

**<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>**

下圖 1，首先 1~30 行，都是一些參數宣告，特別注意的是，Mutex 和 Semaphore 的宣告方式和用法。Mutex 只能被一個 Task 同時獲得，而 Semaphore 則可以被多個 Task 同時獲得。

```
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 static void taskA();
16 static void taskB();
17 static void taskC();
18 static void taskD();
19 static void counter();
20 static void cleaner();
21
22 //Mutex 只能被一個 Task 同時獲得，而 Semaphore 則可以被多個 Task 同時獲得
23
24 SemaphoreHandle_t Toilet = NULL;
25
26 int usingToilet = 0; //
27 int waitToilet = 0;  //判斷誰是first person
28
29 SemaphoreHandle_t cleanerSemaphore = NULL;
30 //SemaphoreHandle_t allTaskDone = NULL;
31
```

▲圖 1

下圖 2，第 32~75 行，首先值得注意的是，第 33 行的 cleanerSemaphore，它是一個 mutex semaphore，也就是說只能有一個任務或線程能夠同時擁有它，其他任務或線程需要等待之前持有者釋放後才能獲得它。

而第 34 行的 Toilet，它是一個 counting semaphore，最初的計數器值是 2，也就是說同時只能有 2 個任務或線程能夠獲得它，當一個線程或任務成功獲取了 semaphore，semaphore 的計數器會減去 1 為建立任務。

接下來的幾行為題目要求，分別建立計算時間、4 位學生和 cleaner 的 task，其中，為了此實驗的目的，必須把 4 位學生和 1 為 cleaner 的權重設定為相同，在此我皆設定為 1。

```
32=void demo( void ){
33     cleanerSemaphore = xSemaphoreCreateMutex();
34     Toilet = xSemaphoreCreateCounting(2, 2); // (MaxCount, InitialCount) 有 2 間隔間，當一個線程或任務成功獲取了 semaphore，semaphore 的計數器會減去 1
35     xTaskCreate( counter, /* The function that implements the task. */
36                 "counter", /* The text name assigned to the task - for debug only as it is not used by the kernel. */
37                 configMINIMAL_STACK_SIZE, /* The size of the stack to allocate to the task. */
38                 NULL, /* The parameter passed to the task - just to check the functionality. */
39                 2,
40                 NULL );
41     xTaskCreate( taskA, /* The function that implements the task. */
42                 "A", /* The text name assigned to the task - for debug only as it is not used by the kernel. */
43                 configMINIMAL_STACK_SIZE, /* The size of the stack to allocate to the task. */
44                 NULL, /* The parameter passed to the task - just to check the functionality. */
45                 1, /* The priority assigned to the task. */
46                 NULL );
47     xTaskCreate( taskB, /* The function that implements the task. */
48                 "B", /* The text name assigned to the task - for debug only as it is not used by the kernel. */
49                 configMINIMAL_STACK_SIZE, /* The size of the stack to allocate to the task. */
50                 NULL, /* The parameter passed to the task - just to check the functionality. */
51                 1, /* The priority assigned to the task. */
52                 NULL );
53     xTaskCreate( taskC, /* The function that implements the task. */
54                 "C", /* The text name assigned to the task - for debug only as it is not used by the kernel. */
55                 configMINIMAL_STACK_SIZE, /* The size of the stack to allocate to the task. */
56                 NULL, /* The parameter passed to the task - just to check the functionality. */
57                 1, /* The priority assigned to the task. */
58                 NULL );
59     xTaskCreate( taskD, /* The function that implements the task. */
60                 "D", /* The text name assigned to the task - for debug only as it is not used by the kernel. */
61                 configMINIMAL_STACK_SIZE, /* The size of the stack to allocate to the task. */
62                 NULL, /* The parameter passed to the task - just to check the functionality. */
63                 1, /* The priority assigned to the task. */
64                 NULL );
65
66     xTaskCreate( Cleaner, /* The function that implements the task. */
67                 "Cleaner", /* The text name assigned to the task - for debug only as it is not used by the kernel. */
68                 configMINIMAL_STACK_SIZE, /* The size of the stack to allocate to the task. */
69                 NULL, /* The parameter passed to the task - just to check the functionality. */
70                 1, /* The priority assigned to the task. */
71                 NULL );
72
73     vTaskStartScheduler();
74     while(1);
75 }
```

▲圖 1.2

接下來 77~104 為 cleanToilet 函數，首先，程式會印出 "Cleaner is waiting" 這句話，表示清潔人員正在等待可以開始清潔廁所。

接下來，使用了一個 if 判斷式，判斷是否有使用者正在使用廁所。若 waitToilet 為 0，代表所有的廁所都關閉了，沒有人在使用，此時就可以開始進行清潔。

在清潔的同時，為了避免其他人進入廁所，使用了 xSemaphoreTake 函式來取得 cleanerSemaphore 這個 mutex。Mutex 是一個用於控制對共享資源存取的同步機制，當一個任務獲取了 mutex，其他任務就不能獲取，直到該 mutex 釋放。因此，當清潔人員正在清潔廁所時，其他使用者就不能進入廁所，以避免互相干擾。

接著，使用 vTaskDelay 函式延遲 300 個 tick，模擬清潔廁所的時間。當清潔完成後，使用 xSemaphoreGive 函式釋放 mutex，讓其他人可以進入廁所。

最後，使用 break 關鍵字跳出迴圈，回到 Cleaner() 函式。如果 waitToilet 不為 0，代表仍有人在使用廁所，此時就會等待 100 個 tick，再進行下一輪的檢查。

```
77 static void cleanToilet(){
78     printf("Cleaner is waiting\n");
79     while(1){
80         if( xSemaphoreTake( allTaskDone, (TickType_t) 0) == pdTRUE){
81             xSemaphoreGive( allTaskDone );
82         }
83         else{
84             continue;
85         }
86         if( waitToilet == 0){ // usingToilet
87             printf("All Toilets are closed for cleaning!!\n");
88
89             //在清理的同時 其他人都不能進入廁所
90             if( xSemaphoreTake( cleanerSemaphore, (TickType_t) 0) == pdTRUE){
91                 vTaskDelay(300);
92                 printf("Cleaning is completed\n");
93                 xSemaphoreGive( cleanerSemaphore );
94             }
95             else{
96                 printf("error\n");
97             }
98             break;//回到Cleaner()
99         }
100     }
101     else{
102         vTaskDelay(100);
103     }
104 }
```

▲圖 3

接下來 106~148 為 useToilet 函數，它有三個參數，分別是使用廁所的時間 (duration)、離開廁所後的休息時間 (period)、以及該學生的名稱 (task)。

首先，函數會檢查這是否是第一個進入廁所的學生，如果是，就會印出一條訊息。然後，函數會增加等待使用廁所的人數 (waitToilet)，代表有一個學生正在等待使用廁所。接著，函數會檢查清潔員是否在清理廁所，如果是，函數就會卡住等待，直到獲取到 cleanerSemaphore，代表清潔員已經完成了清潔工作。

接下來，函數會等待進入廁所，使用 xSemaphoreTake 函數來獲取 Toilet 信號量。如果成功獲取到信號量，代表廁所可用，就會增加使用廁所的人數 (usingToilet)，然後印出使用廁所的訊息。之後，該學生會延遲 duration 毫秒，模擬使用廁所的時間。接著，該學生會印出離開廁所的訊息，並休息 period 毫秒。最後，函數會減少等待使用廁所的人數 (waitToilet)，以及減少使用廁所的人數 (usingToilet)。如果這個學生是最後一個離開廁所的人，函數就會印出一條訊息，表示可以釋放 Toilet 信號量了。最後，函數會釋放 Toilet 信號量，以便其他學生可以進入使用。

最後，函數會延遲 period 毫秒，以便其他任務可以運行，並清空 stdout 緩衝區。

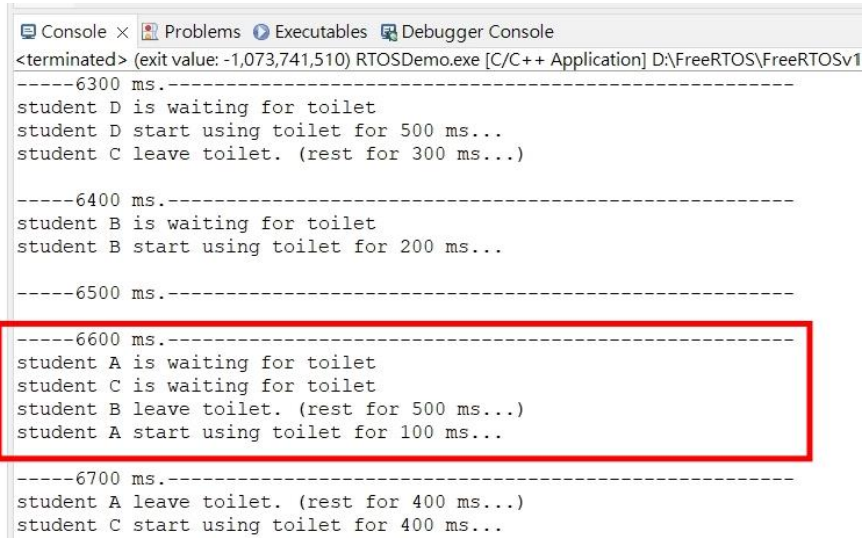
```
106 static void useToilet(int duration, int period, char task){
107
108     //判斷第一個進廁所的人
109     if(waitToilet == 0){
110         printf("student %c is the first one who want get in to toilet, check the toilet-semaphore\n", task);
111     }
112     waitToilet++;
113
114     //Cleaner沒在清理廁所 如果在清理就卡住等 直到拿到cleanerSemaphore
115     if( xSemaphoreTake( cleanerSemaphore, (TickType_t) portMAX_DELAY) == pdTRUE){
116         printf("student %c is waiting for toilet\n", task);
117         xSemaphoreGive( cleanerSemaphore );
118     }
119
120     //等進廁所
121     if( xSemaphoreTake( Toilet, portMAX_DELAY) == pdTRUE){
122         // if( xSemaphoreTake( cleanerSemaphore, (TickType_t) 0) == pdTRUE){
123         //     xSemaphoreGive( cleanerSemaphore );
124         // }
125         usingToilet++;
126
127         printf("student %c start using toilet for %d ms...\n", task, duration);
128
129         vTaskDelay(duration);
130
131         printf("student %c leave toilet. (rest for %d ms...)\n", task, period);
132         waitToilet--;
133
134         usingToilet--;
135         if(usingToilet == 0){
136             printf("student %c is the last one get out the toilet, release the toilet-semaphore\n", task);
137         }
138
139         xSemaphoreGive( Toilet );//釋放廁所
140     }
141     else{
142         printf("task %c error\n",task);
143     }
144
145     vTaskDelay(period); //delay for 100ms, but in windows freeRTOS simulator, it's more than 100ms.
146     fflush( stdout ); //clean buffer
147 }
148
```

▲圖 4

## <問題與討論>

### 1. 在第 6600ms 時，誰在使用廁所？

下圖 5 為執行結果，在第 6600ms 的時候 A 在使用廁所。



```
Console x Problems Executables Debugger Console
<terminated> (exit value: -1,073,741,510) RTOSDemo.exe [C/C++ Application] D:\FreeRTOS\FreeRTOSv1
-----6300 ms.-----
student D is waiting for toilet
student D start using toilet for 500 ms...
student C leave toilet. (rest for 300 ms...)

-----6400 ms.-----
student B is waiting for toilet
student B start using toilet for 200 ms...

-----6500 ms.-----
-----6600 ms.-----
student A is waiting for toilet
student C is waiting for toilet
student B leave toilet. (rest for 500 ms...)
student A start using toilet for 100 ms...

-----6700 ms.-----
student A leave toilet. (rest for 400 ms...)
student C start using toilet for 400 ms...
```

▲圖 5

## 2. 當 Student D 的 period 改為 200ms 時，會發生什麼問題，為什麼？

下圖 6 為把 Student D 的 period 改為 200ms 的執行結果，原本在第 1200ms 應該會全部人都離開使 cleaner 可以開始清理廁所，但這裡會發現如果把 D 改成 200ms，會無法達成所有學生都不想上廁所的條件，導致 cleaner 無法清理廁所。

```

-----900 ms.-----
student B leave toilet. (rest for 500 ms...)
student D is waiting for toilet
student D start using toilet for 500 ms...

-----1000 ms.-----
student A is waiting for toilet

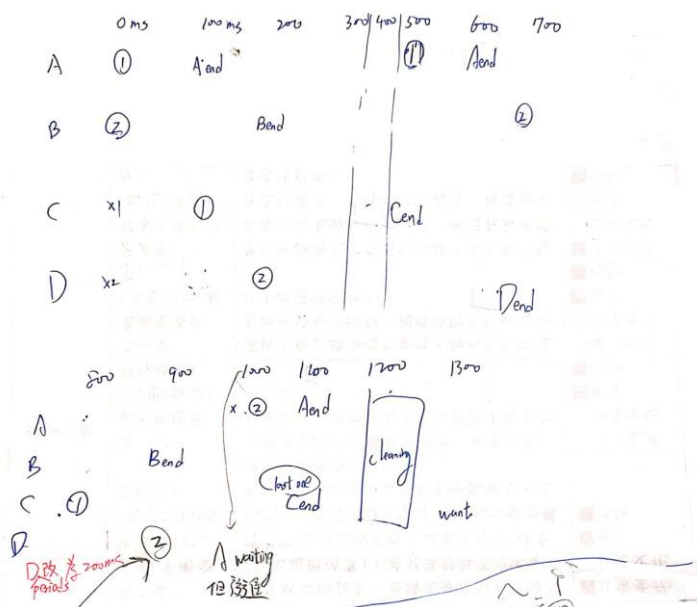
-----1100 ms.-----

-----1200 ms.-----
student C leave toilet. (rest for 300 ms...)
student A start using toilet for 100 ms...

-----1300 ms.-----
student A leave toilet. (rest for 400 ms...)

-----1400 ms.-----
student B is waiting for toilet
student B start using toilet for 200 ms...
student D leave toilet. (rest for 200 ms...)
    
```

▲圖 6





### <心得與收穫>

這次是 OS 這堂課的第 2 個 LAB，主要學到了如何使用 FreeRTOS 的 mutex 和 counting semaphore 來實現多線程的同步和互斥。mutex semaphore 可以保證在線程中某些關鍵代碼區塊只能被一個線程訪問，從而防止競爭條件和數據不一致的問題。而 counting semaphore 則用於限制同一時間能夠訪問某些共享資源的線程數量，從而控制訪問頻率和訪問順序。在編寫多線程的 code 時，必須注意到競爭條件和數據不一致的問題，適當使用 mutex 和 counting semaphore 可以避免這些問題的發生。

這次的實驗花了我比較多時間，mutex 和 semaphore 本身的概念並不難，往往是卡在邏輯上面的實現，花了很久搞清楚此時到底誰應該上廁所或是清理，我認為畫圖還是最快速可以搞懂的方法，總之，藉由這次 LAB 提高了我對多線程的概念和實作能力。

### <參考資料>

<http://wiki.csie.ncku.edu.tw/embedded/freertos>

[https://jasonblog.github.io/note/linux\\_system/mutex\\_yu\\_semaphore\\_zui\\_da\\_de\\_cha\\_yi\\_shi.html](https://jasonblog.github.io/note/linux_system/mutex_yu_semaphore_zui_da_de_cha_yi_shi.html)

<https://ithelp.ithome.com.tw/articles/10279050>

<https://hackmd.io/@RinHizakura/rJhEpdYNw>