FREERTOS V9
CHAPTER 5
SOFTWARE TIMER MANAGEMENT

Kate Huang

Outline

- I. Introduction
- II. Software Timer Callback Functions
- III. Attributes and States of a Software Timer
- IV. The Context of a Software Timer
- V. Creating and Starting a Software Timer
- VI. The Timer ID
- VII. Changing the Period of a Timer
- VIII. Resetting a Software Timer

I. Introduction & Scope

◎ Introduction

- Software timers are implemented by, and are under the control of, the FreeRTOS kernel.
- They do not require hardware support, and are not related to hardware timers or hardware counters.

I. Introduction & Scope

◎ Scope :

- The characteristics of a software timer compared to the characteristics of a task.
- The RTOS daemon task.
- The timer command queue.
- The difference between a one shot software timer and a periodic software timer.
- How to create, start, reset and change the period of a software timer.h.

II.Settings

- Software timer functionality is optional. To include software timer functionality:

  1. Build the FreeRTOS source file FreeRTOS/src/timers.c as part of your project.

  2. Set configUSE_TIMERS to 1 in FreeRTOSConfig.h

```
/* Software timer related definitions. */
#define configUSE_TIMERS                    1
#define configTIMER_TASK_PRIORITY           3
#define configTIMER_QUEUE_LENGTH            10
#define configTIMER_TASK_STACK_DEPTH        configMINIMAL_STACK_SIZE
```

https://github.com/feilipu/Arduino_FreeRTOS_Library

https://sourceforge.net/projects/avrfreertos/?source=typ_redirect

II. Function Prototype

- Void ATimerCallback(TimerHandle_t xTomer);

```
void ATimerCallback( TimerHandle_t xTimer );
```

III.TYPE

- One-shot timers
- Auto-reload timers

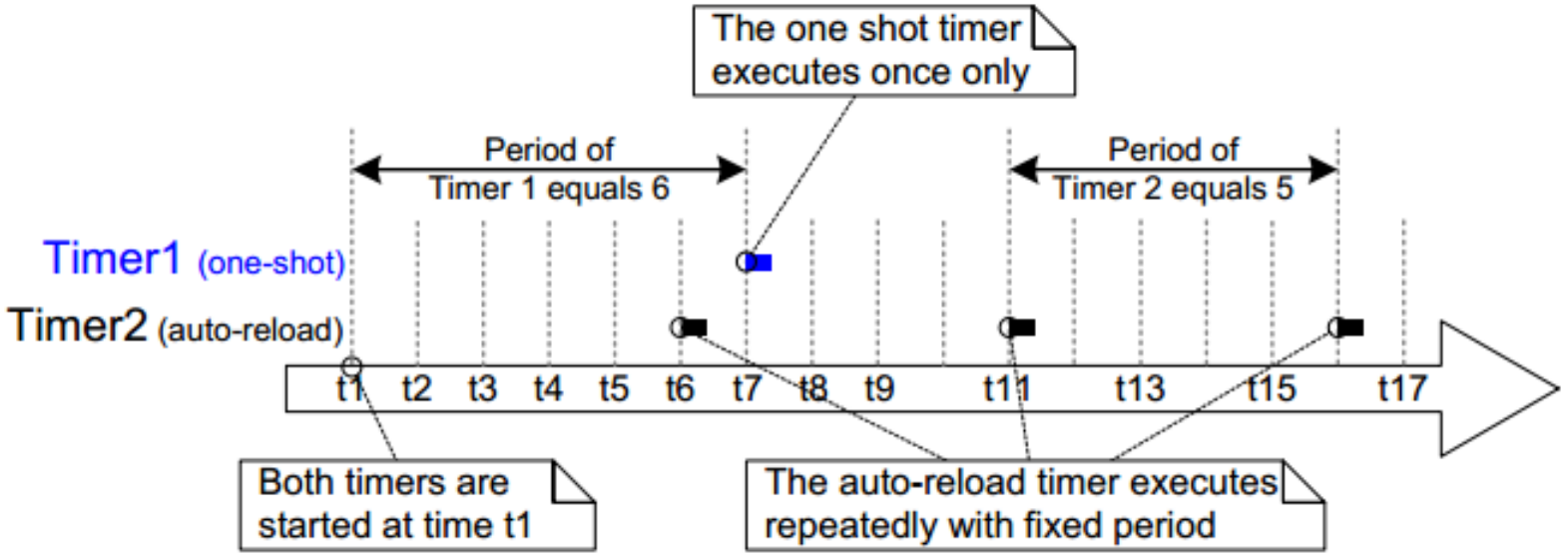

Figure 38 The difference in behavior between one-shot and auto-reload software timers

III. Status and Transitions of Timers

- Auto-reload timer

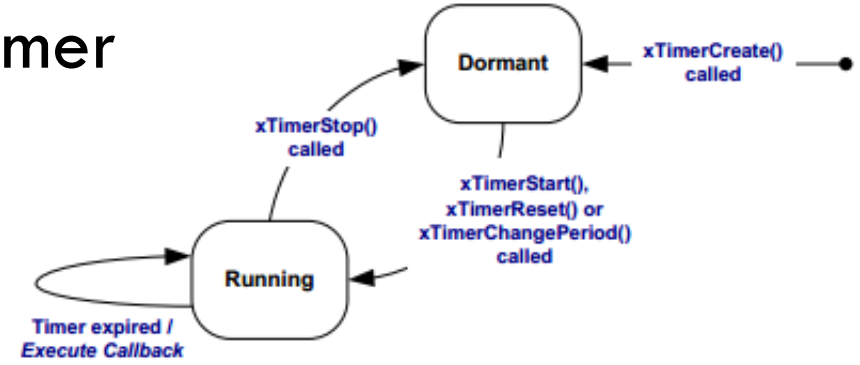

Figure 39 Auto-reload software timer states and transitions

- One-shot timer

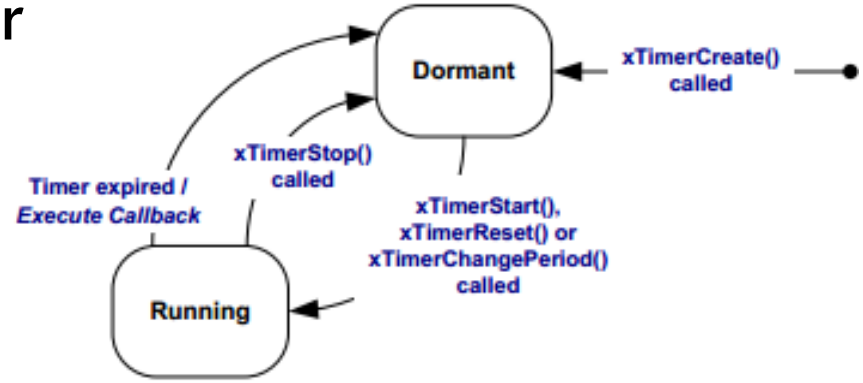

Figure 40 One-shot software timer states and transitions

- The xTimerDelete() : A timer can be deleted at any time.

IV. The RTOS Daemon (Timer Service) Task

- It is a standard FreeRTOS task that is created automatically when the scheduler is started.
- Its priority and stack size are set by the configTIMER_TASK_PRIORITY and configTIMER_TASK_STACK_DEPTH compile time configuration constants respectively.
- Both constants are defined within FreeRTOSConfig.h

IV. The RTOS Daemon (Timer Service) Task

- The daemon task is scheduled like any other FreeRTOS task; it will only process commands, or execute timer callback functions, when it is the highest priority task that is able to run.

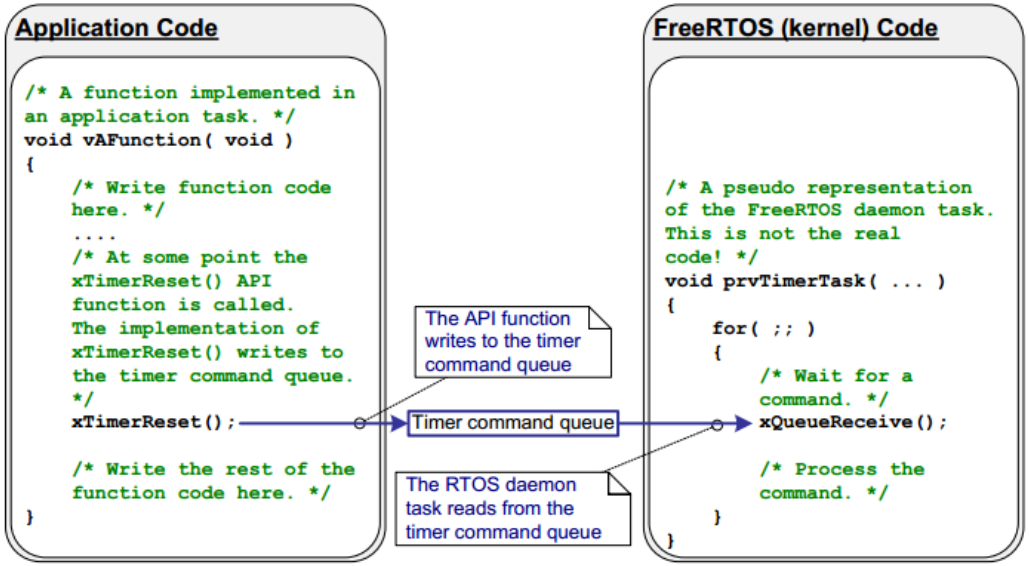

Figure 41 The timer command queue being used by a software timer API function to communicate with the RTOS daemon task

IV. The RTOS Daemon (Timer Service) Task

- The priority of Task 1 is higher than the priority of the daemon task, and the priority of the daemon task is higher than the priority of the Idle task.

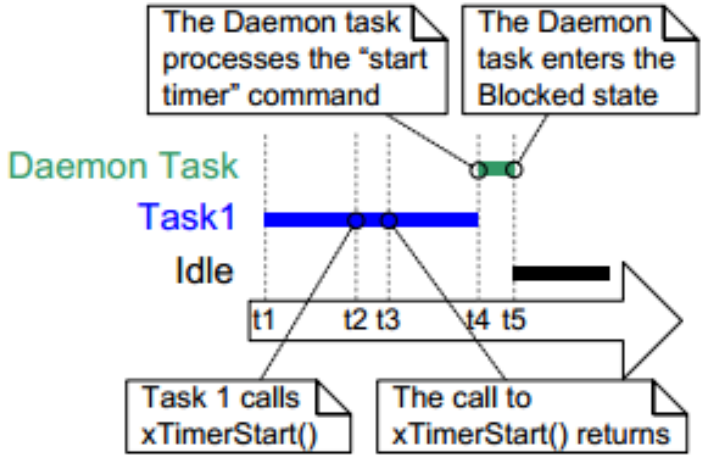

Figure 42 The execution pattern when the priority of a task calling xTimerStart() is above the priority of the daemon task

IV. The RTOS Daemon (Timer Service) Task

- The priority of the daemon task is higher than the priority of Task 1, and the priority of the Task 1 is higher than the priority of the Idle task.

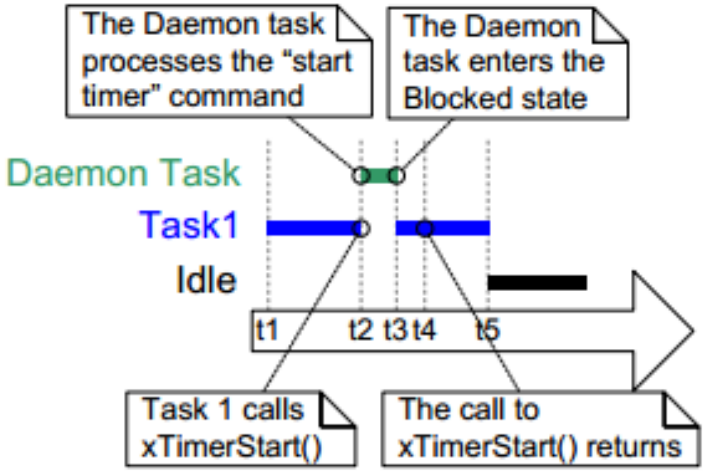

Figure 43 The execution pattern when the priority of a task calling xTimerStart() is below the priority of the daemon task

V. Creating & Starting

- **The xTimerCreate() API Function**

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

- **The xTimerStart() API Function**

```
TimerHandle_t xTimerStart( TimerHandle_t xTimer, TickType_t xTicksToWait );
```

V. Creating & Starting

- **Sample 1**

  Creating and Starting functions

- **Sample 2**

  Creating one-shot and auto-reload timers

VI. The Timer ID

- **The vTimerSetTimerID() API Function**

```
void vTimerSetTimerID( const TimerHandle_t xTimer, void *pvNewID );
```

- **The pvTimerGetTimerID() API Function**

```
void *pvTimerGetTimerID( TimerHandle_t xTimer );
```

VI. The Timer ID

- **Sample3:**
  Using the callback function parameter and the software timer ID

VII. Changing the Period

- **The xTimerChangePeriod() API Function**

```
BaseType_t xTimerChangePeriod( TimerHandle_t xTimer,
                               TickType_t xNewTimerPeriodInTicks,
                               TickType_t xTicksToWait );
```

- **Sample 4:**

  Changing the Period(LED Toggles)

VIII. Resetting

- Resetting a software timer means to re-start the timer; the timer's expiry time is recalculated to be relative to when the timer was reset, rather than when the timer was originally started. This is demonstrated by Figure 46, which shows a timer that has a period of 6 being started, then reset twice, before eventually expiring and executing its callback function.

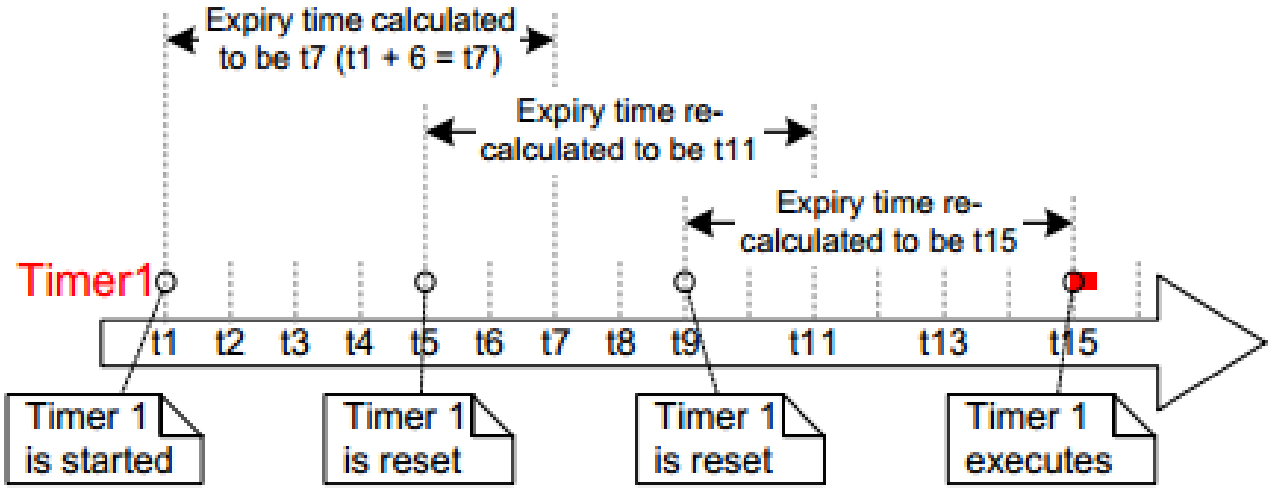

Figure 46 Starting and resetting a software timer that has a period of 6 ticks

VIII. Resetting

- **The xTimerReset() API Function**

```
BaseType_t xTimerReset( TimerHandle_t xTimer, TickType_t xTicksToWait );
```

- **Sample 5:**

  Resetting a software timer

THE END

- Thanks for your attendance.

APPENDIX

Kate Huang

Hardware Timer v.s. Software Timer

- HW:
- hardware timer □

1.integrated into microcontroller or separate chip □

2.can be exact □

3.allows actions in parallel□

Hardware Timer v.s. Software Timer

- SW: 效率跟準確性都取決於程式的寫法
- software delay loop □

1.easy to insert; requires no additional hardware □

2.Causing problem if other actions are to be done in parallel □

3.Time can be accurate if code is carefully timed □

4.IDE may have timing option

Reference

- HW : Arduino Mega 2560
- SW : Eclipse Cpp Neon IDE
- Document : Mastering the FreeRTOS Real Time Kernel - a Hands On Tutorial Guide