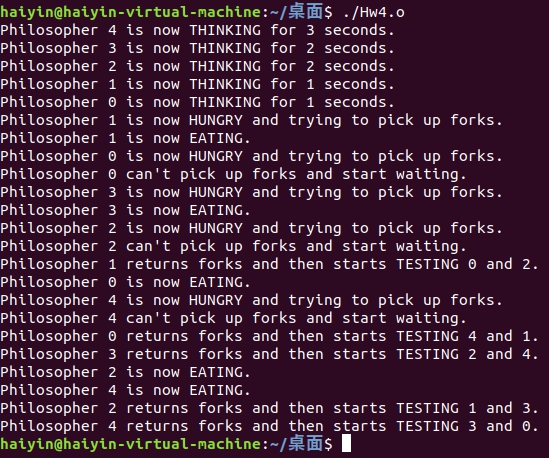
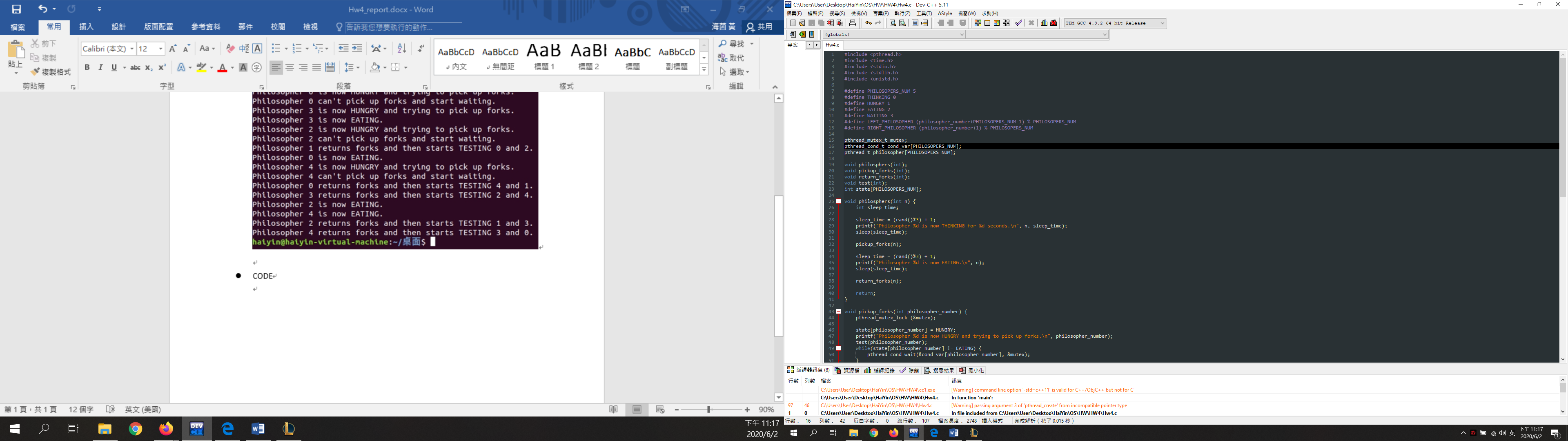
**HW4 Dining Philosophers Problem**

105072123 黃海茵

* OUTPUT



* CODE

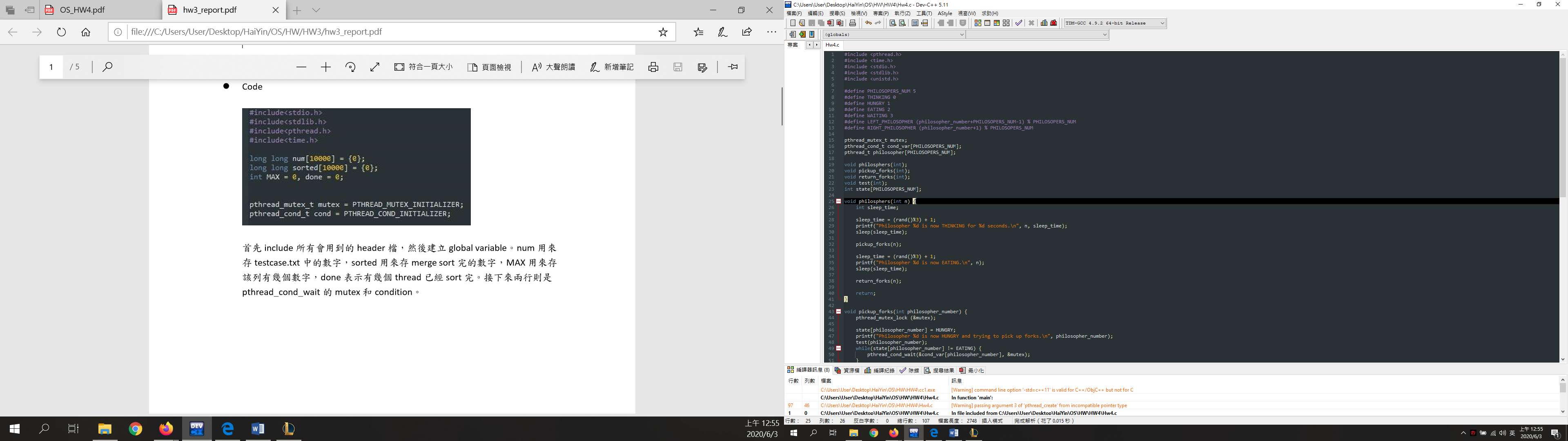


首先include所有會用到的header檔，然後define會用到的常數。

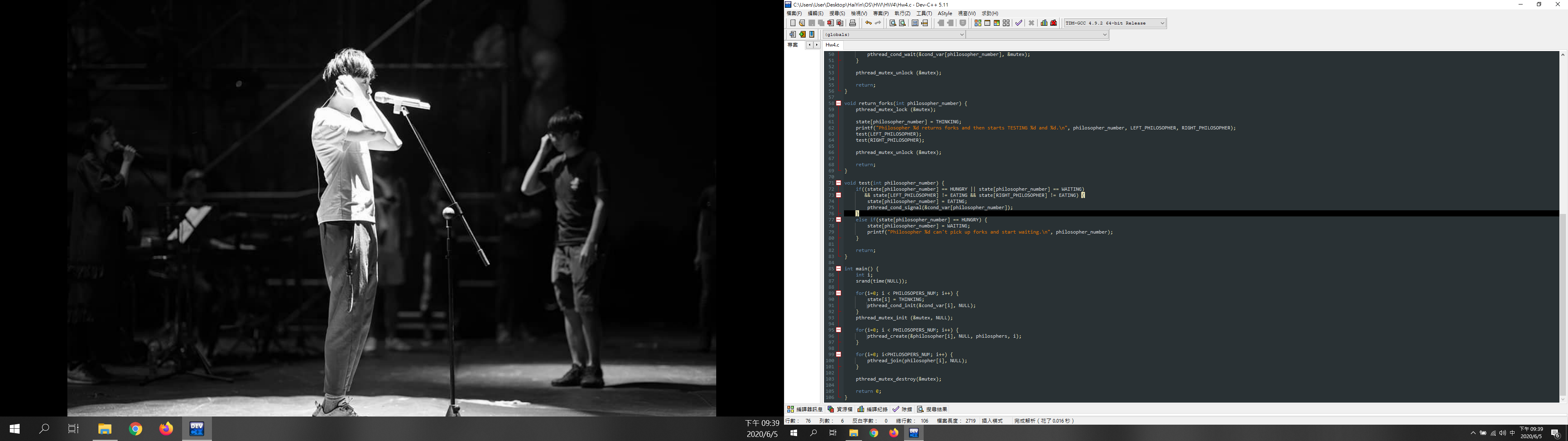
PHILOSOPERS\_NUM代表philosopher的人數

THINKING, HUNGRY, EATING, WAITING代表四個不同的state

LEFT\_ PHILOSOPERSR跟RIGHT\_ PHILOSOPERS代表左邊和右邊的philosopher



宣告會用到的變數及function。

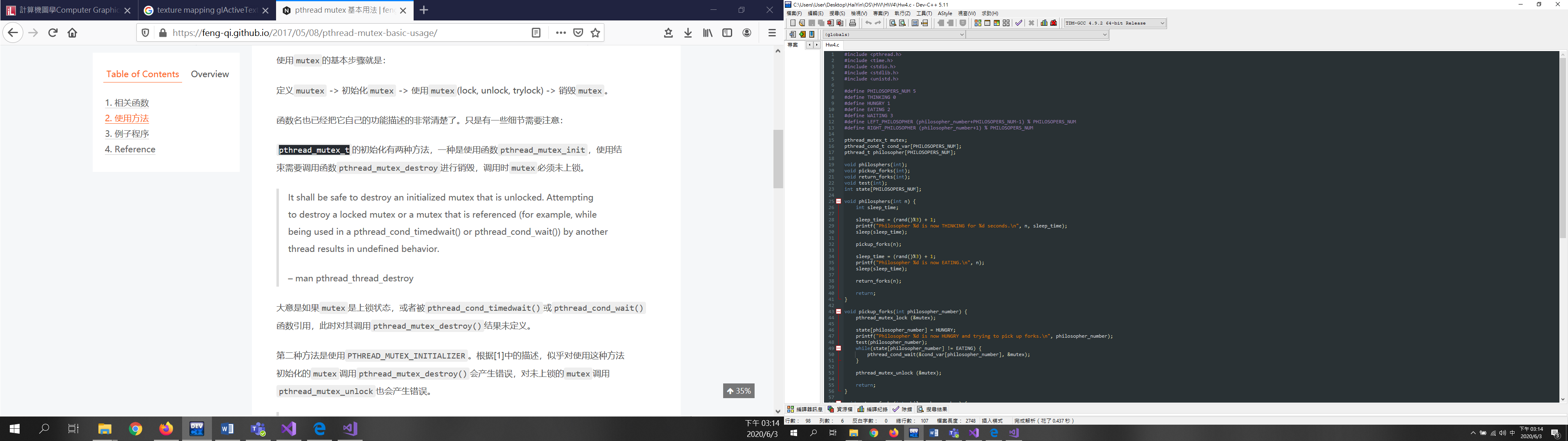


用srand()函數改變一開始的亂數值，srand()需要傳入一個參數，以產生一個新的亂數數列。這個參數用目前的時間傳入，也就是使用time() 函數。然後for loop依序把每個philosopher的state設為THINKING，並用pthread\_cond\_init和pthread\_mutex\_init來做初始化。

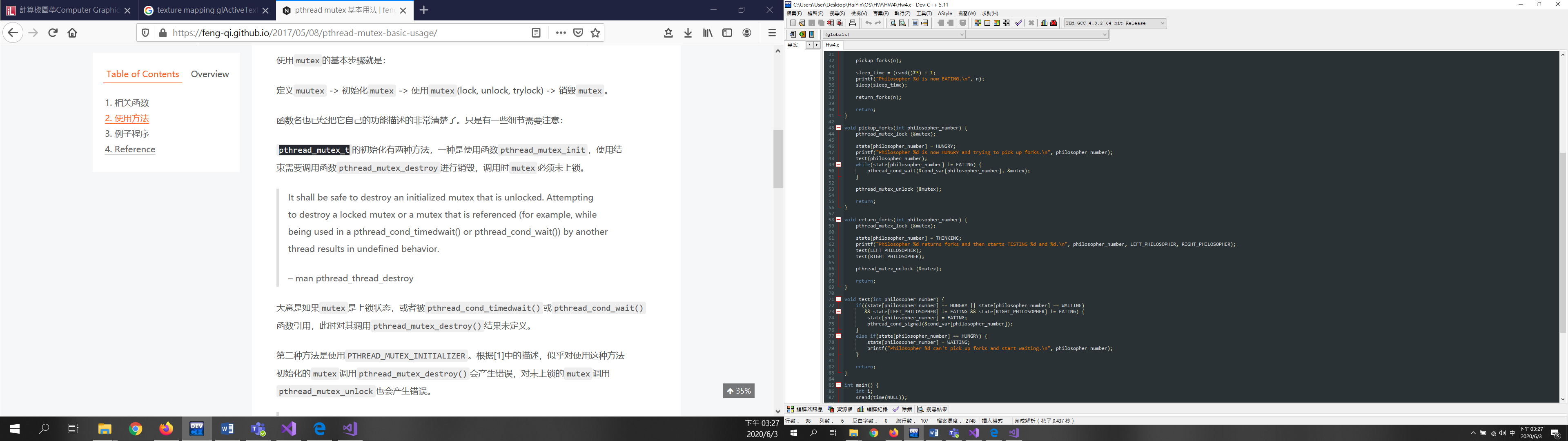
再用pthread\_create來依序建立thread，第一個參數為指向thread識別符的pointer，第二個參數用來設定thread屬性，第三個參數是thread執行function的起始位址，最後一個參數是執行fucntion的參數。

接下來pthread\_join用來等待一個thread的結束。第一個參數為被等待的thread識別符號，第二個引數為一個使用者定義的指標，它可以用來儲存被等待thread的return值。這個function是一個thread阻塞的function，呼叫它的function將一直等待到被等待的thread結束為止，當function return時，被等待thread的資源被收回。

最後使用pthread\_mutex\_destroy來destroy pthread\_mutex\_t。

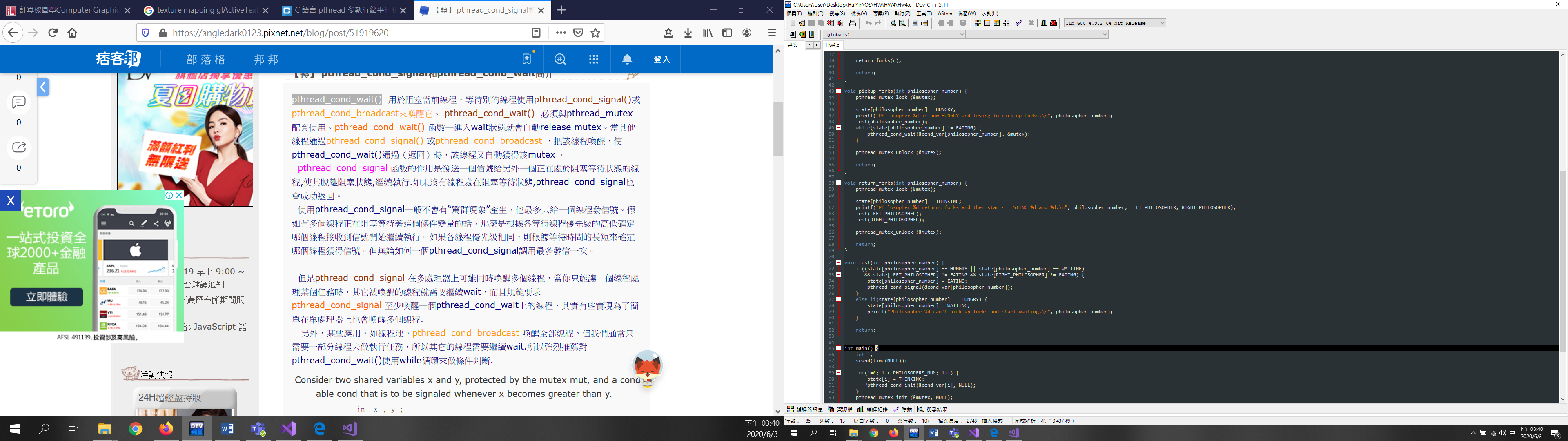


根據題目要求，把sleep\_time設為1~3的random number，模擬thinking的時間。thinking結束後，就可以pickup\_forks。然後根據題目要求，再把sleep\_time設為1~3的random number，模擬eating的時間，eating結束後，就可以return\_forks。



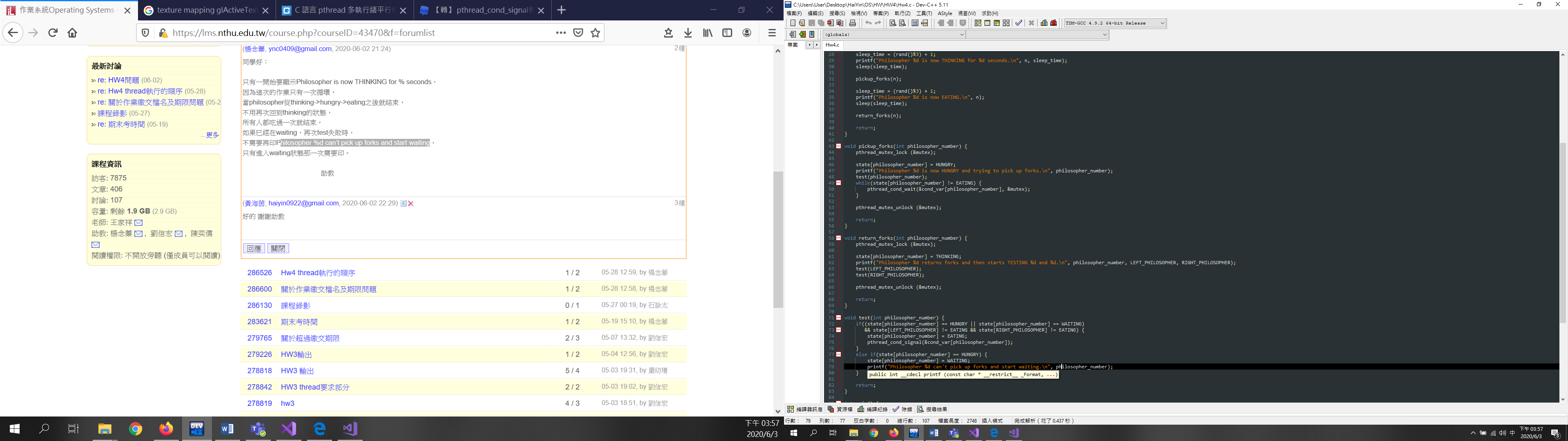
在pickup\_forks中使用pthread\_mutex\_lock，來避免兩位philosopher同時拿起同一隻餐具而產生錯誤。也就是不可以被多個thread同時執行的程式碼片段，用mutex包起來，當一個thread執行到該處時，就會先lock起來，避免其他thread進入。若其他的thread同時也要執行該處的程式碼時，就必須等待先前的thread執行完之後，才能接著進入，這樣就可以避免多個thread混雜執行，讓結果出錯的問題。

進去了之後，把state設為HUNGRY並test是否能夠開始吃。如果不能就開始wait直到用pthread\_cond\_signal()喚醒它。最後unlock，其他thread就可以進入了。



test用來測試該philosopher是否能夠開始吃，若他是HUNGRY或WAITING的狀態，且左右兩邊的人都沒有正在吃，那麼他就可以進入EATING的state，並喚醒wait condition。

但若他是HUNGRY的狀態，且左右兩邊至少有其中一人正在吃的話，就進入WAITING的state。我原先沒有設這個state，是在討論區問了助教後得知「若已經在waiting，再次test失敗時則不需再印一次Philosopher %d can’t pick up forks and start waiting.。所以多設了一個WAITING state來判斷。



最後是return\_forks，吃完後就會進到這個function。和pickup\_forks一樣使用了mutex來上鎖，用途已經在pickup\_forks中解釋過，這邊就不再次詳述。餐具return後就可以test該philosopher的左右兩邊的人是否能夠開始吃，然後一樣unlock。