdPASS) {
        printf("Timer start error"); }
```

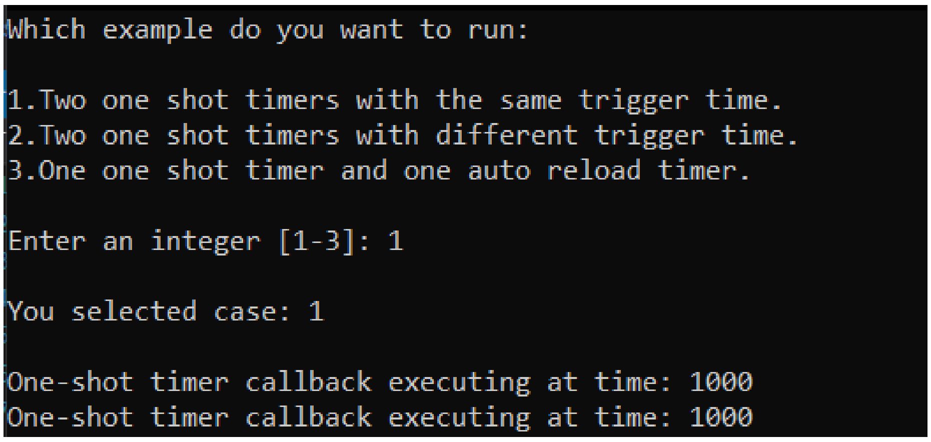
This will start the timer and if an error occures, it will print an error on the screen to notify the user that something is wrong with the X timer.

When the timers have been created and will start counting. When the reach their period, they will expire and they will call their callback function. Each of those function just get the current ticking time and they printing it on the screen. Also, they print if the callback function was a result of an one-shot timer or an auto-reload timer.

In the second file, lab13_014.c, the user does not provide any input. There are three timers, all of them are auto-reload timers with different periods. The first timer has the biggest period and the second has much smaller period. The third timer has a period which is between the other two periods. The first two timers are created in the main function and they also start there.

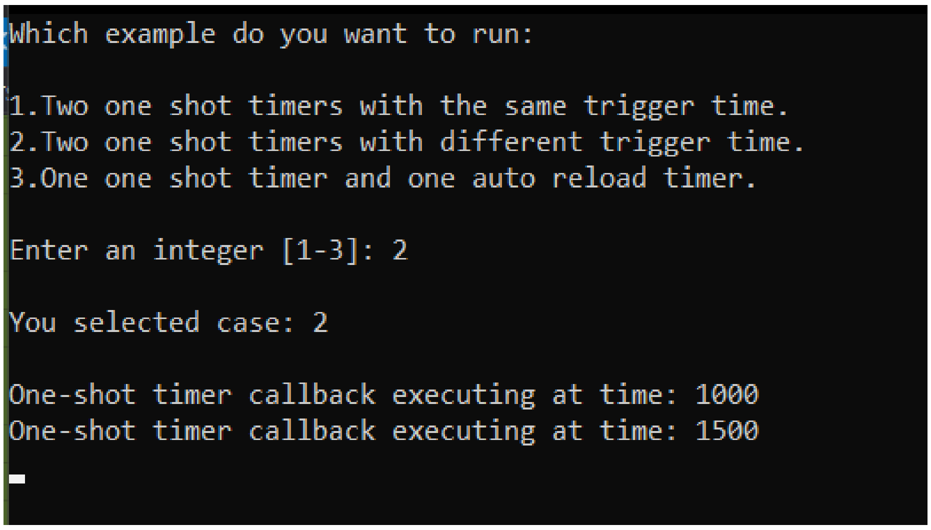
The third timers is being created when the callback of timer2 is executed for the first time. This is because the timer2 expires faster than the timer1.

The callback functions of timers 1 and 2 just print how many ticks passed from the beginng of the program till the moment they are called. The callback of timer3 has a different purpose. There are two global variables in which two ”timestamps” are stored. Specically, they just store how many ticks had been passed when the callbacks of timer1 and timer2 are executed. The callback of timer3 will calculate the absolute difference of those two ”timestamps” and it will print it on the screen.



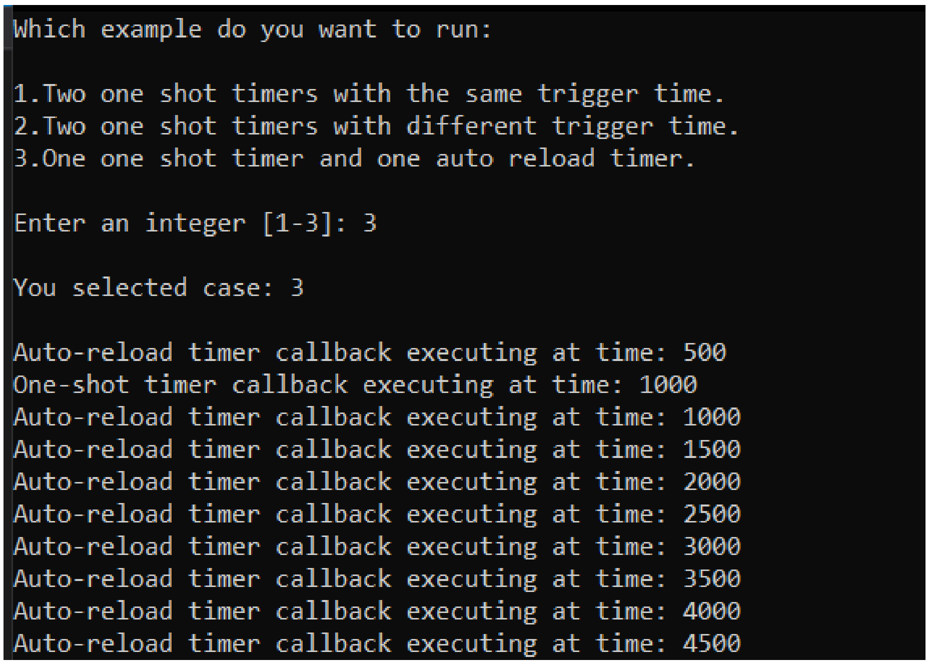
```
Which example do you want to run:

1.Two one shot timers with the same trigger time.
2.Two one shot timers with different trigger time.
3.One one shot timer and one auto reload timer.

Enter an integer [1-3]: 1

You selected case: 1

One-shot timer callback executing at time: 1000
One-shot timer callback executing at time: 1000
```

Figure 3: Two one-shot timers were activated with the same trigger times.



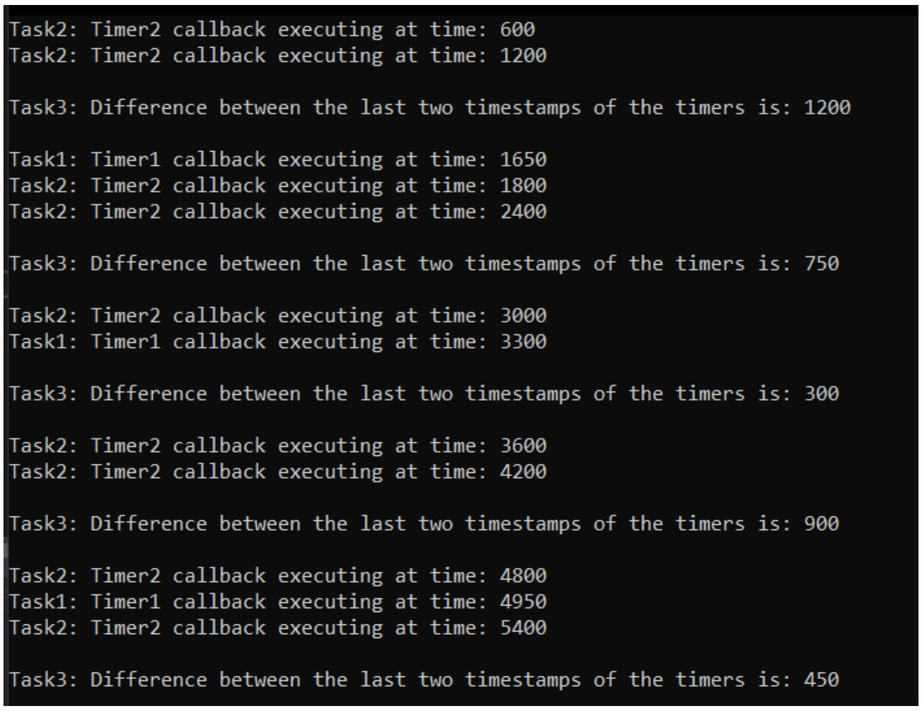
```
Which example do you want to run:

1.Two one shot timers with the same trigger time.
2.Two one shot timers with different trigger time.
3.One one shot timer and one auto reload timer.

Enter an integer [1-3]: 2

You selected case: 2

One-shot timer callback executing at time: 1000
One-shot timer callback executing at time: 1500
```

Figure 4: Two one-shot timers were activated with different trigger times.



```
Which example do you want to run:

1.Two one shot timers with the same trigger time.
2.Two one shot timers with different trigger time.
3.One one shot timer and one auto reload timer.

Enter an integer [1-3]: 3

You selected case: 3

Auto-reload timer callback executing at time: 500
One-shot timer callback executing at time: 1000
Auto-reload timer callback executing at time: 1000
Auto-reload timer callback executing at time: 1500
Auto-reload timer callback executing at time: 2000
Auto-reload timer callback executing at time: 2500
Auto-reload timer callback executing at time: 3000
Auto-reload timer callback executing at time: 3500
Auto-reload timer callback executing at time: 4000
Auto-reload timer callback executing at time: 4500
```

Figure 5: One one-shot timer and one auto-reload timer were activated.



```
Task2: Timer2 callback executing at time: 600
Task2: Timer2 callback executing at time: 1200

Task3: Difference between the last two timestamps of the timers is: 1200

Task1: Timer1 callback executing at time: 1650
Task2: Timer2 callback executing at time: 1800
Task2: Timer2 callback executing at time: 2400

Task3: Difference between the last two timestamps of the timers is: 750

Task2: Timer2 callback executing at time: 3000
Task1: Timer1 callback executing at time: 3300

Task3: Difference between the last two timestamps of the timers is: 300

Task2: Timer2 callback executing at time: 3600
Task2: Timer2 callback executing at time: 4200

Task3: Difference between the last two timestamps of the timers is: 900

Task2: Timer2 callback executing at time: 4800
Task1: Timer1 callback executing at time: 4950
Task2: Timer2 callback executing at time: 5400

Task3: Difference between the last two timestamps of the timers is: 450
```

Figure 6: Three auto-reload timers were activated. Two of them are just printing the ticks that have passed every time they call their callback function and the third one print the difference in the execution between the two callbacks.

4 Conclusions

In general timers are really useful in real time operating systems and they help in concurrent events and when precision is needed. A problem which has been faced in the lab13_014.c file was that the third autoreload timer could not be started in the main function and thus it was started by a callback function. This most probably has to do with the FreeRTOS system. Also, the microproccesors and general the real time operating systems have a finite number of timers so this add some restriction. Plus, some timers have specialized functions. For a example a PWM timers will be mostly used to generated rectangular pulses.

Furthermore, in this lab, not all the functionalities of the timers were used. The pvTimerID identifier is a really interesting function that the FreeRTOS OS provides but it was not used. The manual reset of an one-shot timer was not also used.

Finally, it is important to mention that the periods that were assigned to the timers and the values of ticks which were print on screen had 100% match which is really important. However, this was tested only in a simulation in a windows system and not in real time operating system.