Διαδίκτυο των Πραγμάτων

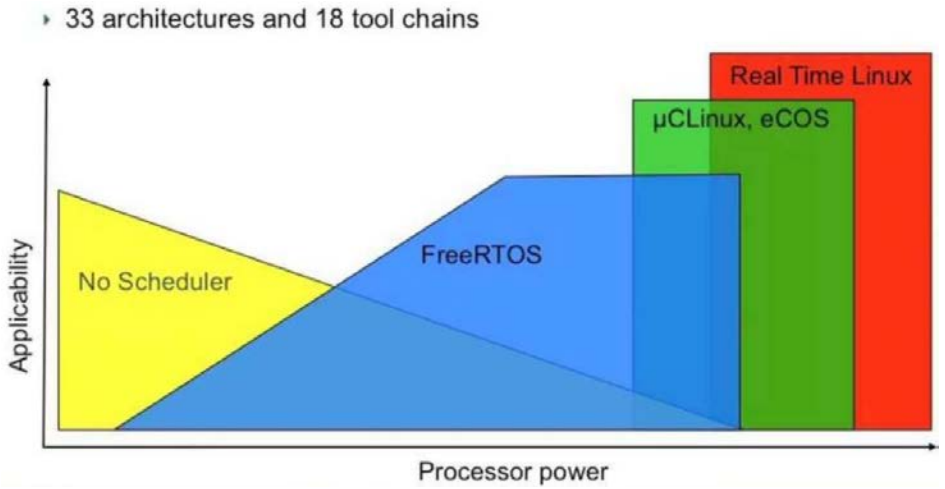
Μελέτη Περίπτωσης FreeRTOS

Περιεχόμενα

ΓΕΝΙΚΑ ΣΤΟΙΧΕΙΑ

Γενικά στοιχεία

- Δημοφιλές RTOS ανοικτού κώδικα κατάλληλο για συσκευές IoT
- Δημιουργός ο Richard Barry το 2002
- Περιλαμβάνει έναν πυρήνα και ένα αυξανόμενο σύνολο βιβλιοθηκών λογισμικού
- Έμφαση στην αξιοπιστία και την ευκολία χρήσης



Hardware Partners

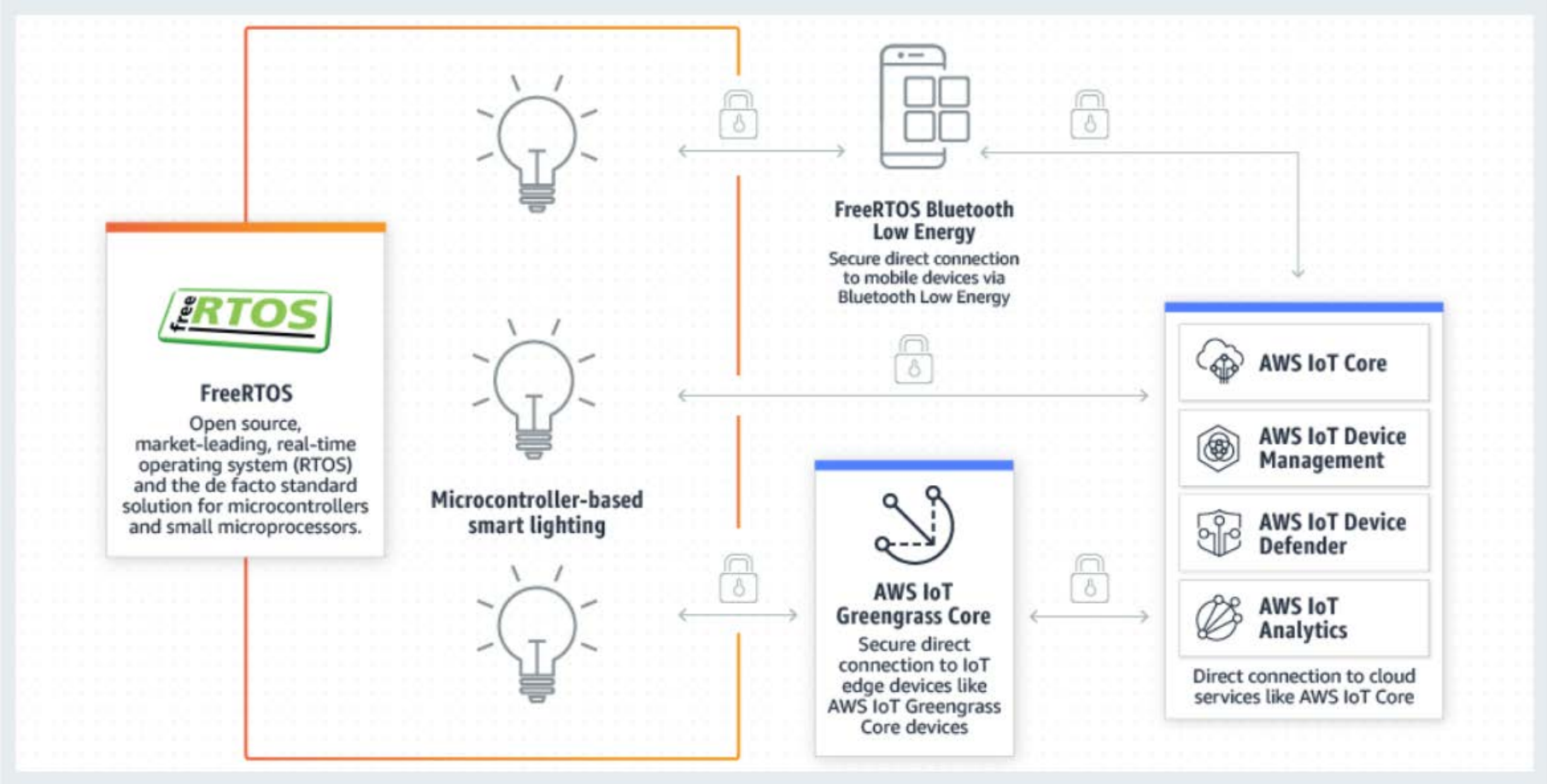
CYPRESS EMBEDDED IN TOMORROW
ESPRESSIF
Infineon
MARVELL
MEDIATEK
MICROCHIP
NORDIC SEMICONDUCTOR
nuvoTon
NXP
RENESAS
life.augmented
Texas Instruments
XILINX

Developer	Real Time Engineers Ltd.
OS family	Real-time operating systems
Working state	Current
Source model	Open source
Latest release	10.3.1[1] / 2020-02-19
Repository	github.com/FreeRTOS/FreeRTOS
Marketing target	Embedded devices
Platforms	ARM (ARM7, ARM9, Cortex-M3, Cortex-M4, Cortex-M7, Cortex-A), Atmel AVR, AVR32, HCS12, MicroBlaze, Cortus (APS1, APS3, APS3R, APS5, FPF3, FPS6, FPS8), MSP430, PIC, Renesas H8/S, SuperH, RX, x86, 8052, Coldfire, V850, 78K0R, Fujitsu MB91460 series, Fujitsu MB96340 series, Nios II, Cortex-R4, TMS570, RM4x, Espressif ESP32, RISC-V
Kernel type	Microkernel
License	MIT[2]
Official website	www.freertos.org

Διανομές FreeRTOS

- Ο πυρήνας του FreeRTOS είναι γραμμένος στη γλώσσα C
  - 6 αρχεία C: tasks.c, list.c, queue.c, timers.c, event_groups.c, croutine.c
  - 1 architecture/compiler specific αρχείο (π.χ. για δυναμική κατανομή μνήμης)
- Μικρές απαιτήσεις μνήμης
  - Π.χ. το port RL78 (16-bit CPU core by Renesas Electronics) με 13 tasks, 2 queues και 4 software timers → memory footprint < 4Kbytes RAM
- Εκτενές API documentation
- Διανομές με επιπλέον λειτουργικότητα
  - SAFERTOS με TÜV πιστοποίηση
  - FreeRTOS+TCP (πλήρη υποστήριξη TCP/IP stack)
  - FreeRTOS+FAT (υποστήριξη συστήματος αρχείων)
  - Amazon FreeRTOS (συνεργασία με AWS cloud platform)

Σενάριο χρήσης Amazon FreeRTOS



FreeRTOS tasks

FreeRTOS tasks

- Τα tasks του FreeRTOS είναι αντίστοιχα με τα processes του Linux – οντότητες με τον δικό τους κώδικα και μνήμη
- Δεν υποστηρίζονται μηχανισμοί εικονικής μνήμης
  - δεν υπάρχει αυστηρή προστασία και διαχωρισμός των περιοχών μνήμης
- Εκχωρείται μέγιστος χώρος στη στοίβα κατά τη δημιουργία ενός task που μπορεί να χρησιμοποιηθεί από αυτό
- Εκτίμηση χώρου από τον προγραμματιστή
  - stack overflow vs. σπατάλη μνήμης
- Ανίχνευση stack overflow
  - `configCHECK_FOR_STACK_OVERFLOW1` (FreeRTOSConfig.h)
  - `void vApplicationStackOverflowHook(TaskHandle_t xTask, signed char *pcTaskName)`

FreeRTOS tasks

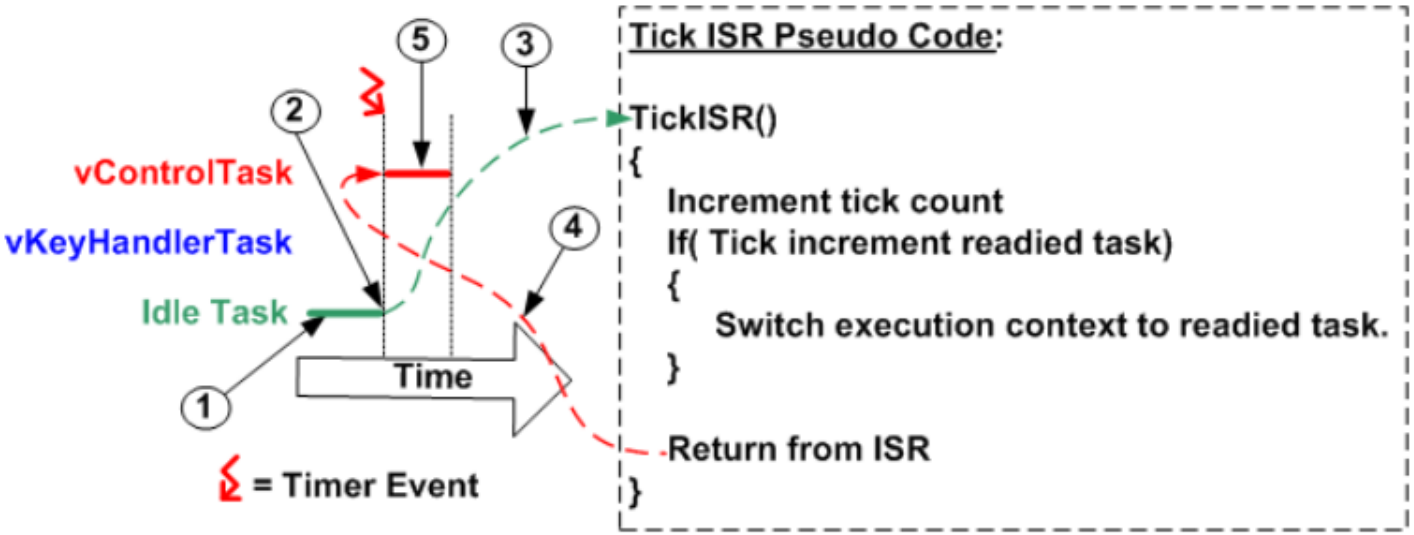
- Η API function για τη δημιουργία ενός task:

```
BaseType_t xTaskCreate( TaskFunction_t pvTaskCode,
                        const char * const pcName,
                        uint16_t usStackDepth,
                        void *pvParameters,
                        UBaseType_t uxPriority,
                        TaskHandle_t *pxCreatedTask );
```

- `pvTaskCode` → C συνάρτηση με τον κώδικα που θα τρέχει το task
- `pcName` → όνομα για το task
- `usStackDepth` → το μέγεθος της μνήμης που θα εκχωρηθεί ως στοίβα
- `pvParameters` → παράμετροι στη συνάρτηση του task
- `uxPriority` → η προτεραιότητα που δίνεται αρχικά στο task
  - Η προτεραιότητα είναι μια τιμή που δίνεται στην δημιουργία του task αλλά μπορεί να αλλάξει σε οποιαδήποτε στιγμή της ζωής του.
- `pxCreatedTask` → handler στο task που θα δημιουργηθεί (ένα άλλο task που έχει τον handler να μπορεί π.χ. να αλλάξει την προτεραιότητα του ή να το διαγράψει)

FreeRTOS tick

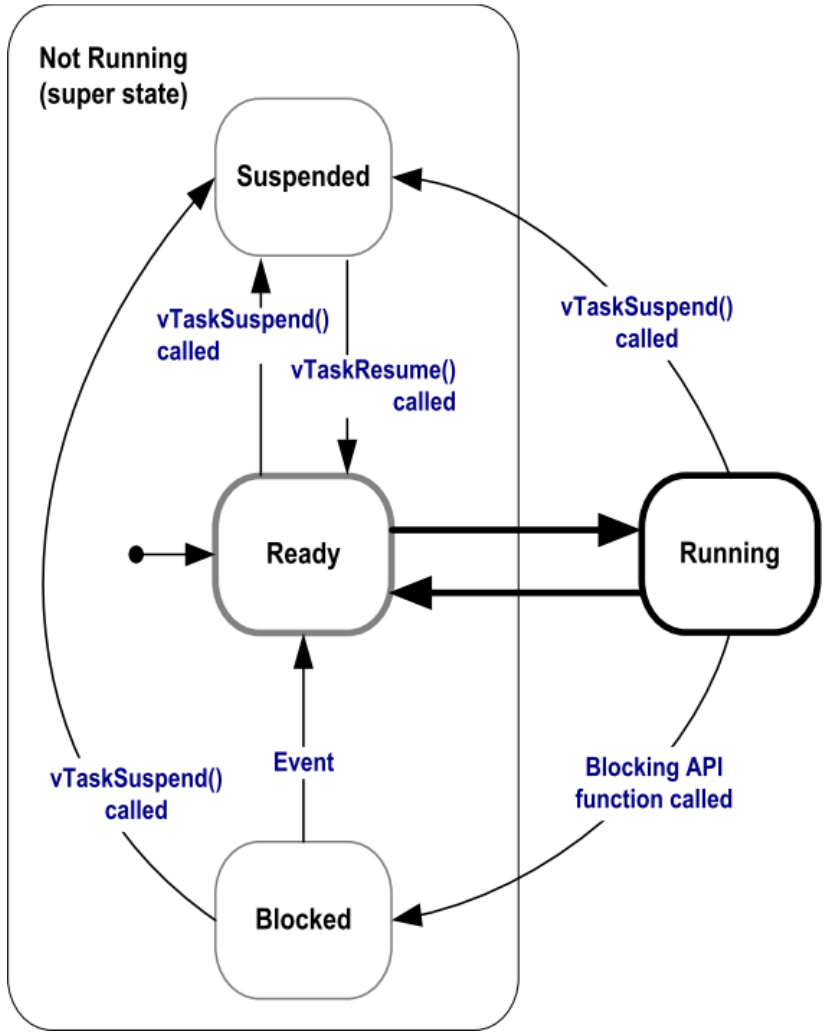
- Τα FreeRTOS “ticks” συμβαίνουν με περίοδο που τίθεται στο αρχείο `FreeRTOSConfig.h`
  - ανάλογα με τον επεξεργαστή στον οποίο έχει γίνει port το FreeRTOS επιλέγεται και η πηγή αυτών των interrupt
    - Π.χ. Private Timer του ARM core στο Zynq-700 SoC της Xilinx (xscutimer.h)
  - Μια τυπική τιμή είναι 100 Hz → περίοδος 10ms
- Υποστηρίζεται και tickless mode για ελαχιστοποίηση της κατανάλωσης ισχύος
- Σε κάθε tick interrupt τρέχει μια Interrupt Service Routine (ISR):



Χρόνος στο FreeRTOS

- Η περίοδος των tick αποτελεί την μονάδα χρόνου για το FreeRTOS
- Για παράδειγμα αν θέλουμε ένα task να κάνει delay 5ms:
  - `vTaskDelay(pdMS_TO_TICKS(5))`
  - γίνεται μετατροπή σε “ticks” τα οποία κι αναμένει η **vTaskDelay**
  - η κλήση θέτει το task σε blocked state

Καταστάσεις FreeRTOS tasks



FreeRTOS scheduling

FreeRTOS scheduling

- Ο scheduler αποφασίζει ποιο task σε Ready state θα μεταβεί στην Running state
- Παράμετροι διαμόρφωσης στο `FreeRTOSConfig.h`
  - configUSE_PREEMPTION
  - configUSE_TIME_SLICING
  - configUSE_TICKLESS_IDLE (default **τιμή 0)**
- Το FREE_RTOS δημιουργεί δύο ειδικά tasks: **idle task** και **daemon task**
- Το idle task έχει την ελάχιστη προτεραιότητα (PRIORITY 0) και συμβάλλει στην απελευθέρωση μνήμης διαγραφέντων tasks
- Το daemon task παρέχει υποστήριξη στην υλοποίηση των software timers

Πολιτική scheduling	configUSE_PREEMPTION	configUSE_TIME_SLICING
Fixed Priority Pre-emptive Scheduling with Time Slicing	1	1
Prioritized Pre-emptive Scheduling without Time Slicing	1	0
Co-operative Scheduling	0	Any value

Ανταλλαγή δεδομένων

Ανταλλαγή δεδομένων

- Η ανταλλαγή δεδομένων μεταξύ tasks χρησιμοποιεί τα queues (FIFO δομή)
- Τα tasks εγγραφείς γράφουν στο τέλος της και τα tasks αναγνώστες διαβάζουν από την αρχή.
- Δημιουργία queue

```
QueueHandle_t xQueueCreate( UBaseType_t uxQueueLength, UBaseType_t uxItemSize );
```

- Κάθε task το οποίο διαθέτει το handler στο queue μπορεί να καλέσει τις:

```
BaseType_t xQueueSendToFront( QueueHandle_t xQueue,
                              const void * pvItemToQueue,
                              TickType_t xTicksToWait );

BaseType_t xQueueSendToBack( QueueHandle_t xQueue,
                             const void * pvItemToQueue,
                             TickType_t xTicksToWait );

BaseType_t xQueueReceive( QueueHandle_t xQueue,
                          void * const pvBuffer,
                          TickType_t xTicksToWait );
```

Software timers

Software timers

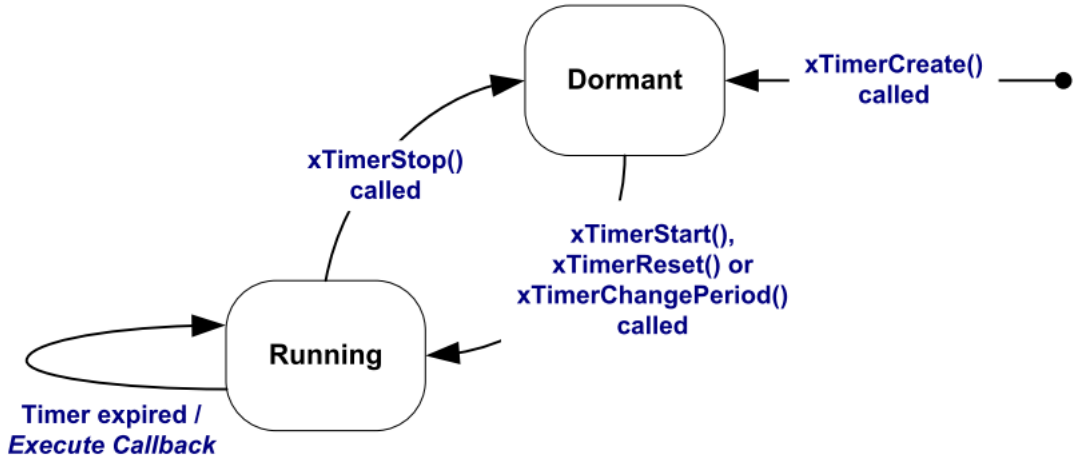
- Οντότητες του FreeRTOS που εξασφαλίζουν την εκτέλεση μιας C συνάρτησης (callback) σε αυστηρά καθορισμένο χρόνο
  - one-shot timer (μια φορά) π.χ. `BacklightOff()` σε 10 ticks
  - auto-reload timer (περιοδικά) π.χ. `LedToggle()` κάθε 5 ticks
- Υλοποίηση από τον πυρήνα χωρίς εμπλοκή timers/counters υλικού
- Προαιρετική λειτουργικότητα του πυρήνα (timers.c)
- Software timer callback function prototype:

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

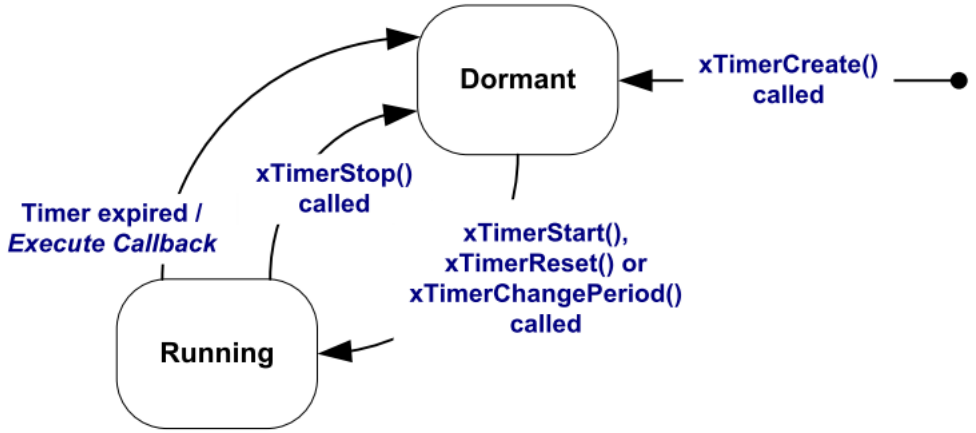
- Τα callbacks πρέπει να είναι σύντομα και να μην οδηγούν σε Blocked κατάσταση (π.χ. όχι κλήση της `vTaskDelay()`)

Καταστάσεις software timer

- Dormant
  - ο software timer υπάρχει στο σύστημα, αλλά δεν εκτελείται, και κατά συνέπεια οι callback functions που σχετίζονται δεν θα εκτελεστούν
- Running
  - ο software timer θα εκτελέσει την callback function σύμφωνα με τον προσδιορισμένο χρόνο του

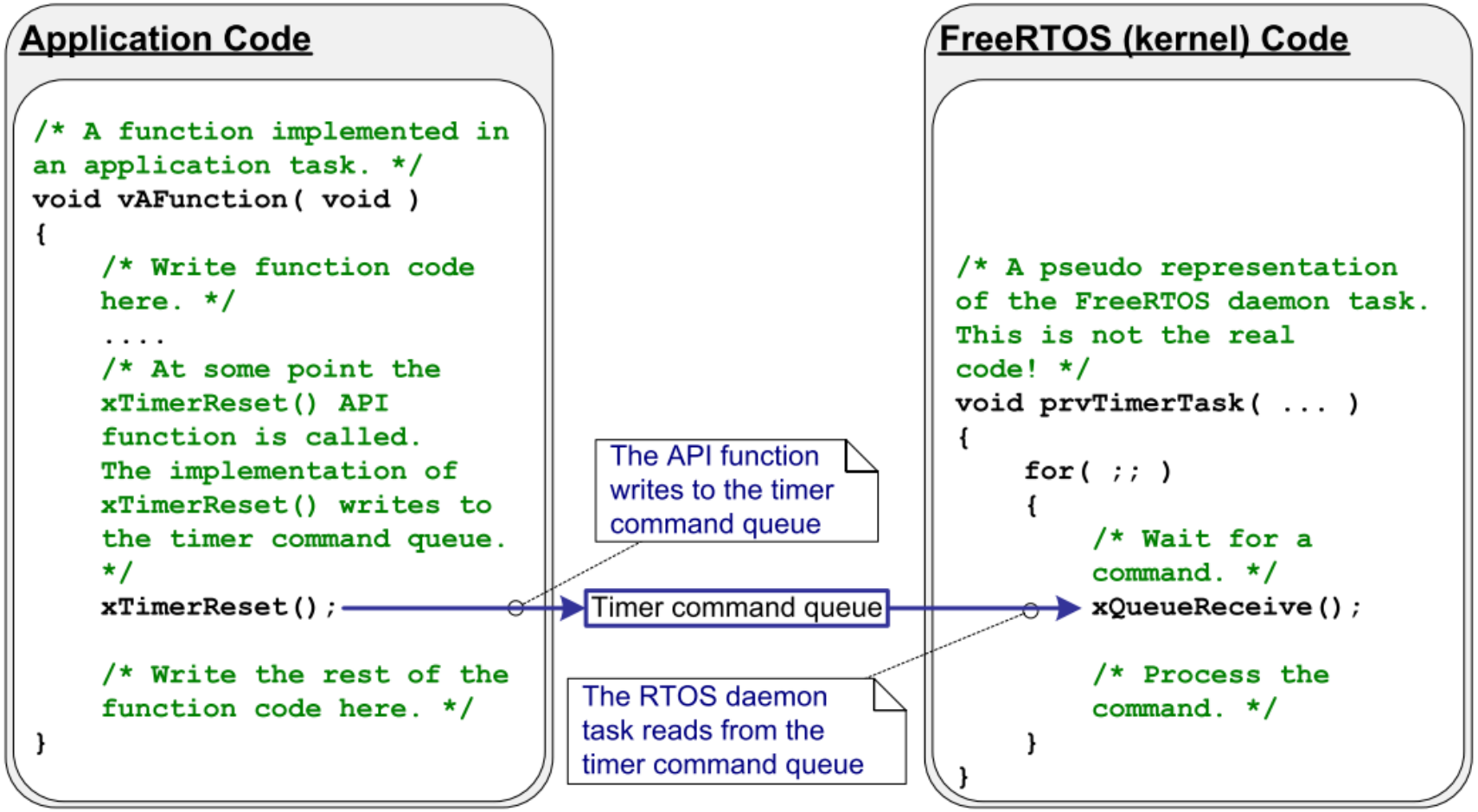


Auto-reload software timer states and transitions



One-shot software timer states and transitions

Timer command queue



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

Daemon task scheduling

- Το daemon task δρομολογείται όπως και κάθε άλλο FreeRTOS task
- Ο ρόλος του είναι να επεξεργαστεί commands (τα διαβάζει από το command queue)
- Ουρά άδεια → daemon task Blocked
- Χρειάζεται context switch μεταξύ application task και daemon task → το daemon task να έχει αρκετά μεγάλη προτεραιότητα
- Ο χρόνος που θέτουμε σε ένα software timer call δεν επηρεάζεται από το πότε θα τρέξει το daemon task
  - κάθε command στο command queue έχει ένα timestamp (σε πλήθος ticks)
  - το timestamp γεφυρώνει τον χρόνο που περνά μεταξύ της δημιουργίας του command (application task) και της επεξεργασίας του (daemon task)
  - Τι θα γίνει αν ο daemon δεν τρέξει γρηγορότερα από το ζητούμενο;

Interrupts

Διαχείριση διακοπών

- Η σημασία των events στη λειτουργία των ενσωματωμένων συστημάτων πραγματικού χρόνου
- Επιλογή καλύτερης στρατηγικής για την επεξεργασία των συμβάντων
  - Πώς πρέπει να εντοπιστεί το συμβάν; Interrupts ή Polling;
  - Πόση επεξεργασία πρέπει να γίνει εντός της ρουτίνας εξυπηρέτησης διακοπών (ISR), και πόση εκτός;
  - Πώς τα συμβάντα κοινοποιούνται στον κύριο κώδικα (εκτός ISR) και πώς μπορεί αυτός ο κώδικας να δομηθεί;
- Το FreeRTOS παρέχει δυνατότητες ώστε η επιλεγμένη στρατηγική να υλοποιηθεί με απλό και επεκτάσιμο τρόπο
- Προτεραιότητες task vs. προτεραιότητες interrupt
  - Ένα task είναι μια οντότητα λογισμικού που δεν σχετίζεται με το υλικό στο οποίο εκτελείται το FreeRTOS
  - Μια ρουτίνα εξυπηρέτησης διακοπών, αν και λογισμικό, στην πράξη έχει ένα υπόστρωμα το οποίο εκφράζει μια δυνατότητα υλικού
  - Τα tasks μπορούν να εκτελούνται μόνο όταν δεν εκτελούνται ρουτίνες εξυπηρέτησης διακοπών
  - Εξάρτηση διακοπών από τις αρχιτεκτονικές MCU

Interrupt safe API

- Πολλά system calls εκτελούν ενέργειες που δεν είναι έγκυρες μέσα σε μια ρουτίνα εξυπηρέτησης διακοπών (ISR)
  - Π.χ. μετάβαση του task που έκανε χρήση του system call σε κατάσταση Blocked
- Το FreeRTOS παρέχει δύο εκδόσεις ορισμένων system calls:
  - API για χρήση από tasks
  - Interrupt safe API (κατάληξη ονόματος system call “FromISR”)
- NB: Δεν πρέπει να καλούμε εντός μιας ISR system calls που δεν έχουν κατάληξη στο όνομά τους το "FromISR“
- Σενάριο εφαρμογής: *Όταν πατηθεί ένα πλήκτρο, ανάβει ο οπίσθιος φωτισμός LCD. Εάν περάσουν 5 δευτερόλεπτα χωρίς να πατηθεί ένα πλήκτρο, τότε ο οπίσθιος φωτισμός LCD σβήνει*.
  - **xBacklightTimer**, one-shot software timer για χρόνο 5 secs
  - **vBacklightTimerCallback()** , callback function του software timer
  - **xTimerResetFromISR(...)**, interrupt safe API για χρήση εντός ISR

Υλοποίηση σεναρίου εφαρμογής


```
/* The callback function assigned to the one-shot timer.  In this
case the parameter is not used. */
void vBacklightTimerCallback( TimerHandle_t pxTimer )
{
    /* The timer expired, therefore 5 seconds must have passed
since a key    was pressed.  Switch off the LCD back-light. */
    vSetBacklightState( BACKLIGHT_OFF );
}
```

```
/* The key press interrupt service routine. */
void vKeyPressEventInterruptHandler( void )
{
     BaseType_t xHigherPriorityTaskWoken = pdFALSE;

    /* Ensure the LCD back-light is on, then reset the timer that
 is responsible for turning the back-light off after 5 seconds of
key inactivity. This is an interrupt service routine so can only
call FreeRTOS API functions that end in "FromISR". */
    vSetBacklightState( BACKLIGHT_ON );

    /* xTimerStartFromISR() or xTimerResetFromISR() could be
called here as both cause the timer to re-calculate its expiry
time. xHigherPriorityTaskWoken was initialised to pdFALSE when it
was declared (in this function). */
    if( xTimerResetFromISR( xBacklightTimer,
                            &xHigherPriorityTaskWoken ) != pdPASS
)
    {
        /* The reset command was not executed successfully.  Take
appropriate action here. */
    }

    /* Perform the rest of the key processing here. */

    /* If xHigherPriorityTaskWoken equals pdTRUE, then a context
switch should be performed.  The syntax required to perform a
context switch from inside an ISR varies from port to port, and
from compiler to compiler. Inspect the demos for the port you are
using to find the actual syntax required. */
    if( xHigherPriorityTaskWoken != pdFALSE )
    {
        /* Call the interrupt safe yield function here (actual
function depends on the FreeRTOS port being used). */
    }
}
```

```
void main( void )
{
    long x;

    /* Create then start the one-shot timer that is responsible for
turning the back-light off if no keys are pressed within a 5 second
period. */
    xBacklightTimer = xTimerCreate
              (
                   /* Just a text name, not used by the RTOS kernel. */
                   "BacklightTimer",
                   /* The timer period in ticks. */
                   ( 5000 / portTICK_PERIOD_MS),
                   /* The timer is a one-shot timer. */
                   pdFALSE,
                   /* The id is not used by the callback so can take any
                   value. */
                   0,
                   /* The callback function that switches the LCD back-
light off. */
                   vBacklightTimerCallback
               );

    if( xBacklightTimer == NULL )
    {
        /* The timer was not created. */
    }
    else
    {
        /* Start the timer.  No block time is specified, and even if one
was it would be ignored because the RTOS scheduler has not yet been
started. */
        if( xTimerStart( xBacklightTimer, 0 ) != pdPASS )
        {
            /* The timer could not be set into the Active state. */
        }
    }

    /* ...
     Create tasks here.
    ... */

    /* Starting the RTOS scheduler will start the timer running as it has
    already been set into the active state. */
    vTaskStartScheduler();

    /* Should not reach here. */
    for( ;; );
}
```

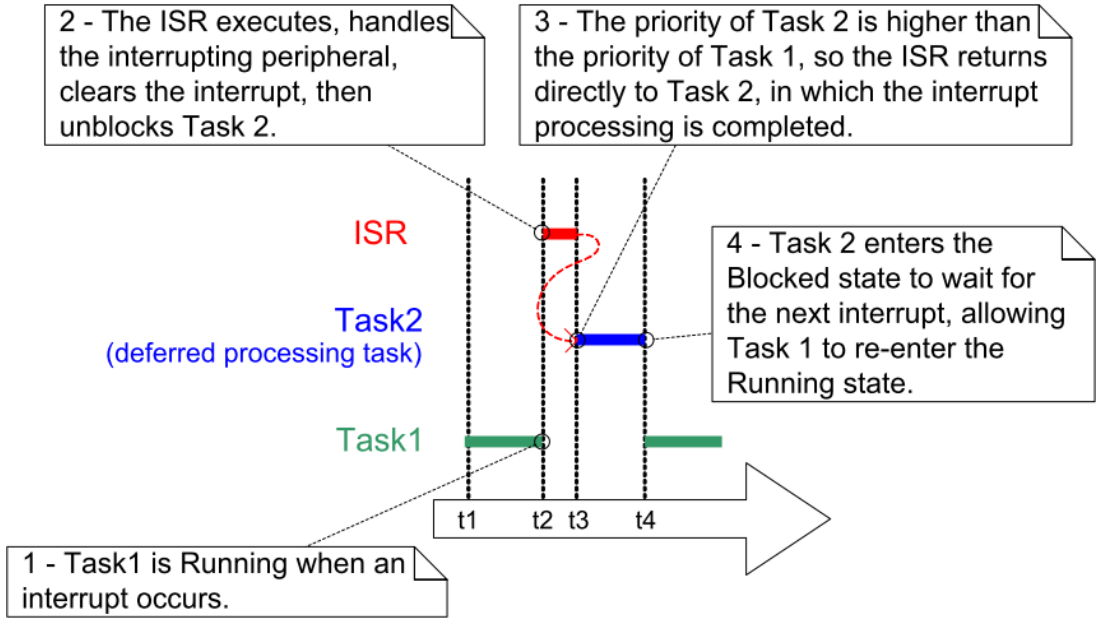
Μέγεθος ISR – Keep it short!

Κίνητρο:

- Δεν καθυστερούμε υπερβολικά την εκτέλεση των υπόλοιπων tasks
- Μειώνουμε το «jitter» στους χρόνους εκτέλεσης των tasks
- Επιτρέπουμε την αποδοχή νέων διακοπών
- Μειώνουμε την πιθανότητα για interrupt nesting
- Περιορίζουμε την πολυπλοκότητα σχετική με την προστασία κοινών πόρων

Deferred interrupt processing

- Τι γίνεται όταν ένα event απαιτεί αρκετό χρόνο επεξεργασίας (π.χ. λήψη ενός network packet από Wifi Controller απαιτεί εκτέλεση TCP/IP κώδικα);
- Deferred interrupt processing
  - task υψηλής προτεραιότητας για να συνεχίσει αμέσως το interrupt processing
  - απαιτεί έναν μηχανισμό συγχρονισμού μεταξύ ISR και task (binary semaphores, task notifications)



Completing interrupt processing in a high priority task

Δραστηριότητα

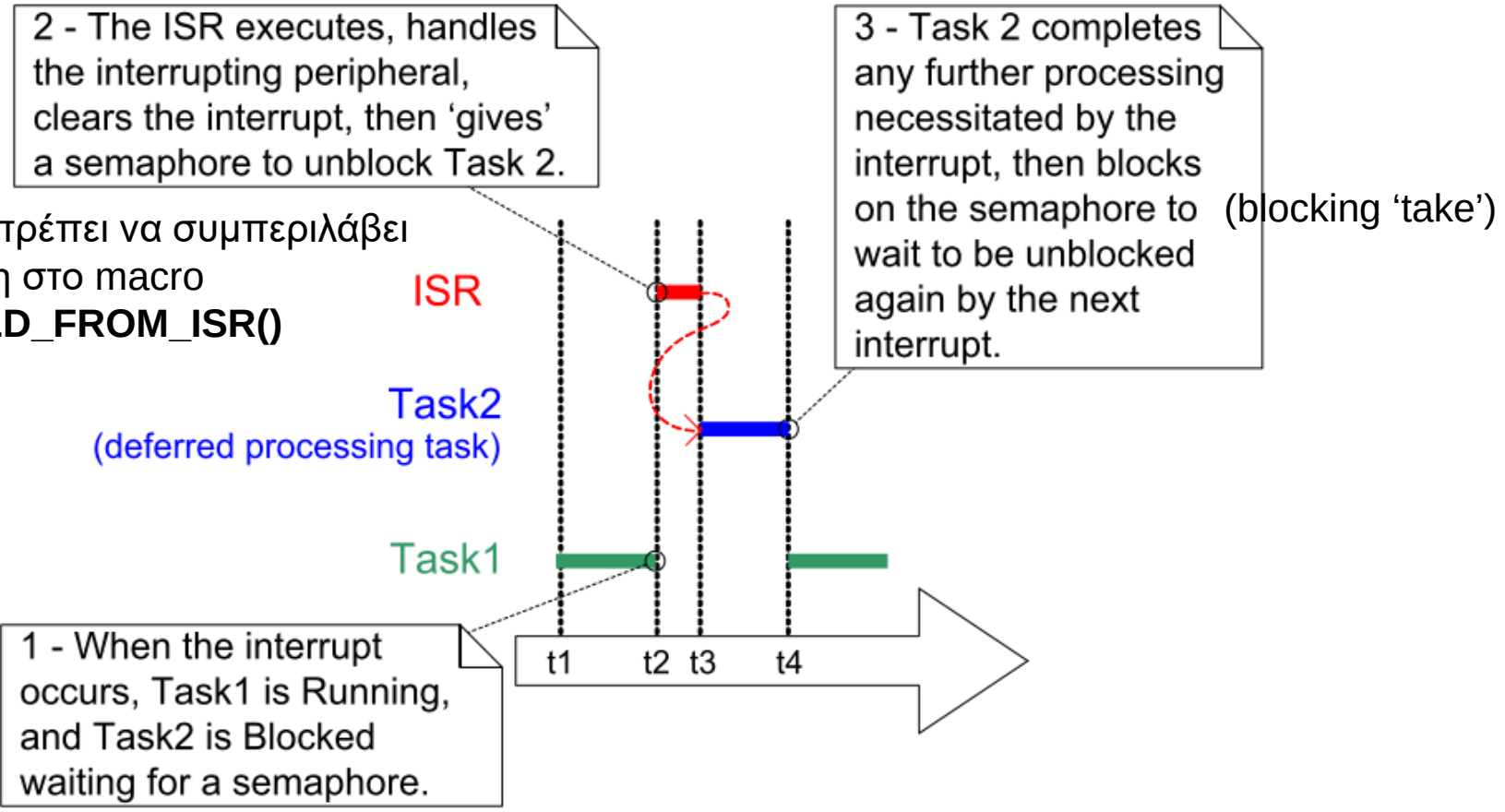
Πότε είναι καλύτερο να εκτελούμε όλη την επεξεργασία που απαιτείται από ένα interrupt στη ISR και πότε είναι καλύτερο να αναθέσουμε μέρος της επεξεργασίας σε ένα task;

- Όταν η επεξεργασία που απαιτείται από τη διακοπή δεν είναι απλή – π.χ. analog to digital conversion + filtering
- Όταν πρέπει να γράψουμε κάτι σε μια κονσόλα ή να γίνει κατανομή μνήμης
- Όταν η επεξεργασία δεν είναι ντετερμινιστική

Συγχρονισμός

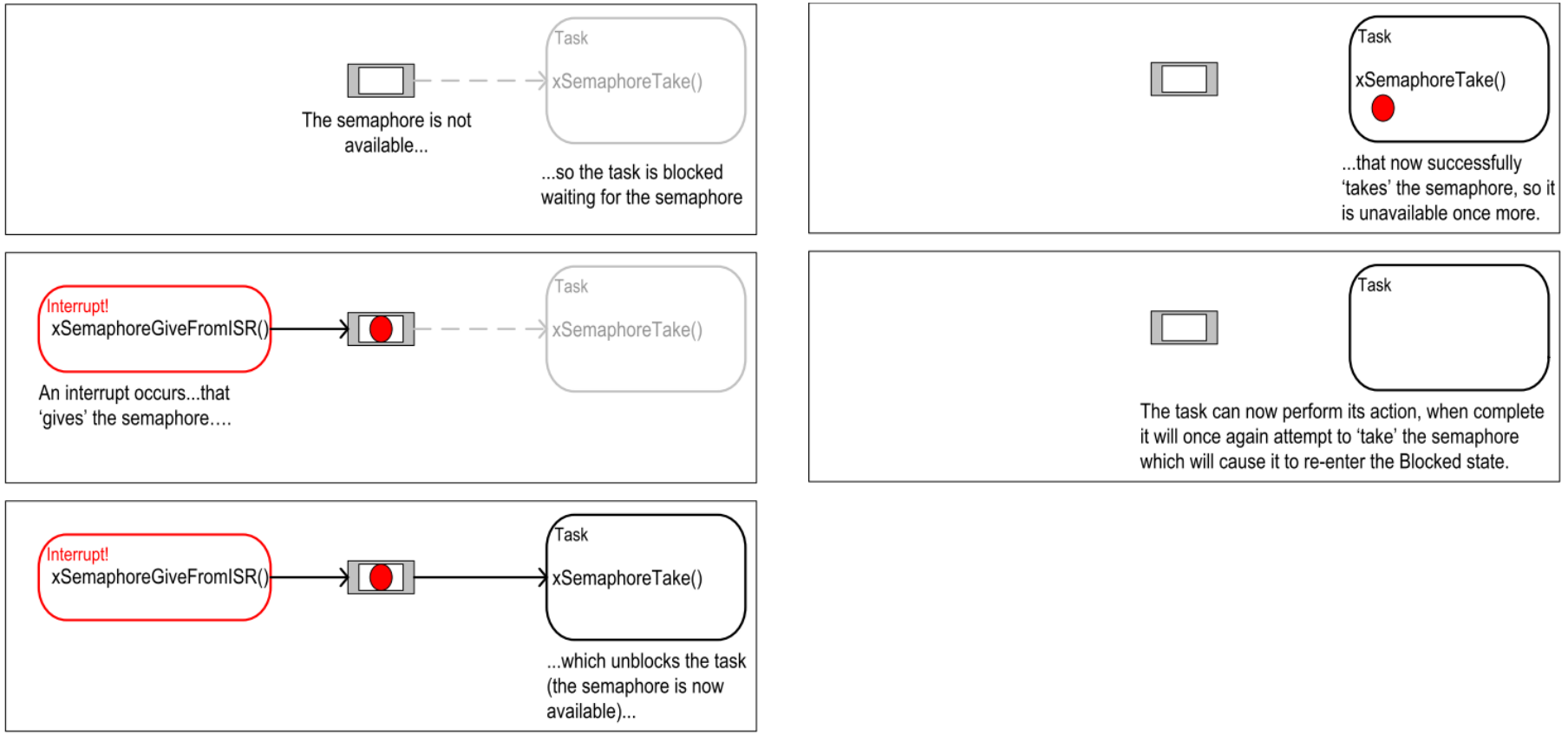
Συγχρονισμός μεταξύ ISR και task

Επίσης, πρέπει να συμπεριλάβει μια κλήση στο macro **portYIELD_FROM_ISR()**



Using a binary semaphore to implement deferred interrupt processing

Συγχρονισμός μεταξύ ISR και task



Using a binary semaphore to synchronize a task with an interrupt

Υλοποίηση παραδείγματος (1/4)

Συγχρονισμός μεταξύ ISR και task με χρήση binary semaphore

```
/* The number of the software interrupt used in this example.  The code shown is from
the Windows project, where numbers 0 to 2 are used by the FreeRTOS Windows port
itself, so 3 is the first number available to the application. */
#define mainINTERRUPT_NUMBER    3

static void vPeriodicTask( void *pvParameters )
{
const TickType_t xDelay500ms = pdMS_TO_TICKS( 500UL );

    /* As per most tasks, this task is implemented within an infinite loop. */
    for( ;; )
    {
        /* Block until it is time to generate the software interrupt again. */
        vTaskDelay( xDelay500ms );

        /* Generate the interrupt, printing a message both before and after
        the interrupt has been generated, so the sequence of execution is evident
        from the output.

        The syntax used to generate a software interrupt is dependent on the
        FreeRTOS port being used.  The syntax used below can only be used with
        the FreeRTOS Windows port, in which such interrupts are only simulated. */
        vPrintString( "Periodic task - About to generate an interrupt.\r\n" );
        vPortGenerateSimulatedInterrupt( mainINTERRUPT_NUMBER );
        vPrintString( "Periodic task - Interrupt generated.\r\n\r\n\r\n" );
    }
}
```

Implementation of the task that periodically generates a software interrupt

Υλοποίηση παραδείγματος (2/4)

Συγχρονισμός μεταξύ ISR και task με χρήση binary semaphore

```
static void vHandlerTask( void *pvParameters )
{
    /* As per most tasks, this task is implemented within an infinite loop. */
    for( ;; )
    {
        /* Use the semaphore to wait for the event.  The semaphore was created
        before the scheduler was started, so before this task ran for the first
        time.  The task blocks indefinitely, meaning this function call will only
        return once the semaphore has been successfully obtained - so there is
        no need to check the value returned by xSemaphoreTake(). */
        xSemaphoreTake( xBinarySemaphore, portMAX_DELAY );

        /* To get here the event must have occurred.  Process the event (in this
        Case, just print out a message). */
        vPrintString( "Handler task - Processing event.\r\n" );
    }
}
```

The implementation of the task to which the interrupt processing is deferred (the task that synchronizes with the interrupt)

Υλοποίηση παραδείγματος (3/4)

Συγχρονισμός μεταξύ ISR και task με χρήση binary semaphore

```
static uint32_t ulExampleInterruptHandler( void )
{
BaseType_t xHigherPriorityTaskWoken;

    /* The xHigherPriorityTaskWoken parameter must be initialized to pdFALSE as
    it will get set to pdTRUE inside the interrupt safe API function if a
    context switch is required. */
    xHigherPriorityTaskWoken = pdFALSE;

    /* 'Give' the semaphore to unblock the task, passing in the address of
    xHigherPriorityTaskWoken as the interrupt safe API function's
    pxHigherPriorityTaskWoken parameter. */
    xSemaphoreGiveFromISR( xBinarySemaphore, &xHigherPriorityTaskWoken );

    /* Pass the xHigherPriorityTaskWoken value into portYIELD_FROM_ISR().  If
    xHigherPriorityTaskWoken was set to pdTRUE inside xSemaphoreGiveFromISR()
    then calling portYIELD_FROM_ISR() will request a context switch.  If
    xHigherPriorityTaskWoken is still pdFALSE then calling
    portYIELD_FROM_ISR() will have no effect. Unlike most FreeRTOS ports, the
    Windows port requires the ISR to return a value - the return statement
    is inside the Windows version of portYIELD_FROM_ISR(). */
    portYIELD_FROM_ISR( xHigherPriorityTaskWoken );
}
```

The ISR for the software interrupt

Υλοποίηση παραδείγματος (4/4)

Συγχρονισμός μεταξύ ISR και task με χρήση binary semaphore

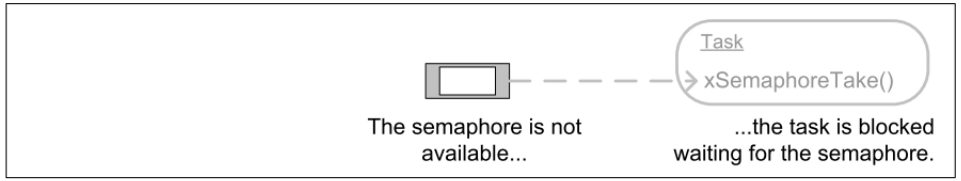
```
int main( void )
{
    /* Before a semaphore is used it must be explicitly created.  In this example
    a binary semaphore is created. */
    xBinarySemaphore = xSemaphoreCreateBinary();

    /* Check the semaphore was created successfully. */
    if( xBinarySemaphore != NULL )
    {
        /* Create the 'handler' task, which is the task to which interrupt
        processing is deferred.  This is the task that will be synchronized with
        the interrupt.  The handler task is created with a high priority to ensure
        it runs immediately after the interrupt exits.  In this case a priority of
        3 is chosen. */
        xTaskCreate( vHandlerTask, "Handler", 1000, NULL, 3, NULL );

        /* Create the task that will periodically generate a software interrupt.
        This is created with a priority below the handler task to ensure it will
        get preempted each time the handler task exits the Blocked state. */
        xTaskCreate( vPeriodicTask, "Periodic", 1000, NULL, 1, NULL );

        /* Install the handler for the software interrupt.  The syntax necessary
        to do this is dependent on the FreeRTOS port being used.  The syntax
        shown here can only be used with the FreeRTOS windows port, where such
        interrupts are only simulated. */
        vPortSetInterruptHandler( mainINTERRUPT_NUMBER, ulExampleInterruptHandler );

        /* Start the scheduler so the created tasks start executing. */
        vTaskStartScheduler();
    }

    /* As normal, the following line should never be reached. */
    for( ;; );
}
```

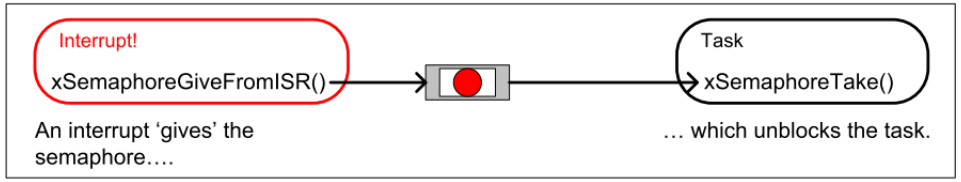
Δραστηριότητα

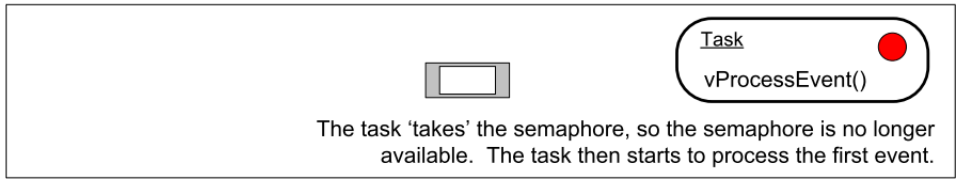
- Με αναφορά στην προηγούμενη υλοποίηση:
  - Τι θα συμβεί αν ένα δεύτερο interrupt εμφανιστεί, ενώ το task εξυπηρετεί ακόμη το πρώτο;
  - Τι θα συμβεί αν ένα τρίτο interrupt εμφανιστεί, ενώ το task εξυπηρετεί ακόμη το πρώτο;

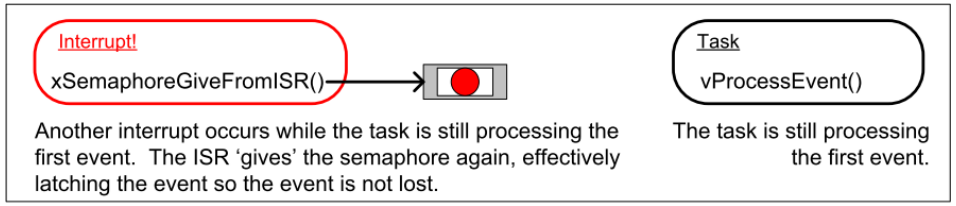
Δραστηριότητα

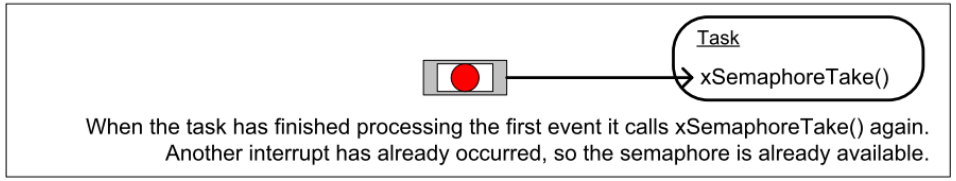
Τι θα συμβεί αν ένα δεύτερο interrupt εμφανιστεί, ενώ το task εξυπηρετεί ακόμη το πρώτο;

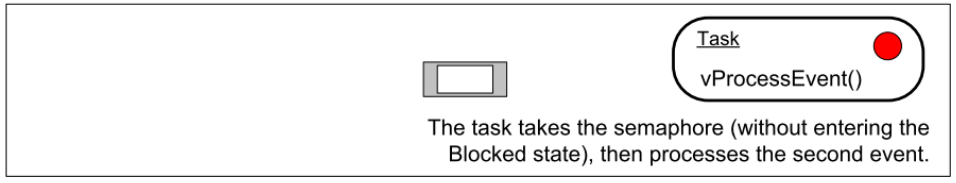






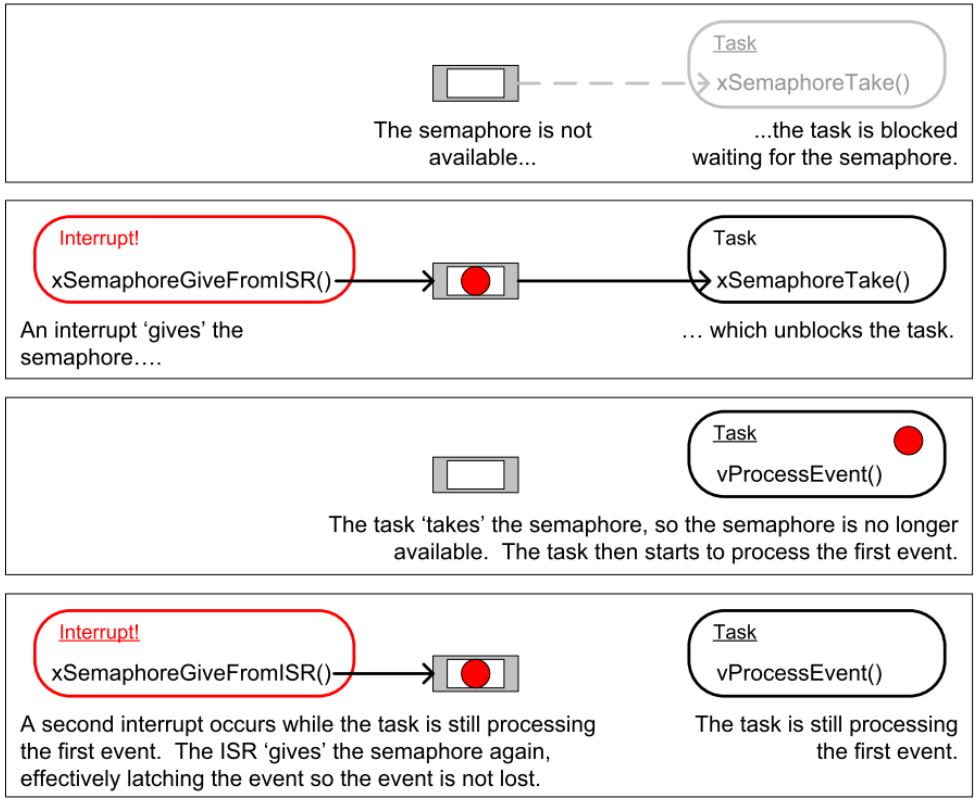


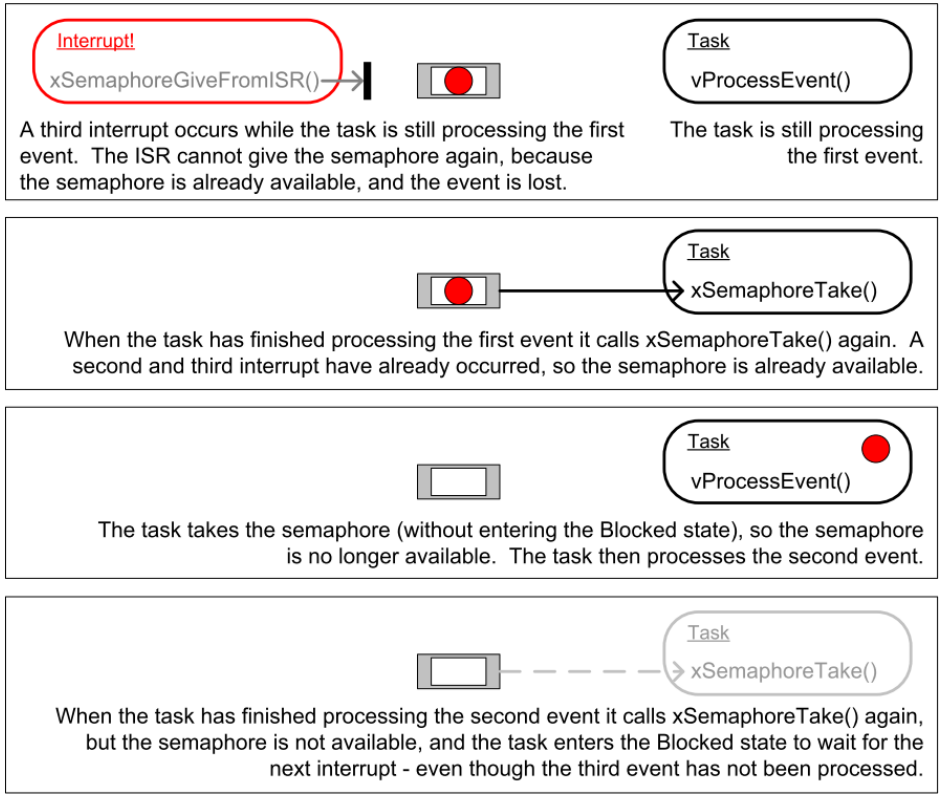




Δραστηριότητα

Τι θα συμβεί αν ένα τρίτο interrupt εμφανιστεί, ενώ το task εξυπηρετεί ακόμη το πρώτο;



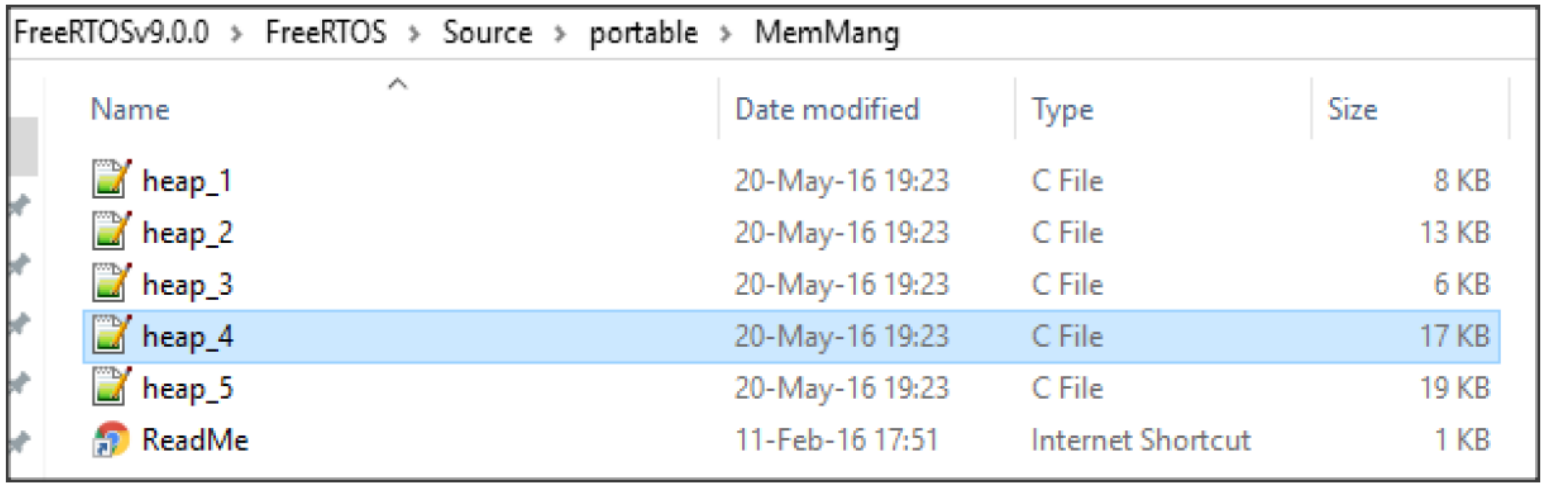


UART receive handler


```
static void vUARTReceiveHandlerTask( void *pvParameters )
{
/* xMaxExpectedBlockTime holds the maximum time expected between two interrupts. */
const TickType_t xMaxExpectedBlockTime = pdMS_TO_TICKS( 500 );

    /* As per most tasks, this task is implemented within an infinite loop. */
    for( ;; )
    {
        /* The semaphore is 'given' by the UART's receive (Rx) interrupt.  Wait a
        maximum of xMaxExpectedBlockTime ticks for the next interrupt. */
        if( xSemaphoreTake( xBinarySemaphore, xMaxExpectedBlockTime ) == pdPASS )
        {
            /* The semaphore was obtained.  Process ALL pending Rx events before
            calling xSemaphoreTake() again.  Each Rx event will have placed a
            character in the UART's receive FIFO, and UART_RxCount() is assumed to
            return the number of characters in the FIFO. */
            while( UART_RxCount() > 0 )
            {
                /* UART_ProcessNextRxEvent() is assumed to process one Rx character,
                reducing the number of characters in the FIFO by 1. */
                UART_ProcessNextRxEvent();
            }

            /* No more Rx events are pending (there are no more characters in the
            FIFO), so loop back and call xSemaphoreTake() to wait for the next
            interrupt.  Any interrupts occurring between this point in the code and
            the call to xSemaphoreTake() will be latched in the semaphore, so will
            not be lost. */
        }
        else
        {
            /* An event was not received within the expected time.  Check for, and if
            necessary clear, any error conditions in the UART that might be
            preventing the UART from generating any more interrupts. */
            UART_ClearErrors();
        }
    }
}
```

Δυναμική Μνήμη

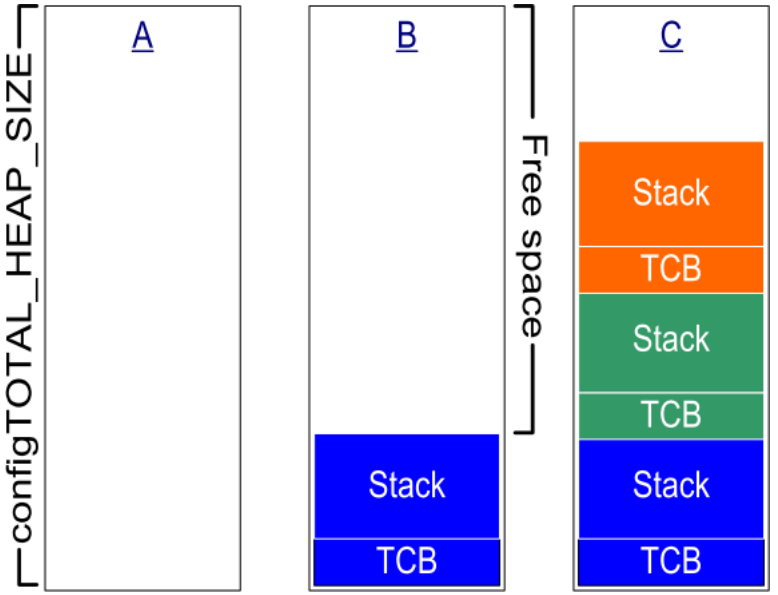
Δυναμική μνήμη στο FreeRTOS



- Υλοποίηση των συναρτήσεων **pvPortMalloc()** και **vPortFree()**
- Γιατί όχι χρήση των C standard συναρτήσεων malloc() και free();
- Μέχρι την έκδοση V9.0.0 ο χώρος μνήμης για τα αντικείμενα του πυρήνα παραχωρείται σε χρόνο εκτέλεσης (run-time).
- Από την έκδοση V9.0.0 δίνεται η δυνατότητα στατικής παραχώρησης μνήμης σε χρόνο μεταγλώττισης (compilation-time).

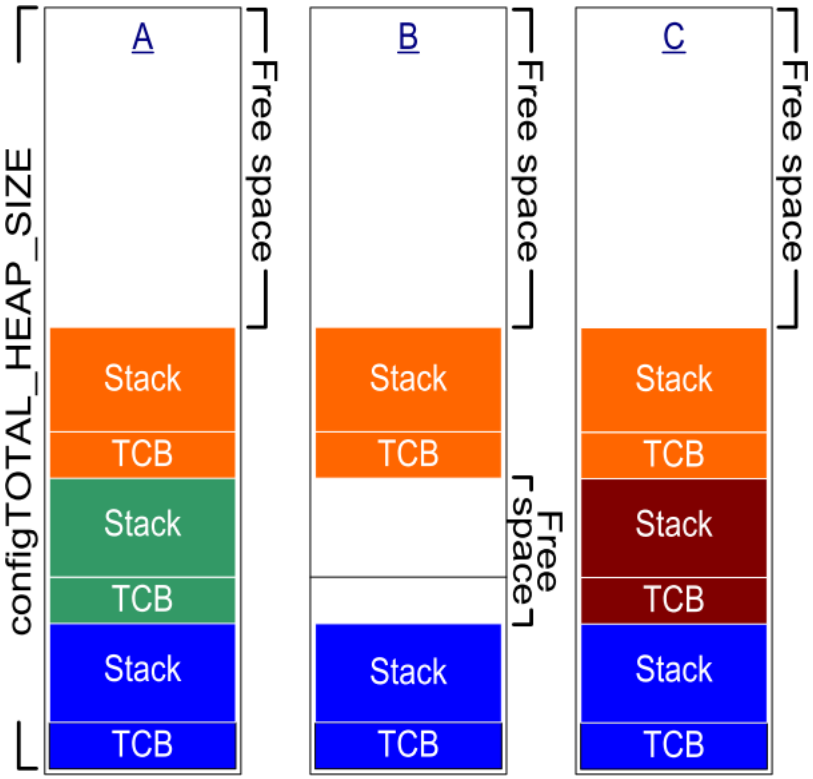
Heap_1

- Καταλληλότητα για συστήματα τα οποία δημιουργούν όλα τα αντικείμενα πυρήνα (π.χ. tasks) πριν ξεκινήσει να τρέχει ο scheduler
- Μικρό μέγεθος και απλότητα
- Δεν χρειάζεται η vPortFree(), απλή υλοποίηση και ντετερμινιστικότητα της pvPortMalloc()
- Πίνακας C μεγέθους "**configTOTAL_HEAP_SIZE**" (FreeRTOSConfig.h)
- Ο πίνακας αυτός υποδιαιρείται σε μικρότερα block



Heap_2

- Καταλληλότητα για εφαρμογές που δημιουργούν και διαγράφουν αντικείμενα πυρήνα δυναμικά
- Υλοποιείται και η συνάρτηση vPortFree()
- Αλγόριθμος best-fit για εύρεση ελεύθερου block μνήμης
- Δεν υλοποιείται συνένωση γειτονικών ελεύθερων block
- Πιθανός κατακερματισμός
- Καλύτερη επιλογή Heap_4
- Δεν είναι ντετερμινιστική
- Ταχύτερη από malloc() / free()

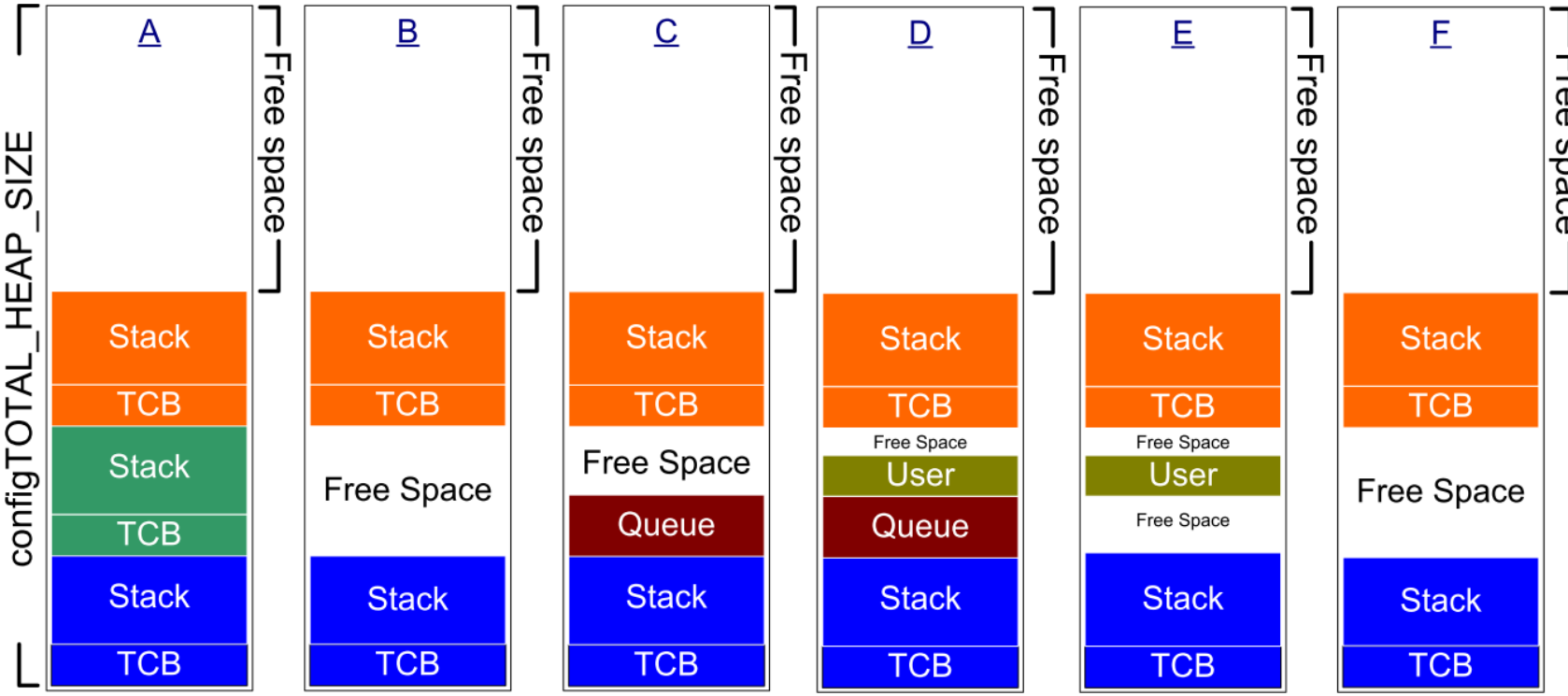


Heap_3

- Σε αυτή τη διάρθρωση pvPortMalloc() και vPortFree() χρησιμοποιούν τις standard malloc() και free()
- Το μέγεθος σωρού καθορίζεται από το linker configuration
- thread-safe με απενεργοποίηση του scheduler πριν την κλήση των malloc() και free()

Heap_4

- Για την pvPortMalloc() υλοποιεί αλγόριθμο first-fit
- Συνένωση γειτονικών ελεύθερων block σε ένα ενιαίο block.
- Δυνατότητα ορισμού της διεύθυνσης του πίνακα σωρού
- Δεν είναι ντετερμινιστική, γρηγορότερη από malloc() / free().



Heap_5

- Παρόμοια με την την Heap_4
- Δυνατότητα παραχώρησης μνήμης από μη συνεχόμενα τμήματα διευθύνσεων.
- Κλήση της **vPortDefineHeapRegions()** → ορισμός διακριτών μη συνεχόμενων περιοχών μνήμης ως Heap
  - Η συνάρτηση αναμένει σαν όρισμα έναν δείκτη σε πίνακα των C δομών τύπου HeapRegion

```
typedef struct HeapRegion
{
    /* The start address of a block of memory that will be part of the heap.*/
    uint8_t *pucStartAddress;

    /* The size of the block of memory in bytes. */
    size_t xSizeInBytes;

} HeapRegion_t;
```

The HeapRegion_t structure

Δυναμική μνήμη στο FreeRTOS

- Διαφορετικές επιλογές για τη δυναμική διαχείριση μνήμης
  - Απλότητα, για εφαρμογές αυστηρά πραγματικού χρόνου (hard-real time) (Heap_1)
  - Ευελιξία και λειτουργικότητα (Heap_4)
- Οι συναρτήσεριις pvPortMalloc() και vPortFree() μπορούν να κληθούν και σε user application code στη θέση των malloc() και free()

Σύγκριση FreeRTOS – Linux

FreeRTOS vs Linux

- Δυνατότητες
- Απαιτήσεις πόρων
- Υποστηριζόμενες αρχιτεκτονικές
- Διαχείριση διεργασιών
- Εικονική μνήμη
- Ευκολία ανάπτυξης

Δυνατότητες

FreeRTOS

- Παρέχονται έτοιμες μόνο οι βασικές λειτουργίες πυρήνα ενός ΛΣ: scheduler, μηχανισμοί IPC, συγχρονισμός
- Επιπλέον λειτουργικότητα ως add-on (TCP/IP stack, FAT σύστημα αρχείων)
- Δεν υποστηρίζεται η δυναμική φόρτωση διεργασιών
- Ενιαίο image πυρήνα και εφαρμογής

Linux

- Πληθώρα λειτουργιών όπως γραφικό περιβάλλον, compilers, σύστημα αρχείων, TCP/IP stack, κ.ά.
- Φόρτωση εφαρμογών από σύστημα αρχείων
- Ανεξαρτησία πυρήνα και εφαρμογών

Απαιτήσεις πόρων

FreeRTOS

- Απαιτείται μνήμη της τάξης των μερικών KB σε συσκευές IoT
- 8-bit AVR: 5Kb FLASH μνήμη και 200 bytes RAM (2 tasks, 1 ουρά, 1 σημαφόρο συγχρονισμού)
- Η απαιτούμενη επεξεργαστική ισχύς είναι μικρή λόγω μικρού και απλού κώδικα πυρήνα

Linux

- Απαιτείται μνήμη της τάξης μερικών MB
- Σημαντικές απαιτήσεις σε επεξεργαστική ισχύ ώστε να μειωθεί ο αντίκτυπος των διάφορων overheads

Υποστηριζόμενες αρχιτεκτονικές

FreeRTOS

- Περίπου 160 ports, από 8bit AVR ως 32-bit ARM Cortex-A
- Ports που αξιοποιούν τις δυνατότητες υλικού:
  - FPU (Floating-point Unit)
  - MPU (Memory Protection Unit).

Linux

- Κατά βάση x86, x64, amd64 και high-end ARM
- Παραλλαγές τύπου uClinux για μProcessors περιορισμένων δυνατοτήτων

Διαχείριση διεργασιών (scheduling)

FreeRTOS

- Δυνατότητα hard-real time χρονικών ορίων
- Preemptive δρομολόγηση με προτεραιότητες στα tasks
- Γρήγορο context-switch λόγω ελαφριού πυρήνα

Linux

- Περιορισμοί στα χρονικά όρια που μπορούν να τεθούν
- Επιλογές δρομολόγησης
- Real time scheduler
- Χρονοβόρο context-switch λόγω πολυπλοκότητας διαχείρισης

Εικονική μνήμη

FreeRTOS

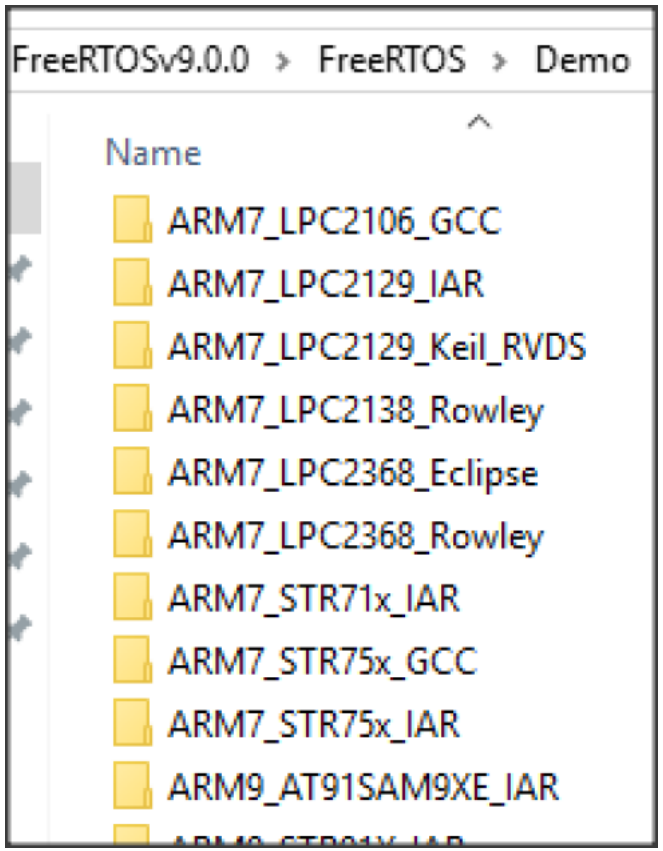
- Στη γενική περίπτωση δεν υλοποιείται εικονική μνήμη
- Για ορισμένες target αρχιτεκτονικές (π.χ. ARM Cortex-M3 και ARM-Cortex-M4F) παρέχεται η δυνατότητα αξιοποίησης μονάδων MPU
  - διαχωρισμός σε Privilleged και User tasks
  - εγγυημένη ανίχνευση stack overflow

Linux

- Πλήρης υποστήριξη σε θέματα εικονικής μνήμης
- Trade-off μεταξύ ευελιξίας και real-time απόδοσης

Ευκολία ανάπτυξης

FreeRTOS



- Πολύ απλό και περιεκτικό documentation και demo για κάθε port.

Linux

- Εκτενέστατο documentation και support από την κοινότητα

Σύνοψη σύγκρισης

- Απλότητα, μικρό μέγεθος και ταχύτητα του FreeRTOS vs. πλήρες εύρος δυνατοτήτων ΛΣ του Linux
- Καταλληλότητα ανάλογα με το σενάριο εφαρμογής
  - Σύστημα που διαβάζει δεδομένα από αισθητήρες, ελέγχει έναν ηλεκτροκινητήρα και χρησιμοποιεί μια οθόνη με απλά γραφικά για παραμετροποίηση και παρακολούθηση
  - Σύστημα που διαβάζει αισθητήρες, στέλνει email με αναφορές μετρήσεων και να τρέχει έναν Web server και Node.js
  - Δυνατότητα χρήσης σε εμπορικά προϊόντα με τροποποίηση του κώδικα του πυρήνα χωρίς να είναι υποχρεωτικό να δημοσιευθεί ο κώδικας της εφαρμογής

Βιβλιογραφία

- Barry, R. (2016). Mastering the FreeRTOS™ Real Time Kernel A Hands-On Tutorial Guide. *Real Time Engineers Ltd*. Available on-line: https://freertos.org/Documentation/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf