FreeRTOS Memory Management

Norman McEntire
norman.mcentire@gmail.com

Textbook Reference

- Mastering the FreeRTOS Real Time Kernel
  by Richard Barry
  - Chapter 2: Heap Memory Management

Topics

- 2.1 Intro and Scope
- 2.2 Example Memory Allocation Schemes
- 2.3 Heap Related Utility Functions

Note

- Starting with FreeRTOS V9.0.0, applications can be **statically allocated memory**
  - Removing the need for a heap memory manager
  - (but for many apps you'll use a heap memory manager…)

2.1 Intro and Scope

- FreeRTOS kernel objects (tasks, queues, semaphores, etc.) can be dynamically allocated at run-time
    - Compared to static allocation at compile time
- FreeRTOS does not use C language malloc() and free() but rather FreeRTOS specific calls
    - Why not use malloc() and free() - see next slide

Issues with malloc() and free()

- Not always available on small embedded systems
- Implementation can be large, taking up valuable code space
- Rarely are the thread safe
- Not deterministic - time to execute can vary
- Can suffer from fragmentation
- Can complicate linker configuration
- Can be source of difficult to debug errors

FreeRTOS and Memory Allocation

- FreeRTOS treats memory allocation as part of the portable layer
- FreeRTOS allocates RAM by calling **pvPortMalloc**()
- FreeRTOS frees RAM by calling **vPortFree**()
- Both **pvPortMalloc**() and **vPortFree**() are public functions - can be called from application code

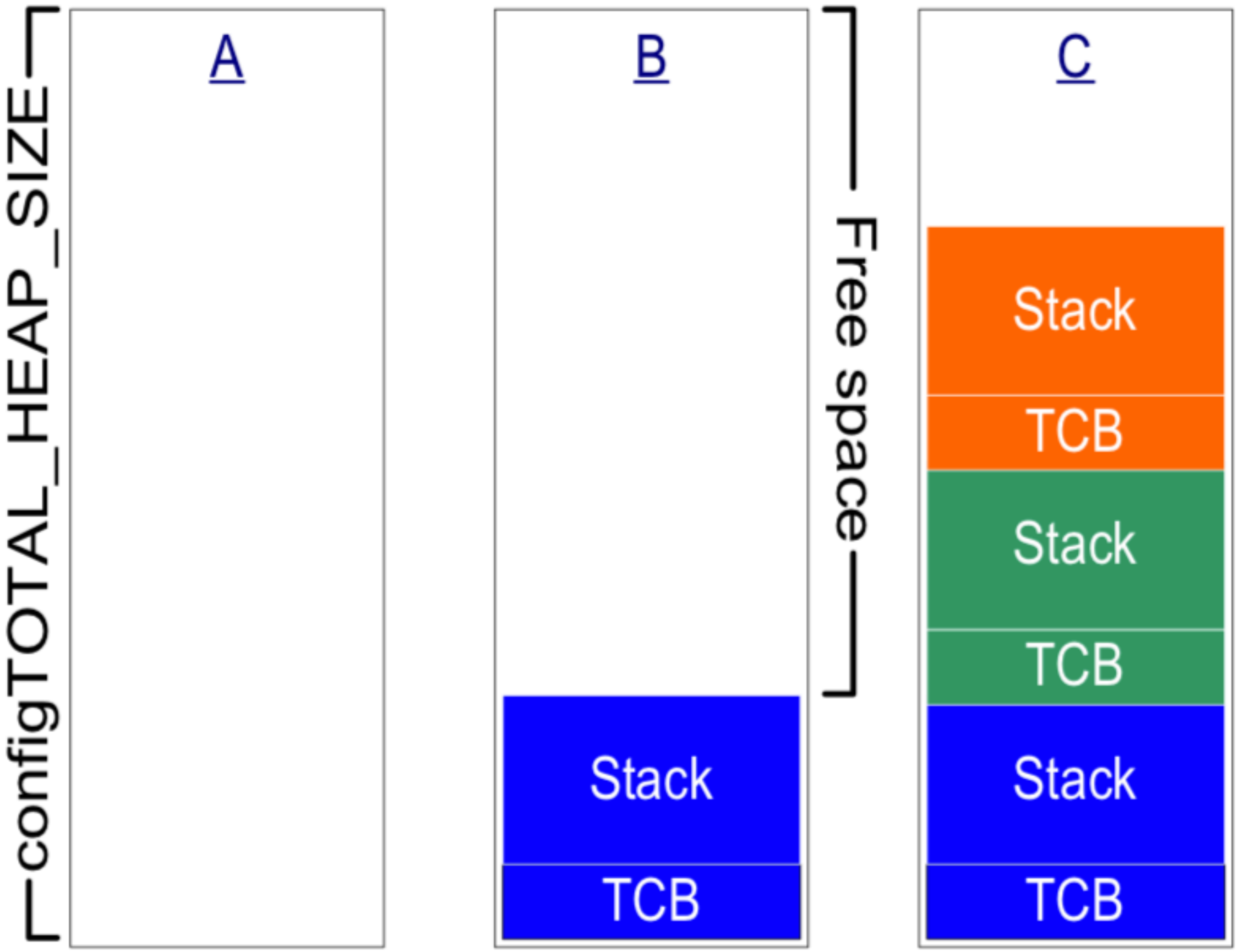
FreeRTOS comes with Five Implementations

- FreeRTOS comes with five implementations of pvPortMalloc() and vPortFree()
- Apps may use one of these example implementations - or use their own
- The five implementations
  - heap_1.c, heap_2.c, heap_3.c, heap_4.c, heap_5.c
  - Located at: FreeRTOS/Source/portable/MemMang

Setting Heap Memory Size

- FreeRTOSConfig.h
  - #define configTOTAL_HEAP_SIZE <SizeInBytes>
- Remember: Each task requires two blocks of memory when allocated
  - TCB - Task Control Block
  - Stack

RAM Allocation
Time A, B, and C



2.2 Example Memory Allocation Schemes

- heap_1.c
- heap_2.c
- heap_3.c
- heap_4.c
- heap_5.c

heap_1.c

- The simplest allocation scheme
- Use this when all allocation done before starting the scheduler
- Only pvPortMalloc() implemented
  - vPortFree() is never used for heap_1.c
- Note: Some commercially critical and/or safety critical systems prohibit use of dynamic memory alloc — hence heap_1.c is best choice
  - Do all memory alloc before the scheduler starts

heap_2.c

- We do not cover this one!
  - No recommended for new designs
    - Included for backwarrds compatibility with older versions of FreeRTOS
- NOTE: Use heap_4.c is a replacement for heap_2.c

heap_3.c

- Uses the standard C library malloc() and free()
    - Size of heap determined by linker configuration
    - configTOTAL_HEAP_SIZE is not used in this configuration
- heap_3.c makes malloc() and free() thread safe by suspending the FreeRTOS scheduler before calling malloc() or free()

heap_4.c

- heap_4.c uses a first-fit algorithm to allocate memory
  - heap_4.c will also combine adjacent free blocks of memory into a single larger block
    - This minimizes the risk of fragmentation
      - Result is the pvPortMalloc() uses the first free block of memory that is large enough to handle the request

heap_5.c

- The algorithm used by heap_5.c to allocate and free memory is identical to that used by heap_4.c
- However, heap_5.c is not limited to a single statically defined array
  - The heap_5.c can rather allocate from multiple and separated memory spaces
- Unlike all other heap_n.c options, the heap_5.c must be initialized using **pvPortDefineHeapRegions**()

vPortDefineHeapRegions()

- void
  vPortDefineHeapRegions(
  const HeapRegion_t *const pxHeapRegions)

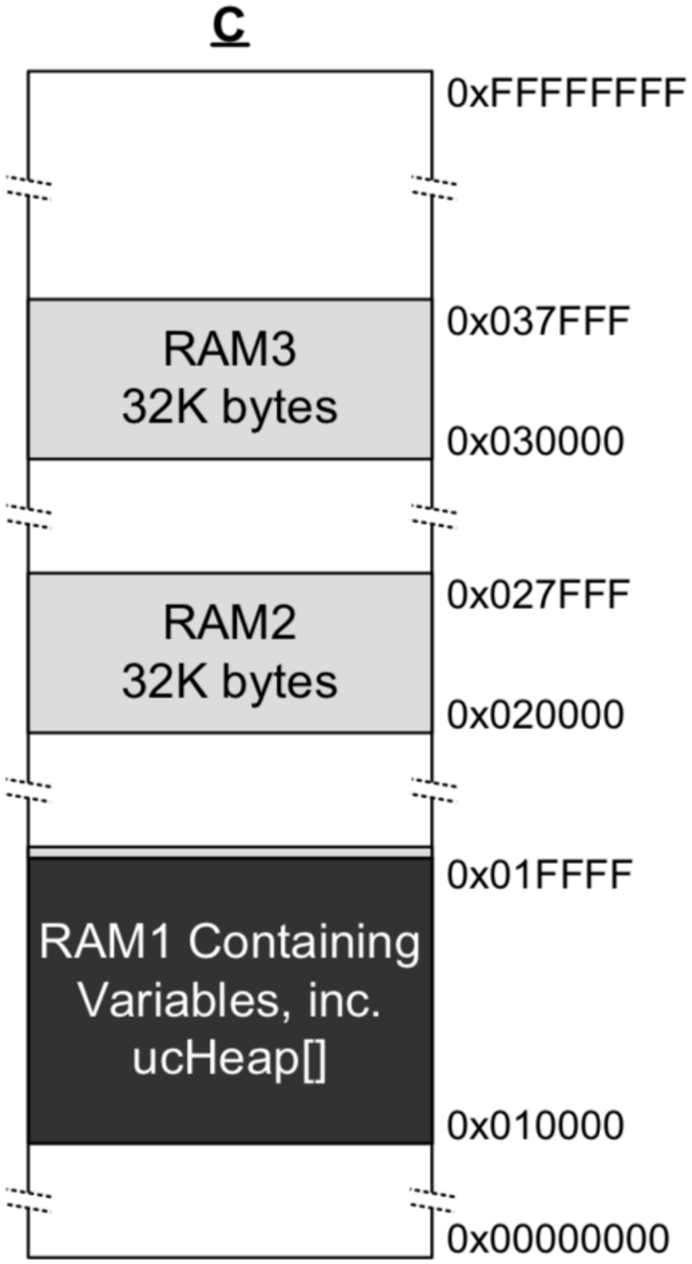
- Each separate memory region is described by HeapRegion_t

  - See next slide for definition of HeapRegion_t

HeapRegion_t

-
  ```
  typedef struct HeapRegion {
      unit8_t *pucStartAddress;
      size_t xSizeInBytes;
  }
  ```

Block Diagram



Code Sample

```
/* Define the start address and size of the two RAM regions not used by the
linker. */
#define RAM2_START_ADDRESS    ( ( uint8_t * ) 0x00020000 )
#define RAM2_SIZE             ( 32 * 1024 )

#define RAM3_START_ADDRESS    ( ( uint8_t * ) 0x00030000 )
#define RAM3_SIZE             ( 32 * 1024 )

/* Declare an array that will be part of the heap used by heap_5.  The array will be
placed in RAM1 by the linker. */
#define RAM1_HEAP_SIZE ( 30 * 1024 )
static uint8_t ucHeap[ RAM1_HEAP_SIZE ];

/* Create an array of HeapRegion_t definitions.  Whereas in Listing 6 the first entry
described all of RAM1, so heap_5 will have used all of RAM1, this time the first
entry only describes the ucHeap array, so heap_5 will only use the part of RAM1 that
contains the ucHeap array.  The HeapRegion_t structures must still appear in start
address order, with the structure that contains the lowest start address appearing
first. */
const HeapRegion_t xHeapRegions[] =
{
    { ucHeap,             RAM1_HEAP_SIZE },
    { RAM2_START_ADDRESS, RAM2_SIZE },
    { RAM3_START_ADDRESS, RAM3_SIZE },
    { NULL,               0         }  /* Marks the end of the array. */
};
```

2.3 Heap Related Utility Functions

- **xPortGetFreeHeapSize**()
  - Returns the number of free bytes in the heap
- Use **xPortGetFreeHeapSize**() to optimize your code
  - After all kernel objects (tasks, queues, etc.) are allocated, if xPortGetFreeHeapSize() still returns some number of bytes, then you can reduce the size of the Heap in the FreeRTOSConfig.h
    - configTOTAL_HEAP_SIZE

xPortGetMinimumEverFreeHeapSize()

- xPortGetMinimumEverFreeHeapSize()
    - Returns the minimum number of unallocated bytes that have ever existed in the heap
        - This is an indication of close the app has come to running out of memory space
- NOTE: This only available for heap_4.c and heap_5.c

Malloc Failed Hook Function

- If pvPortMalloc() cannot return a block of RAM, it will return NULL
- Set configUSE_MALLOC_FAILED_HOOK to 1 in FreeRTOSConfig.h if you want to provide for a failed memory allocation hook
  - void vApplicationMallocFailedHook(void);

Summary

- Intro and Scope
    - pvPortMalloc()
    - vPortFree()
- Example Memory Allocation Schemes
    - heap_1.c, heap_2.c, heap_3.c, heap_4.c, heap_5.c
- 2.3 Heap Related Utility Functions
    - xPortGetFreeHeapSize()
    - xPortGetMinimumEverFreeHeapSize()