作業系統導論 LAB 1

系所：通訊三　　學號 :409430030　　姓名:翁佳煌

<實驗工具>

1.FreeRTOS 即時作業系統。

2.JRE 執行 Eclipse 的必備套件。

3.MinGW，Windows 版的 gcc。

4.Eclipse 開發套件。

<實驗過程與方法>

第 6~13 行:include C 語言以及 FreeRTOS 的一些標頭文件。

第 15~19 行:定義題目所要求的延遲，A 七秒喊一次、B 三秒喊一次、C 兩秒喊一次、D 五秒喊一次，在這裡的單位為 ms。

第 21~25 行:是宣告等等用來建立不同優先級的 Task，共建立了五個 Task，分別為 taskD、taskC、taskB、taskA 和 taskT。這些 Task 會在 FreeRTOS 的排程中進行排列，由高優先級的 Task 先執行，低優先級的 Task 則會等待高優先級的 Task 執行完後再執行。

```
1⊖/*
2  * demo.c
3  *
4  */
5 /* Standard includes. */
6 #include <stdio.h>
7 #include <stdbool.h>
8 /* Kernel includes. */
9 #include "FreeRTOS.h"
10 #include "task.h"
11 #include "semphr.h"
12 #include "list.h"
13 #include "queue.h"
14
15 #define A_DELAY_MS 7000
16 #define B_DELAY_MS 3000
17 #define C_DELAY_MS 2000
18 #define D_DELAY_MS 5000
19 #define TIME_DELAY_MS 1000
20
21 static void taskA( void *p );
22 static void taskB( void *p );
23 static void taskC( void *p );
24 static void taskD( void *p );
25 static void taskT( void *p );
26
```

▲圖 1.1

接下來第 27~67 行:在這裡分別定義了五個 Task 的函式實作，分別是 taskA、taskB、taskC、taskD 和 taskT，在這裡舉第 27 行~35 行的 taskD 程式碼做解釋，其餘 taskA 、B、 C、 T 皆雷同。第 29 行呼叫 xTaskCreate()， 是 FreeRTOS 中建立 Task 的 API 函數，其中需要傳入下列參數：

1. Task function：Task 的實際執行內容，為一個函數指標。
2. Task name：Task 名稱，可以供 Debug 使用。
3. Stack size：Task 所需要的 Stack 大小，以 byte 為單位。
4. Parameter：Task 執行所需要的參數。
5. Priority：Task 的優先級，數值越高代表優先級越高。
6. Handle：Task 的 handle，用來做記憶體管理和控制 Task 的執行。在本程式碼中並未使用到。

最後，在第 65 行呼叫 vTaskStartScheduler()來啟動排程器決定讓哪個 task 開始執行。

<補充::參考資料 1>

當 vTaskStartScheduler() 被呼叫時，會先建立一個 idle task，這個 task 是為了確保 CPU 在任一時間至少有一個 task 可以執行（取代直接切換回 kernel task）而在 vTaskStartScheduler() 被呼叫時自動建立的 user task，idle task 的 priority 為 0 (lowest)，目的是為了確保當有其他 user task 進入 ready list 時可以馬上被執行。

```
27  void demo( void )
28  {
29      xTaskCreate( taskD,                 /* The function that implements the task. */
30                   "D",                       /* The text name assigned to the task - for debug only as it is not used by the kernel. */
31                   configMINIMAL_STACK_SIZE,  /* The size of the stack to allocate to the task. */
32                   NULL,                      /* The parameter passed to the task - just to check the functionality. */
33                   1,                         /* The priority assigned to the task. */
34                   NULL );
35
36
37      xTaskCreate( taskC,                 /* The function that implements the task. */
38                   "C",                       /* The text name assigned to the task - for debug only as it is not used by the kernel. */
39                   configMINIMAL_STACK_SIZE,  /* The size of the stack to allocate to the task. */
40                   NULL,                      /* The parameter passed to the task - just to check the functionality. */
41                   2,                         /* The priority assigned to the task. */
42                   NULL );
43
44      xTaskCreate( taskB,                 /* The function that implements the task. */
45                   "B",                       /* The text name assigned to the task - for debug only as it is not used by the kernel. */
46                   configMINIMAL_STACK_SIZE,  /* The size of the stack to allocate to the task. */
47                   NULL,                      /* The parameter passed to the task - just to check the functionality. */
48                   3,                         /* The priority assigned to the task. */
49                   NULL );
50
51      xTaskCreate( taskA,                 /* The function that implements the task. */
52                   "A",                       /* The text name assigned to the task - for debug only as it is not used by the kernel. */
53                   configMINIMAL_STACK_SIZE,  /* The size of the stack to allocate to the task. */
54                   NULL,                      /* The parameter passed to the task - just to check the functionality. */
55                   4,                         /* The priority assigned to the task. */
56                   NULL );
57
58      xTaskCreate( taskT,                 /* The function that implements the task. */
59                   "Time",                        /* The text name assigned to the task - for debug only as it is not used by the kernel. */
60                   configMINIMAL_STACK_SIZE,  /* The size of the stack to allocate to the task. */
61                   NULL,                      /* The parameter passed to the task - just to check the functionality. */
62                   5,                         /* The priority assigned to the task. */
63                   NULL );
64
65      vTaskStartScheduler();   //讓task 進入 FreeRTOS scheduling
66
67      while(1);
68  }
69
```

▲圖 1.2

再來第 71~121 行:這大段程式碼也都是大同小異，我們拿第 71~79 行的 taskT 來舉例，其餘 taskA 、B、 C、 D 以此類推，因為 taskT 是用來印出每一秒的秒數，因此在第 74 行將他印出，第 75 行使用 vTaskDelay()來做延遲。因此，接下來的 taskA 、B、 C、 D 的差異也只是印出的文字以及 vTaskDelay 的時間不同而已。

```
71 static void taskT( void *p )
72 {
73     while(1){
74         printf("%d second.\n", xTaskGetTickCount()/1000);
75         vTaskDelay(TIME_DELAY_MS);   //delay for 1000ms,
76         fflush( stdout );  //clean buffer
77     }
78 }
79
```

```
81 static void taskA( void *p )
82 {
83     const char *name = "A";
84     while(1){
85
86         printf("%s!!\n", name);
87         vTaskDelay(A_DELAY_MS);   //delay for 7000ms, but in windows freeRTOS simulator, it's more than 100ms.
88         fflush( stdout );  //clean buffer
89     }
90 }
```

```
92 static void taskB( void *p )
93 {
94     const char *name = "B";
95     while(1){
96
97         printf("%s!!\n", name);
98         vTaskDelay(B_DELAY_MS);   //delay for 3000ms, but in windows freeRTOS simulator, it's more than 100ms.
99         fflush( stdout );  //clean buffer
100    }
101 }
```

```
102 static void taskC( void *p )
103 {
104     const char *name = "C";
105     while(1){
106
107         printf("%s!!\n", name);
108         vTaskDelay(C_DELAY_MS);   //delay for 2000ms, but in windows freeRTOS simulator, it's more than 100ms.
109         fflush( stdout );  //clean buffer
110     }
111 }
112 static void taskD( void *p )
113 {
114     const char *name = "D";
115     while(1){
116
117         printf("%s!!\n", name);
118         vTaskDelay(D_DELAY_MS);   //delay for 5000ms, but in windows freeRTOS simulator, it's more than 100ms.
119         fflush( stdout );  //clean buffer
120     }
121 }
```

<心得與收穫>

這次是 OS 這堂課的第一個 LAB，雖然老師說明這堂課的 LAB 不像微處理機是強迫你必須去做的，但我對於 OS 的知識實在覺得非常有趣，於是還是決定要試著做做看。有事先看了助教放在 PPT 給我們參考的 FreeRTOS 的資源，研究完後會發現這次 LAB 不是那麼困難(又或許是因為被微處理機摧殘過)，雖然實驗順利完成，但我發現網站上還有好多看不懂而且長相詭異的函數，希望藉由這次 LAB 入門後，自己能更精進 OS 的架構以及概念。

<參考資料>

1. http://wiki.csie.ncku.edu.tw/embedded/freertos#FreeRTOS%20%E6%9E%B6%E6%A7%8B
2. http://www.aosabook.org/en/freertos.html