# Contents

- Creating a FreeRTOS project
- Heap memory management

# Creating a FreeRTOS project

Barry, R. (2016). *Mastering the freertos real time kernel. Pre-release 161204 Edition*. Real Time Engineers Ltd.

Chapter 1

# Creating a project from scratch

1. Using your chosen tool chain, create a new project that does not yet include any FreeRTOS source files.

   *We use MCUXpresso IDE. An empty project is created by using the new project setup wizard.*


2. Ensure the new project can be built, downloaded to your target hardware, and executed.

# Creating a project from scratch

3. Only when you are sure you already have a working project, add the following FreeRTOS source files to the project:

| File | Location |
|---|---|
| tasks.c | FreeRTOS/Source |
| queue.c | FreeRTOS/Source |
| list.c | FreeRTOS/Source |
| timers.c | FreeRTOS/Source |
| event_groups.c | FreeRTOS/Source |
| All C and assembler files | FreeRTOS/Source/portable/[compiler]/[architecture] |
| heap_n.c | FreeRTOS/Source/portable/MemMang |

# Creating a project from scratch

4. Copy the FreeRTOSConfig.h header file used by the demo project provided for the port in use into the project directory

   *In the example projects, the FreeRTOSConfig.h header file is in the ./inc folder*

5. Add the following directories to the path the project will search to locate header files:
   - FreeRTOS/Source/include
   - FreeRTOS/Source/portable/[compiler]/[architecture]
   - The directory containing the FreeRTOSConfig.h header file

6. Copy the compiler settings from one of the provided examples (or a relevant demo project).

# Creating a project from scratch

7. Install the required FreeRTOS interrupt handlers

   *The MCUXpresso new project wizard generates a CMSIS compliant project. We can therefor use the CMSIS standard names for mapping the interrupt handlers as follows in FreeRTOSConfig.h :*

```
// Used to access OS functions. Is no longer in use but retained
// for backward compatibility
#define vPortSVCHandler SVCall_Handler

// Used by operating systems to force a context switch when no other
// exception is active
#define xPortPendSVHandler PendSV_Handler


// Used as the system tick timer
#define xPortSysTickHandler SysTick_Handler
```

# Data types

**TickType_t**

The number of tick interrupts that have occurred since the FreeRTOS application started is called the tick count. The tick count is used as a measure of time.

Times are specified as multiples of tick periods.

TickType_t can be either an unsigned 16-bit type, or an unsigned 32-bit type, depending on the setting of configUSE_16_BIT_TICKS within FreeRTOSConfig.h

# Data types

**BaseType_t**

This is always defined as the most efficient data type for the architecture.

# Naming conventions

## Variable Names

Variables are prefixed with their type

| Prefix | Datatype |
|---|---|
| c | char |
| s | int16_t (short) |
| l | int32_t (long) |
| x | BaseType_t and any other non-standard types |
| | |
| u | Additionally added if a variable is unsigned |
| p | Additionally added if a variable is a pointer |

# Naming conventions

**Function Names**

Functions are prefixed with both the type they return, and the file they are defined within.

- vTaskPrioritySet()
  returns a **void** and is defined within **task**.c
- xQueueReceive()
  returns a variable of type **BaseType_**t and is defined within **queue**.c
- pvTimerGetTimerID()
  returns a **pointer to void** and is defined within **timers**.c

# Naming conventions

## Macro Names

Written in uppercase and prefixed with lower case letters that indicate where the macro is defined

| Prefix | Location | Example |
|---|---|---|
| port | portable.h or portmacro.h | portMAX_DELAY |
| task | task.h | taskENTER_CRITICAL() |
| pd | projdefs.h | pdTRUE |
| config | FreeRTOSConfig.h | configUSE_PREEMPTION |

# Summary

- Providing information on how a new project can be created.
- Data types
- Naming conventions

# Heap memory management

Barry, R. (2016). *Mastering the freertos real time kernel. Pre-release 161204 Edition*. Real Time Engineers Ltd.
Chapter 2

# Let’s review some C programming concepts

- What does the compiler do?
- What does the linker do?
- What is the stack?
- What is the heap?
- What are the functions malloc() and free() used for?

# Static vs dynamic memory allocation [1)]

- Tasks, queues, etc. are called kernel objects.
- The kernel needs a block of RAM memory for each object to store object specific information
- A kernel object is either allocated dynamically or statically (since V9.0.0)
- Statically allocated objects are allocated at compile-time Advantages:
  - RTOS objects can be placed at specific memory locations
  - The maximum RAM footprint can be determined at link time, rather than run time
  - No need to handle memory allocation failures
- Dynamically allocated objects are allocated at run-time Advantages:
  - Greater simplicity
  - Potentially to minimize the application's maximum RAM usage



1) https://www.freertos.org/Static_Vs_Dynamic_Memory_Allocation.html

# Dynamic memory allocation

- When FreeRTOS requires RAM, it calls pvPortMalloc()
- When FreeRTOS frees RAM, it calls pvPortFree()
- These two functions can also be called from the application code

# Dynamic memory allocation

FreeRTOS comes with five example implementations of these two functions, see FreeRTOS/Source/portable/MemMang

| Example | Description [1)] |
|---|---|
| heap_1.c | The very simplest, does not permit memory to be freed.<br>*Is less useful since FreeRTOS added support for static memory allocation* |
| heap_2.c | Permits memory to be freed but does not coalescence adjacent free blocks. |
| heap_3.c | Simply wraps the standard malloc() and free() for thread safety. |
| **heap_4.c** | Coalescences adjacent free blocks to avoid fragmentation.<br>Includes absolute address placement option.<br>**Is considered the preferred method.** |
| heap_5.c | As per heap_4, with the ability to span the heap across multiple non-adjacent memory areas. |



1) https://www.freertos.org/a00111.html

# Heap Related Utility Functions

```
size_t xPortGetFreeHeapSize( void );
```

**Listing 8. The xPortGetFreeHeapSize() API function prototype**

The number of bytes that remain unallocated in the heap at the time xPortGetFreeHeapSize() is called.

Can be used to optimize the heap size.

# Heap Related Utility Functions

```
size_t xPortGetMinimumEverFreeHeapSize( void );
```

**Listing 9. The xPortGetMinimumEverFreeHeapSize() API function prototype**

The minimum number of unallocated bytes that have existed in the heap since the FreeRTOS application started executing.

# Heap Related Utility Functions

```
void vApplicationMallocFailedHook( void );
```

**Listing 10. The malloc failed hook function name and prototype.**

Hook (callback) function that gets called if a call to pvPortMalloc() fails.

configUSE_MALLOC_FAILED_HOOK must be set to 1.

# Demo FreeRTOS for FRDM-KL25Z

- Demo Week2/Example01
  Dynamic memory allocation
- Demo Week2/Example02
  Static memory allocation

# Dynamic memory considerations

- The FreeRTOS total heap size is configured with configTOTAL_HEAP_SIZE.
- When using heap_4: the heap is not the 'normal' heap, but declared as follows:

```
static uint8_t ucHeap[ configTOTAL_HEAP_SIZE ];
```

- So, increasing the FreeRTOS heap size is done by increasing configTOTAL_HEAP_SIZE and not by updating the linker script.

- The MKL25Z128VLK4 microcontroller contains 16 kB SRAM. Make sure that:

**FreeRTOS heap + linker heap + linker stack + global variables <= 16 kB**

# Dynamic memory considerations

The compiler output provides an indication of the memory that is in use:

```
#define configTOTAL_HEAP_SIZE(  ( size_t ) ( 8 * 1024 ) )

Memory region         Used Size  Region Size  %age Used
   PROGRAM_FLASH:       22640 B       128 KB     17.27%
            SRAM:       10868 B        16 KB     66.33%
Finished building target: Week2 - Example01.axf


#define configTOTAL_HEAP_SIZE(  ( size_t ) ( 16 * 1024 ) )

Memory region         Used Size  Region Size  %age Used
   PROGRAM_FLASH:       22640 B       128 KB     17.27%
            SRAM:       19060 B        16 KB    116.33%
collect2.exe: error: ld returned 1 exit status
make: *** [makefile:40: Week2 - Example01.axf] Error 1
```

# Dynamic memory considerations

The number of kernel objects (tasks, queues, etc.) that can be used in an application thus depends the total available FreeRTOS heap memory, set by `configTOTAL_HEAP_SIZE`.

But also the stack size required by each kernel object.

Although configuring the appropriate stack size for each kernel object can be tedious, FreeRTOS can help!

# Stack Related Utility Functions

```
UBaseType_t uxTaskGetStackHighWaterMark( TaskHandle_t xTask );
```

**Listing 173. The uxTaskGetStackHighWaterMark() API function prototype**

The handle of the task whose stack high water mark is being queried (the subject task)—see the pxCreatedTask parameter of the xTaskCreate() API function for information on obtaining handles to tasks.

A task can query its own stack high water mark by passing NULL in place of a valid task handle.

Is available if `INCLUDE_uxTaskGetStackHighWaterMark` is 1.

# Stack Related Utility Functions

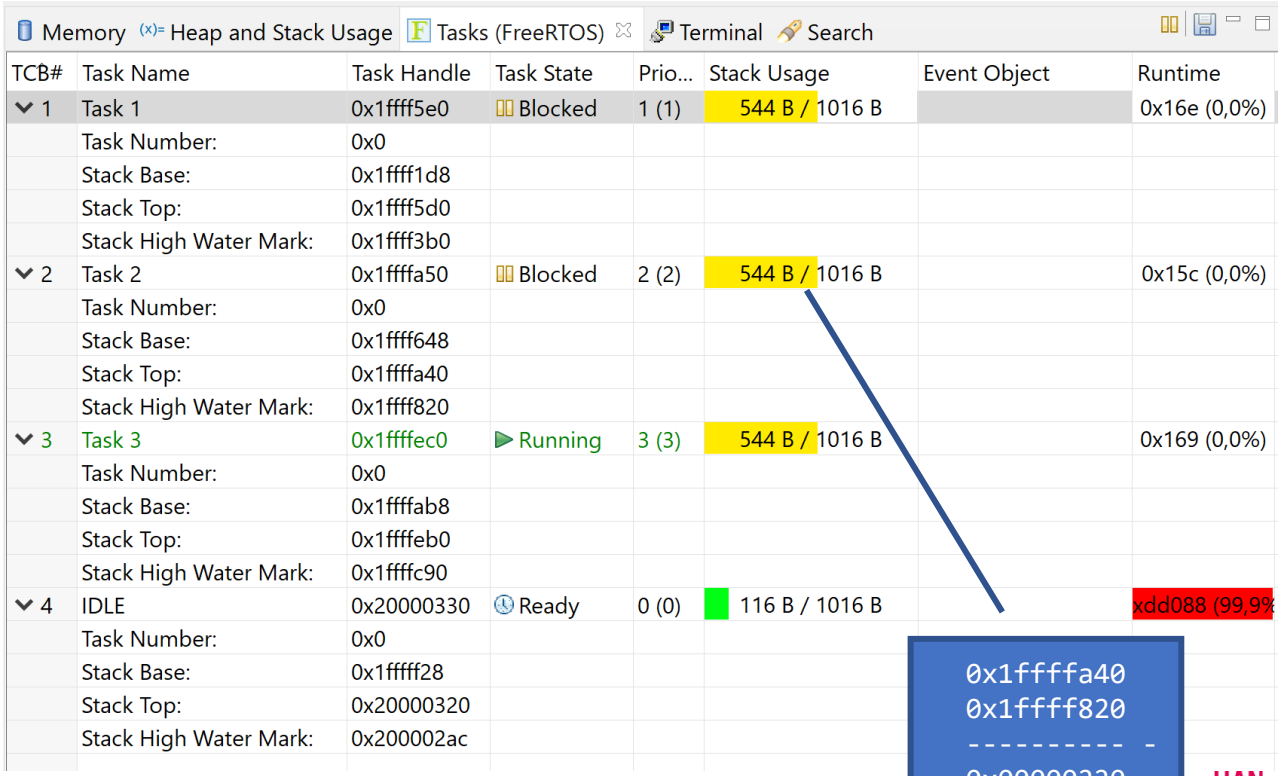
```
UBaseType_t uxTaskGetStackHighWaterMark( TaskHandle_t xTask );
```

**Listing 173. The uxTaskGetStackHighWaterMark() API function prototype**

Returns the minimum amount of remaining stack space that has been available since the task started executing. The closer the high water mark is to zero, the closer the task has come to overflowing its stack.

# Stack Related Utility Functions

The high watermark is also included in the kernel aware debugging feature in MCUXpresso IDE






HAN_UNIVERSITY OF APPLIED SCIENCES

# Stack Related Utility Functions

## Run Time Stack Checking

```
void vApplicationStackOverflowHook( TaskHandle_t *pxTask, signed char *pcTaskName );
```

**Listing 174. The stack overflow hook function prototype**

A hook (callback) function that gets called if a stack overflow is detected.

*The stack overflow hook is provided to make trapping and debugging stack errors easier, but there is no real way to recover from a stack overflow when it occurs.*

pxTask is a pointer to the handle of the task that had a stack overflow.

Note: the time increase to perform a context switch!

# Stack Related Utility Functions

## Run Time Stack Checking

```
void vApplicationStackOverflowHook( TaskHandle_t *pxTask, signed char *pcTaskName );
```

**Listing 174. The stack overflow hook function prototype**

Pointer to the name of the task that had a stack overflow.

# Stack Related Utility Functions

Run Time Stack Checking

Method 1: configCHECK_FOR_STACK_OVERFLOW is set to 1

- The kernel checks that the stack pointer remains within the valid stack space after the context has been saved
- Is quick to execute, but can miss stack overflows that occur between context switches

Method 2: configCHECK_FOR_STACK_OVERFLOW is set to 2

- Performs additional checks
- When a task is created, its stack is filled with a known pattern.
- Tests the last valid 20 bytes of the task stack space to verify that this pattern has not been overwritten.

# Summary

- When FreeRTOS allocates RAM.
- The five example memory allocation schemes supplied with FreeRTOS.
- Which memory allocation scheme to select.
- Utility functions.