作業系統導論 LAB 3

系所：通訊三 學號 :409430030 姓名:翁佳煌

**<實驗工具>**

1. Keil MDK。

2. Nu-Link\_Keil\_Driver。

3. NUC100\_Series\_BSP\_CMSIS。

4. FreeRTOS\_project

5.NUC140 開發板一塊。

6.USB cable

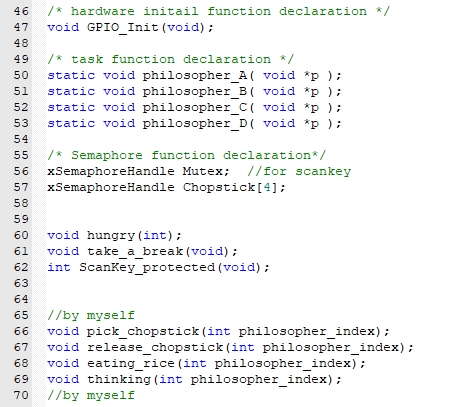
**<實驗過程與方法>**

**Part A:**

首先看到下圖1，50~53行分別先宣告任務函數，分別哲學家A~D。

56~57行的xSemaphoreHandle Mutex這是一個互斥信號量的聲明，用於保護對按鍵的訪問。xSemaphoreHandle Chopstick[4]這是一個計數信號量的聲明，表示四根筷子的可用數量。

再來60~69行為自定義的函示宣告，下方將會詳細介紹函式的功能。

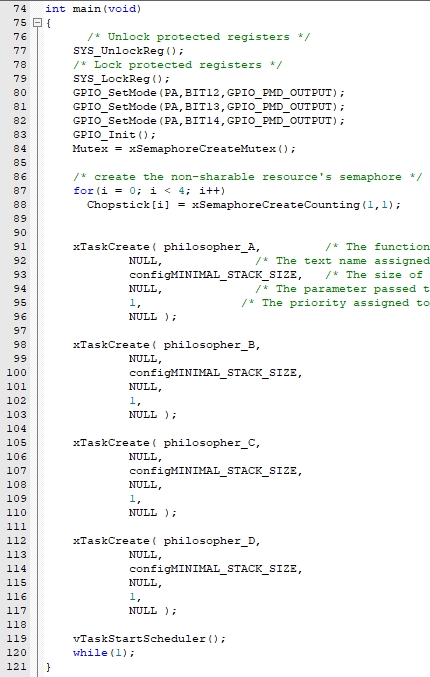


▲圖1

下圖2，77~83行，主要是在設定GPIO的模式，將對應的腳位為輸出，並呼叫 GPIO\_Init() 函數，進行進一步的GPIO初始化。

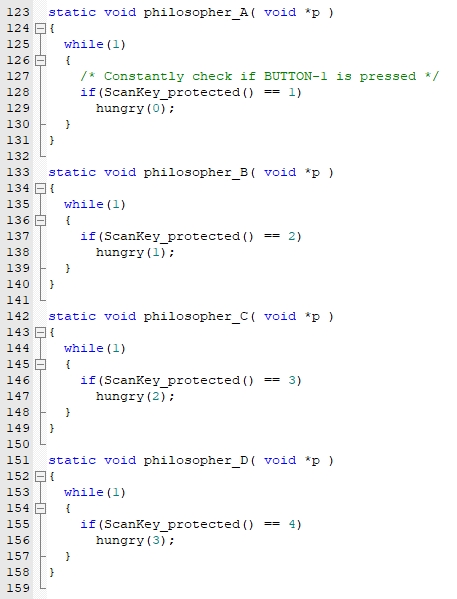
84行，Mutex = xSemaphoreCreateMutex()負責創建一個互斥信號量，並將其分配給 Mutex 變量。

91~120行，則是創建任務，並啟動FreeRTOS任務調度器。



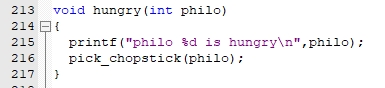
▲圖2

下圖3，當按下對應的Scankey時候，我們舉哲學家A的123~132行為例子，也就是當按下NUC140鍵盤的1位置時，代表哲學家A想要吃飯了，也就會進到hungry這個函式，其他哲學家B~D以此類推。



▲圖3

當按下鍵盤後，便會進入下圖4的hungry函式，代表目前該哲學家肚子餓了，也就會進入第216行的pick\_chopstick函數。



▲圖4

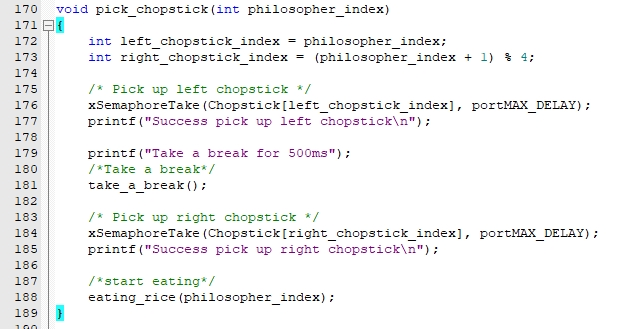
下圖5，首先第172~173行，會根據傳入的 philosopher\_index 計算出左手筷子和右手筷子的索引。

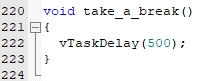
176行使用 xSemaphoreTake() 函數等待左手筷子的可用性，並將其取走，如果左手筷子不可用，這個函數會一直等待，直到筷子可用。

在成功取走左手筷子後，會執行take\_a\_break()進行500毫秒的休息。

休息完後到184行，使用 xSemaphoreTake() 函數等待右手筷子的可用性，並將其取走，如果右手筷子不可用，這個函數會一直等待，直到筷子可用。

在成功取走右手筷子後，會進入188行的 eating\_rice() 函數，進行進食。該函數負責控制哲學家進食一段時間和LED開關。



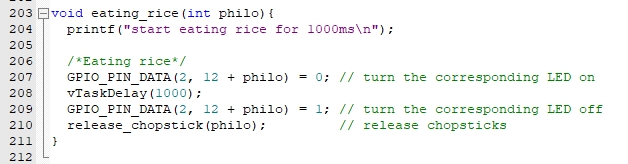


▲圖5

下圖6，進到eating\_rice後，表示哲學家開始進食，因此通過設置 GPIO 達到控制紅色 LED 的目的，將相應的 LED 打開，也就是會根據傳入的 philo 參數，計算出對應的 GPIO 編號，並將其設置為低電平，從而打開對應的 LED。

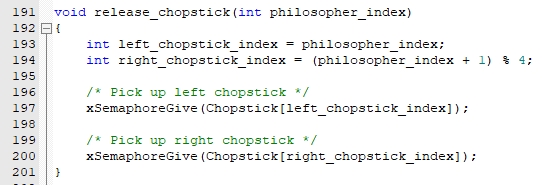
並在208行執行 vTaskDelay(1000)函數，使該任務延遲1000毫秒。這個延遲的時間模擬了哲學家進食的時間。

通過再次設置 GPIO 達到控制紅色 LED 的目的，將相應的 LED 關閉，根據傳入的 philo 參數，計算出對應的 GPIO 編號，並將其設置為高電平，從而關閉對應的 LED。

最後在第210行調用 release\_chopstick() 函數，釋放哲學家手中的筷子，這將使其他哲學家能夠繼續取用這些筷子。

▲圖6

下圖7，191~201為釋放筷子的函式，首先透過193~194行計算出左手和右手筷子的索引，接下來，通過 197和200行的xSemaphoreGive 函數釋放左手和右手的筷子。



▲圖7

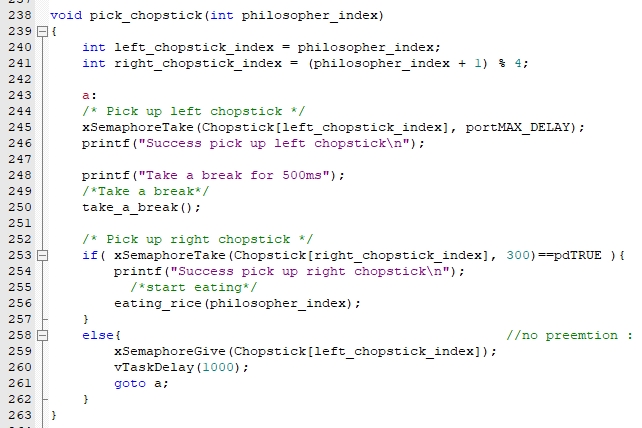
**Part B:**

ParB的要求為請在已完成的Part A程式上實作，讓其中一個deadlock條件永遠不會發生，以實現Deadlock Prevention。

因此我針對下圖8，pick\_chopstick的函數，去拿掉no preemtion這個dedalock的條件。

這函式被修改以處理一個特殊情況。當一位哲學家試圖拿起右手的筷子時，如果該筷子已被其他哲學家佔用，並且等待的時間超過 300 個 tick，則將放下已拿起的左手筷子並進行等待後重新嘗試。

這樣就可成功避免deadlock的問題。



▲圖8

**<問題與討論>**

1. **在Part A的程式，何種情況會發生Deadlock?**

我發現可能會發生死結（Deadlock）的情況是在哲學家就餐的過程中。以下是可能導致死結的情況：

1.位哲學家都同時拿起自己左邊的筷子。假設所有哲學家都同時執行到 xSemaphoreTake(Chopstick[left\_chopstick\_index], portMAX\_DELAY) 這行程式碼，並且所有筷子都被其他哲學家佔用。在這種情況下，每位哲學家都等待左邊的筷子釋放，但其他哲學家都在等待其他筷子釋放，導致死結。

2.每位哲學家都同時拿起自己右邊的筷子。假設所有哲學家都同時執行到 xSemaphoreTake(Chopstick[right\_chopstick\_index], portMAX\_DELAY) 這行程式碼，並且所有筷子都被其他哲學家佔用。在這種情況下，每位哲學家都等待右邊的筷子釋放，但其他哲學家都在等待其他筷子釋放，導致死結。

**2. Deadlock的發生與按下keypad的順序有關嗎?**

是的，Deadlock的發生與按下keypad的順序有關。在這段程式碼中，每位哲學家的任務（task）都在一個無窮迴圈中等待按下特定的keypad按鈕才會進行相應的操作。

如果按下按鈕的順序不當，例如，同時或快速地按下按鍵1~4，就可能導致Deadlock的發生。Deadlock通常發生在多個執行緒或任務（task）需要同時佔用多個共享資源，而這些資源彼此相互等待，導致所有執行緒或任務都無法繼續執行的情況。

1. **在part B你以哪個方法實作Deadlock Prevention? 為什麼這樣就能避免Deadlock?**

我在pick\_chopstick這個函數中，拿掉no preemtion這個dedalock的條件。

這函式被修改為當一位哲學家試圖拿起右手的筷子時，如果該筷子已被其他哲學家佔用，並且等待的時間超過一定時間，則放下目前拿起的左手筷子並進行等待後重新嘗試。

**<心得與收穫>**

這是最後一次的LAB，它讓我更加了解Deadlock，Deadlock是多個進程或執行緒之間的一種阻塞狀態，每個進程都在等待其他進程所擁有的資源，從而無法繼續執行。在這個問題中，哲學家需要兩支筷子才能就餐，但如果每個哲學家都先拿起左邊的筷子，那麼就會發生Deadlock。

以及學習解決Deadlock的策略，要解決這問題，我們可以採用多種策略，如避免、預防、檢測和恢復。在哲學家就餐問題中，我是使用「no preemption」策略，即一旦哲學家無法獲得兩支筷子，就放下已經拿起的筷子，等待一段時間後重新嘗試。這樣可以避免一位哲學家佔用兩支筷子，使其他哲學家無法繼續執行。

總而言之，通過解決哲學家吃飯的問題並避免Deadlock，我深入了解了作業系統的概念和解決策略，並學習了在程式碼設計中應用同步機制和適當的資源管理的重要性。

**<參考資料>**

<https://ithelp.ithome.com.tw/articles/10206625>

<https://hackmd.io/@ExcitedMail/Hy6Ot7s7Y>

[**http://wiki.csie.ncku.edu.tw/embedded/freertos**](http://wiki.csie.ncku.edu.tw/embedded/freertos)

<https://www.keil.com/demo/eval/arm.htm>