|  |  |  |  |
| --- | --- | --- | --- |
| 112 2 嵌入式作業系統分析與實作 Lab Report | | | |
| Lab Date: | 5/09 | Lab No: | Lab 4 |
| 學號: | P76121110 | 姓名: | 張家菖 |
| Q1 簡述這次lab實驗內容:  FreeRTOS中的heap\_2.c使用linked list來記錄空閒記憶體區塊，並使用best-fit來分配記憶體。 但是，heap\_2.c 不會合併連續的空閒記憶體區塊。  在本實驗中，需要修改 FreeRTOS/portable/MemMang/heap\_2.c 中的 prvInsertBlockIntoFreeList macro，使其在空閒記憶體區塊連續的情況下執行合併。  此外，還必須在 FreeRTOS/portable/MemMang/heap\_2.c 中編寫一個 vPrintFreeList 函數，該函數將迭代列表並列印出每個空閒記憶體區塊的信息，例如起始位址、結束位址和區塊大小。 | | | |
| Q2 簡述這次lab遇到的困難或是完成心得:  uint16\_t heapSTRUCT\_SIZE = ( ( sizeof ( BlockLink\_t ) + ( portBYTE\_ALIGNMENT - 1 ) ) & ~portBYTE\_ALIGNMENT\_MASK )  確保 heapSTRUCT\_SIZE 一定是 8 的倍數，且會是 >= sizeof(BlockLink\_t)  portBYTE\_ALIGNMENT 被定義在 portmacro.h，被定義成 8  portPOINTER\_SIZE\_TYPE 被定義在 FreeRTOS.h，被定義成 uint32\_t  configTOTAL\_HEAP\_SIZE 被定義在 FreeRTOSConfig.h，在這次的lab被助教定義成 ( ( size\_t ) ( 5 \* 1024 ) )  ucHeap 是unsinged char的data type的array，每個element大小都是8 bit (1 byte)，數量是 5 \* 1024，大小是 5 kB  xStart.pxNextFreeBlock = ( void \* ) pucAlignedHeap;  xStart.xBlockSize = ( size\_t ) 0;  xEnd.xBlockSize = configADJUSTED\_HEAP\_SIZE;  xEnd.pxNextFreeBlock = NULL;  xStart  pxNextFreeBlock 被設定為經由對齊後的Heap的起始位址  xBlockSize 被設定為0  xEnd  xBlockSize 被設定成 configADJUSTED\_HEAP\_SIZE，這樣 xEnd 的size就會是全部list中最大的  pxNextFreeBlock 被設定為 NULL  BlockLink\_t \*pxFirstFreeBlock;  pxFirstFreeBlock = ( void \* ) pucAlignedHeap;  pxFirstFreeBlock->xBlockSize = configADJUSTED\_HEAP\_SIZE;  pxFirstFreeBlock->pxNextFreeBlock = &xEnd;  紀錄這個free memory block的 BlockLink\_t 就是放在這個free meory block的起始位址，我們只要去取該BlockLink\_t 的address，就可以知道這個free memory block的起始位址(start address)是多少。每塊free memory block，在起始位址的地方都會用掉 heapSTRUCT\_SIZE 的空間來存放 BlockLink\_t，也就是用8 bytes的空間來存放  #define prvInsertBlockIntoFreeList( pxBlockToInsert ) ……  當有一個新的free memory block要被加入list時，由於list中的free memory block會依據block size由小排到大，所以這裡就是要依據block size找到最適合的位子，把新的block插入: best-fit  portSTACK\_GROWTH (-1), full descending stack: 從記憶體位址大往位址小的地方存東西進去stack，pxTopOfStack 指向的位址是有存東西的，當要push新的東西進去，pxTopOfStack 會先 -1，然後才放東西進去。  在 tasks.c 的 xTaskCreate 的程式碼可以看到，若 portSTACK\_GROWTH 小於0的話，會先執行 pxStack = pvPortMalloc( ( ( ( size\_t ) usStackDepth ) \* sizeof( StackType\_t ) ) )，之後才執行 pxNewTCB = ( TCB\_t \* ) pvPortMalloc( sizeof( TCB\_t ) );，也就是先分配stack的記憶體空間，之後才分配TCB的記憶體空間。  Part1: 從 xStart 開始iterate，直到 xEnd 為止，然後把這中間經過的block的資訊都印出來。block size 128、168、和 208 是 task1、task2、task3 實際被分配到的block size，而另外三個block size為 96 的free memory block就是原本被用來存放這三個task的TCB的空間。  Part2: merge相鄰free memory block   1. Free end-address == New start-address 2. New end-address == Free start-address   先取得要插入的新的block的start address和end address  再來，walk through free memory block list的每個 BlockLink\_t  對於每個block，找出他的start address和end address，檢查是否有相鄰  若相鄰:  a.  把指向NewBlock的pointer改成指向FreeBlock  FreeBlock size += NewBlock size;  Remove FreeBlock from list  b.  把NewBlock size += FreeBlock size;  Remove FreeBlock from list  最後，由於list中的 BlockLink\_t 會依據block size由小排到大，所以可以直接利用原本macro最後的code，找出最適合的位址將此 BlockLink\_t 插入。  Macro, pxBlockToInsert 等於 ( ( BlockLink\_t \* ) pxLink )  pxBlockToInsert = pxCurBlock;   * ( ( BlockLink\_t \* ) pxLink ) = pxCurBlock;   error: lvalue required as left operand of assignment  ( ( BlockLink\_t \* ) pxLink ) 是一個rvalue  解法:  BlockLink\_t \*pxBlockPtr = pxBlockToInsert;  pxBlockPtr = pxCurBlock;  xStart 跟 xEnd 是被宣告為 global variable ，global variable 會被存放在 Data section，不是放在 Stack，所以不會有merge。 | | | |
| Q3 其他(optional):  typedef struct A\_BLOCK\_LINK  {  struct A\_BLOCK\_LINK \*pxNextFreeBlock;  size\_t xBlockSize;  } BlockLink\_t;  pxNextFreeBlock: 指向下一個free memory block的 BlockLink\_t  xBlockSiz:e紀錄目前這個free memory block有多大  start address + xBlockSize = end address  void \*pvPortMalloc( size\_t xWantedSize )  要拿來allocate記憶體用的，當一個task要被create時就會呼叫他。  void vPortFree( void \*pv )  當有一塊記憶體空間被釋放時所呼叫的function。 | | | |
|  | | | |