FreeRTOS Lab 4
Memory Management

TA: p76134846@gs.ncku.edu.tw

Outline

- Pre Lab
- Memory Management Introduction
- Requirements
- Grading

Pre Lab

- 新增一個新的專案，並以助教提供的 Template 的檔案取代專案中對應的檔案
  - Template 中包含的檔案如下圖所示 (細節稍後說明)
  - FreeRTOS/portable/MemMang 目錄下只保留 heap_2.c
- Lab4 會使用到前幾次lab 使用過的 USART2 與 LED，相關 pin 腳與 console 的設定請自行依照之前 lab 簡報完成設定

Lab 4 Template

- FreeRTOSConfig.h 修改內容: 縮小 heap 以方便觀察記憶體使用狀況

```
#define configTOTAL_HEAP_SIZE                    ( ( size_t ) ( 5 * 1024 ) )
```

- task.h 修改內容: 宣告 huart2 變數以使用 UART 輸出文字到 Console
  - task.h 會被多個 .c 檔 include，所以需要用 extern 避免重複定義變數

```
#include "stm32f4xx_hal.h"

extern UART_HandleTypeDef huart2;
```

- tasks.c 修改內容: 在建立 Task 時透過 UART 輸出訊息到 Console

```
TCB_t *pxNewTCB;
BaseType_t xReturn;
char name[20];
strcpy(name, pcName);
strcat(name, "\n\r");
HAL_UART_Transmit(&huart2, (uint8_t *)name, strlen(name), 0xffff);
```

Lab 4 Template

- heap_2.c 修改內容 1 :在動態分配記憶體時透過 UART 輸出訊息到

```
size_t BlockSize, WantedSize;
char data[80];
WantedSize = xWantedSize;
BlockSize = xWantedSize;
sprintf(data, "pvReturn: %p | heapSTRUCT_SIZE: %0d | WantedSize: %3d | BlockSize: %3d\n\r",
    pvReturn, heapSTRUCT_SIZE, WantedSize, BlockSize);
HAL_UART_Transmit(&huart2, (uint8_t *)data, strlen(data), 0xffff);
```

- portable.h 修改內容: 宣告 `vPrintFreeList` (for lab4)

```
void vPrintFreeList(void) PRIVILEGED_FUNCTION;
```

Lab 4 Template

- heap_2.c 修改內容 2 : 新增空的 `vPrintFreeList` 實作 (for lab4)

```
void vPrintFreeList(void)
{
    /* TODO: implement this function
     *
     * Reference format
     * > sprintf(data, "StartAddress heapSTRUCT_SIZE xBlockSize EndAddress\n\r");
     * > sprintf(data, "%p         %d           %4d         %p\n\r", ...);
     * > sprintf(data, "configADJUSTED_HEAP_SIZE: %0d xFreeBytesRemaining: %0d\n\r", ...);
     */
}
```

Lab 4 Template

- heap_2.c 修改內容 3 : 修改巨集 `prvInsertBlockIntoFreeList` (for lab4)

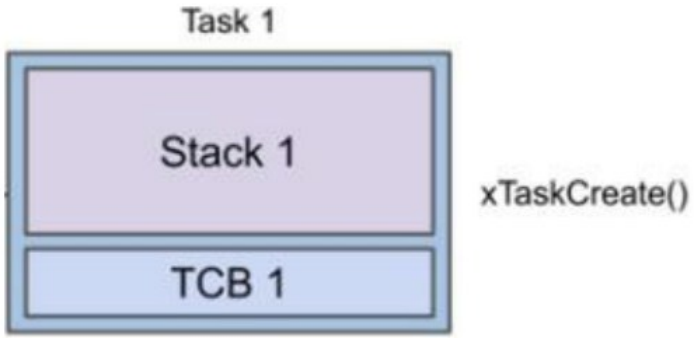
```
#define prvInsertBlockIntoFreeList( pxBlockToInsert )                          \
{                                                                              \
    BlockLink_t *pxIterator;                                                   \
    size_t xBlockSize;                                                         \
    BlockLink_t *pxPrevious, *pxBlockHolding = pxBlockToInsert;                \
    /* TODO: Merge free blocks                                              */\
    /*                                                                      */\
    /* HINT                                                                 */\
    /* 1. You may need to use the `pxPrevious` pointer to keep trace of the */\
    /*    previous block.                                                   */\
    /* 2. Because it is a macro, use `pxBlockHolding` instead of `pxBlockToInsert`*/\
    /*    below, or you may encounter a problem.                            */\
    /* 3. To Keep the free blocks sorted in block size after merging, you may */\
    /*    want to take adjacent free blocks out.                            */\
    /* 4. Beware the start of the address when merge.                       */\
    /*                                                                      */\
```

Lab 4 Template

- main.c 修改內容: 新增 6 個 Tasks 以觀察記憶體使用情況
  - red_LED_task, green_LED_task: 持續閃爍LED，會持續占用同一塊記憶體
  - task1, task2, task3: 刪除自己的task，用來釋放記憶體
  - print_task: 定期呼叫vPrintFreeList 以觀察記憶體使用狀況

Lab 4 Template 預期輸出

- 若設定無誤，使用 Template 的程式碼直接執行後應該會看到 LED 燈閃爍，並可在 console 可以看到與下方右圖相似的輸出訊息
  - TCB 與 Stack 會分開分配，因此每個 Task 都會呼叫兩次 `pvPortMalloc`




```
RED_LED
pvReturn: 0x20000240 | heapSTRUCT_SIZE: 8 | WantedSize: 400 | BlockSize: 408
pvReturn: 0x200003d8 | heapSTRUCT_SIZE: 8 | WantedSize:  88 | BlockSize:  96
TASK1
pvReturn: 0x20000438 | heapSTRUCT_SIZE: 8 | WantedSize: 200 | BlockSize: 208
pvReturn: 0x20000508 | heapSTRUCT_SIZE: 8 | WantedSize:  88 | BlockSize:  96
TASK2
pvReturn: 0x20000568 | heapSTRUCT_SIZE: 8 | WantedSize: 120 | BlockSize: 128
pvReturn: 0x200005e8 | heapSTRUCT_SIZE: 8 | WantedSize:  88 | BlockSize:  96
GREEN_LED
pvReturn: 0x20000648 | heapSTRUCT_SIZE: 8 | WantedSize: 520 | BlockSize: 528
pvReturn: 0x20000858 | heapSTRUCT_SIZE: 8 | WantedSize:  88 | BlockSize:  96
TASK3
pvReturn: 0x200008b8 | heapSTRUCT_SIZE: 8 | WantedSize: 160 | BlockSize: 168
pvReturn: 0x20000960 | heapSTRUCT_SIZE: 8 | WantedSize:  88 | BlockSize:  96
PRINT
pvReturn: 0x200009c0 | heapSTRUCT_SIZE: 8 | WantedSize: 520 | BlockSize: 528
pvReturn: 0x20000bd0 | heapSTRUCT_SIZE: 8 | WantedSize:  88 | BlockSize:  96
IDLE
pvReturn: 0x20000c30 | heapSTRUCT_SIZE: 8 | WantedSize: 520 | BlockSize: 528
pvReturn: 0x20000e40 | heapSTRUCT_SIZE: 8 | WantedSize:  88 | BlockSize:  96
```

FreeRTOS Memory Management

- FreeRTOS kernel objects are dynamically allocated at run-time.
  - Including
    - Tasks
    - Software Timers
    - Queues
    - Event Groups
    - Semaphores and Mutexes
  - Benefits
    - reduces design and planning effort
    - minimizes the RAM footprint

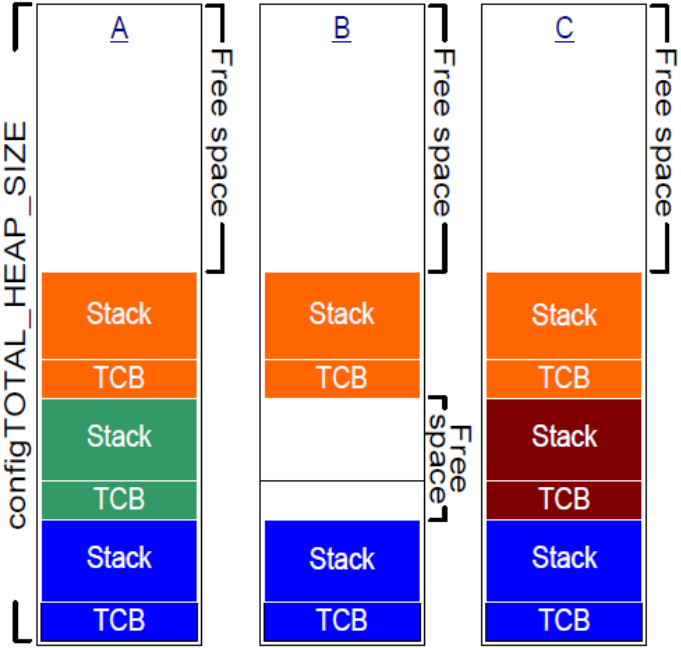
FreeRTOS Memory Management

- FreeRTOS puts memory allocation in the portable layer.
- Interfaces: **`pvPortMalloc()`** and **`vPortFree()`**, declared in portable.h.
- FreeRTOS provides 5 example implementations.
  - heap_x.c where x = 1-5
  - use one heap_x.c or provide your own (if you need dyma. mem. alloc.).
  - we take heap_2.c for our lab.

Heap_2

- Heap (with size **`configTOTAL_HEAP_SIZE`**) is statically allocated.
- Use best-fit method to allocate memory, and allow memory to be freed.
  - A free list sorted with ascending size is maintained.
- Split (the best-fit) free memory when allocation, but NO MERGE when free
  - May cause fragmentation.
- Suitable for an application that creates and deletes tasks repeatedly, given that the tasks have the same sizes of stack.
  - TCB and task stacks can be reused repeatedly.
- Not deterministic, but faster than most std. lib. `malloc()/free()` implementations.

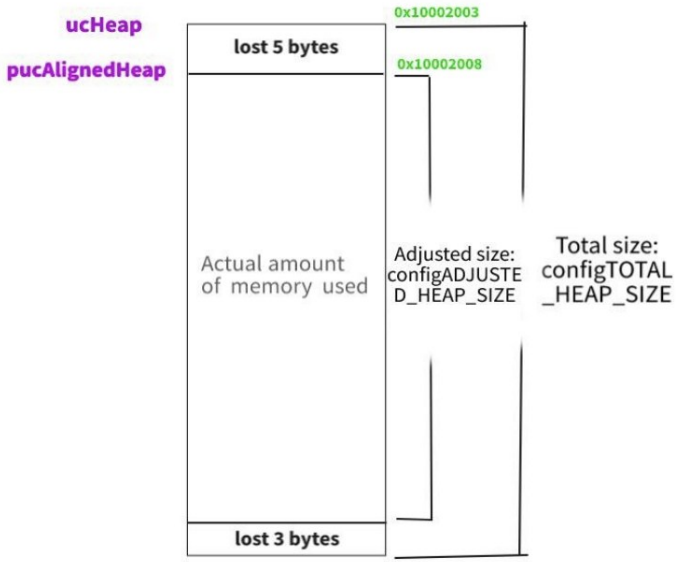
Without Merge



https://github.com/FreeRTOS/FreeRTOS-Kernel-Book/releases/download/V1.1.0/Mastering-the-FreeRTOS-Real-Time-Kernel.v1.1.0.pdf

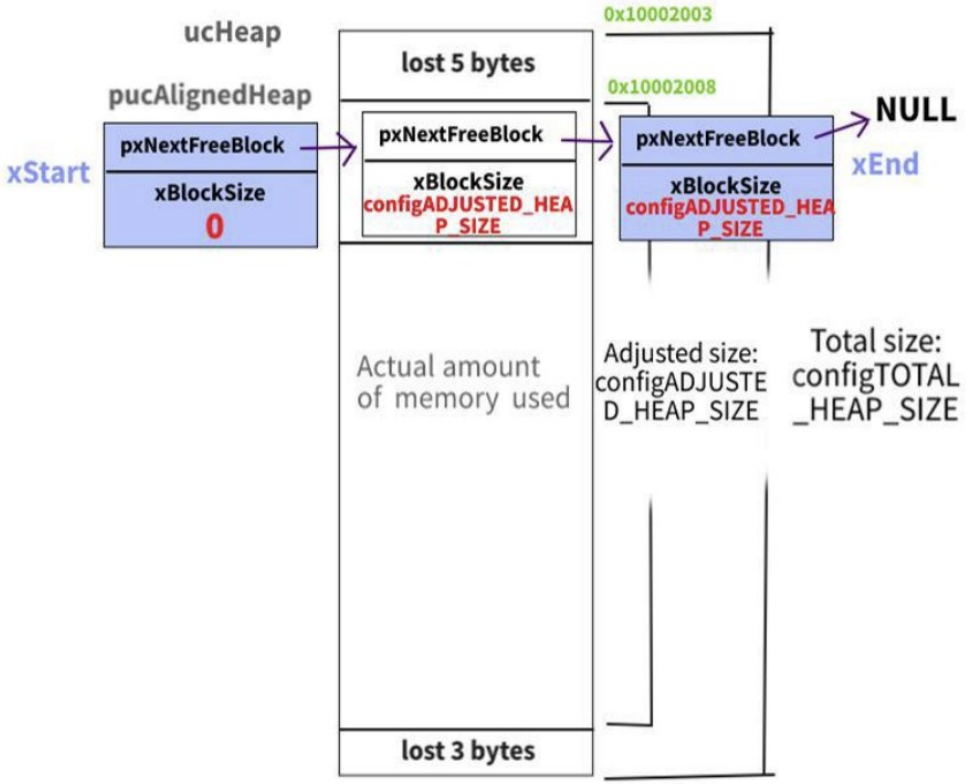
Alignment

- Some bytes may be lost due to the byte alignment of the heap start address.
- `portBYTE_ALIGNMENT`：defined in portmacro.h
- For example, if `portBYTE_ALIGNMENT` is 8



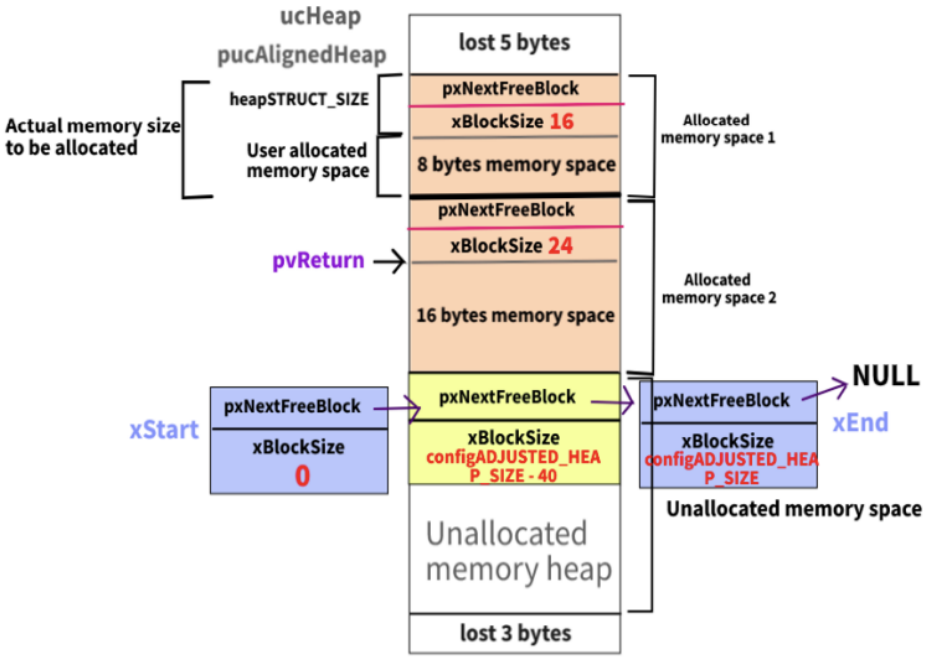
Initialization

- API: `prvHeapInit()`



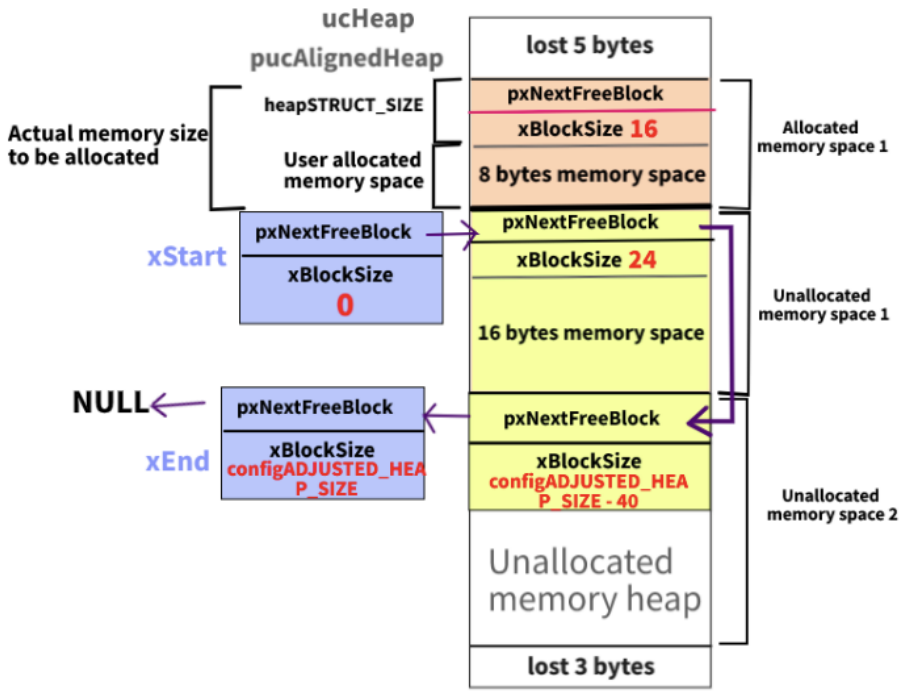
Memory Allocation

- API: `pvPortMallocHeapInit()`
- `xWantedSize`: Manager should indicate memory size which the task needs.



Free Memory Allocation

- Macro: `prvInsertBlockIntoFreeList( pxBlockToInsert)`
- `pxBlockToInsert`: The free block
- Insert free block to the free block list base on its xBlockSize
- No merge



Requirements

- Part 1 (2%) Implement the **`vPrintFreeList`**
- Part 2 (3%) Modify the **`prvInsertBlockIntoFreeList`** macro

Part 1

- Implement your **`vPrintFreeListfunction`** in heap_2.c.
- Task print_task should print the correct free block list to console through UART.

```
void vPrintFreeList(void)
{
    /* TODO: implement this function
     *
     * Reference format
     * > sprintf(data, "StartAddress heapSTRUCT_SIZE xBlockSize EndAddress\n\r");
     * > sprintf(data, "%p         %d           %4d         %p\n\r", ...);
     * > sprintf(data, "configADJUSTED_HEAP_SIZE: %0d xFreeBytesRemaining: %0d\n\r", ...);
     */
}
```

vPrintFreeList

- Print the information of **StartAddress**, **heapSTRUCT_SIZE**, **xBlockSize** and **EndAddress** through UART.
- The free blocks are sorted by their size in ascending order.

Part 2

- Modify **`prvInsertBlockIntoFreeListmacro`** in heap_2.c to implement memory merge.
- After merging the adjacent free blocks, the free blocks should still be sorted in ascending order based on their size.

Part 2

```
/*
 * Insert a block into the list of free blocks - which is ordered by size of
 * the block.  Small blocks at the start of the list and large blocks at the end
 * of the list.
 */
#define prvInsertBlockIntoFreeList( pxBlockToInsert )                          \
{                                                                              \
    BlockLink_t *pxIterator;                                                   \
    size_t xBlockSize;                                                         \
    BlockLink_t *pxPrevious, *pxBlockHolding = pxBlockToInsert;                \
    /* TODO: Merge free blocks                                              */\
    /*                                                                      */\
    /* HINT                                                                 */\
    /* 1. You may need to use the `pxPrevious` pointer to keep trace of the */\
    /*    previous block.                                                   */\
    /* 2. Because it is a macro, use `pxBlockHolding` instead of `pxBlockToInsert`*/\
    /*    below, or you may encounter a problem.                            */\
    /* 3. To Keep the free blocks sorted in block size after merging, you may */\
    /*    want to take adjacent free blocks out.                            */\
    /* 4. Beware the start of the address when merge.                       */\
    /*                                                                      */\
```

prvInsertBlockIntoFreeList

- Contiguous blocks will be merged into bigger block.
- The list of free blocks is ordered by their size in ascending order.

Grading

- Including **Demo** and **Report**
- **Demo (5%)**
  - Implement the **`vPrintFreeList`** (2%)
  - Modify the **`prvInsertBlockIntoFreeList`** macro (3%)
  - The demo can be done in class, or you can upload a demo video
- **Report (3%)**
  - Briefly explain the program logic
  - Write according to the format given on Moodle
  - Please submit a PDF file with the file name as studentID_labNo.pdf for example: P76131234_lab4.pdf
- Both the **Demo** and **Report** must be uploaded on Moodle before 23:59 on 5/15; submissions after the deadline will not be graded