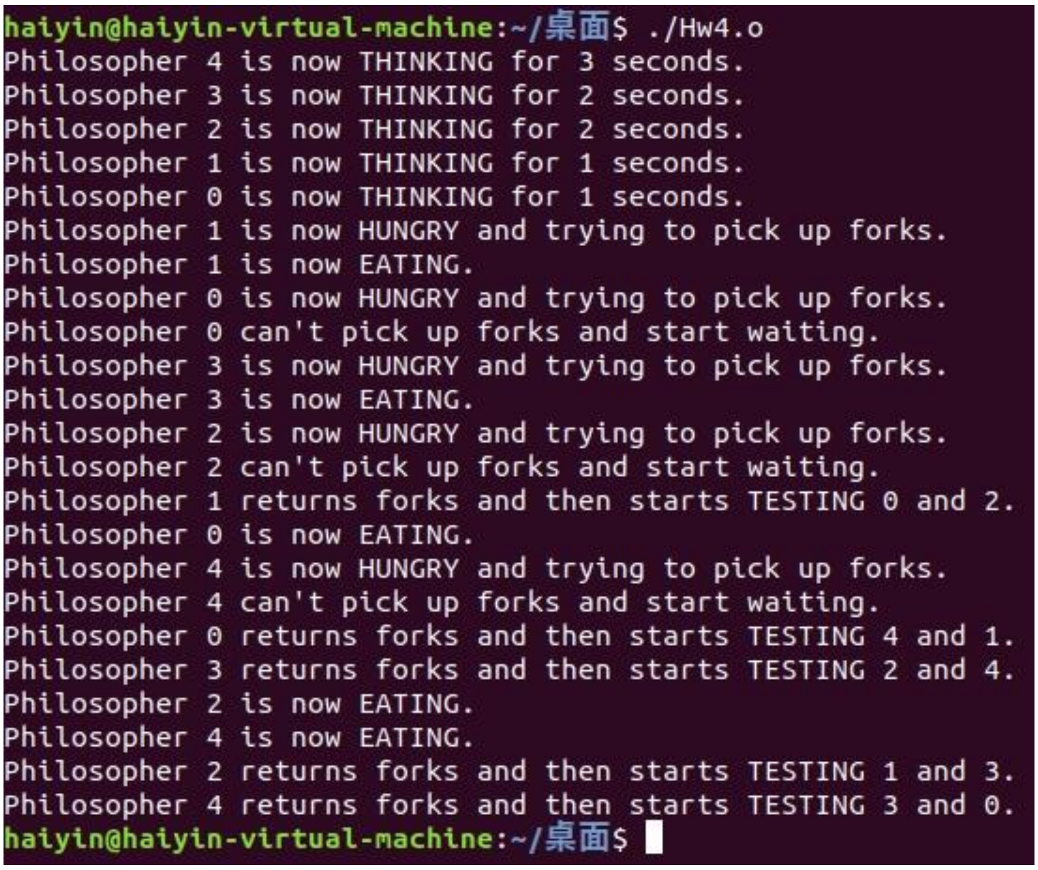
HW4 Dining Philosophers Problem

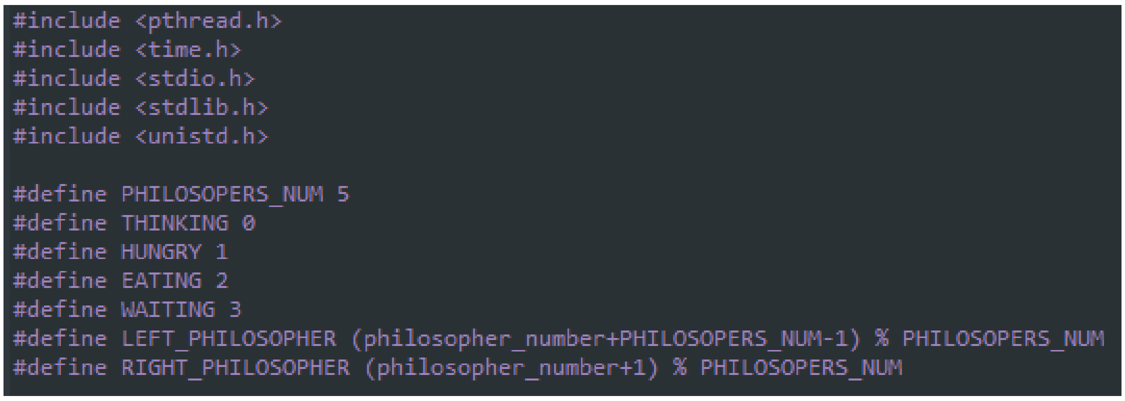
105072123 黃海茵

- OUTPUT



```
haiyin@haiyin-virtual-machine:~/桌面$ ./Hw4.o
Philosopher 4 is now THINKING for 3 seconds.
Philosopher 3 is now THINKING for 2 seconds.
Philosopher 2 is now THINKING for 2 seconds.
Philosopher 1 is now THINKING for 1 seconds.
Philosopher 0 is now THINKING for 1 seconds.
Philosopher 1 is now HUNGRY and trying to pick up forks.
Philosopher 1 is now EATING.
Philosopher 0 is now HUNGRY and trying to pick up forks.
Philosopher 0 can't pick up forks and start waiting.
Philosopher 3 is now HUNGRY and trying to pick up forks.
Philosopher 3 is now EATING.
Philosopher 2 is now HUNGRY and trying to pick up forks.
Philosopher 2 can't pick up forks and start waiting.
Philosopher 1 returns forks and then starts TESTING 0 and 2.
Philosopher 0 is now EATING.
Philosopher 4 is now HUNGRY and trying to pick up forks.
Philosopher 4 can't pick up forks and start waiting.
Philosopher 0 returns forks and then starts TESTING 4 and 1.
Philosopher 3 returns forks and then starts TESTING 2 and 4.
Philosopher 2 is now EATING.
Philosopher 4 is now EATING.
Philosopher 2 returns forks and then starts TESTING 1 and 3.
Philosopher 4 returns forks and then starts TESTING 3 and 0.
haiyin@haiyin-virtual-machine:~/桌面$ 
```

- CODE



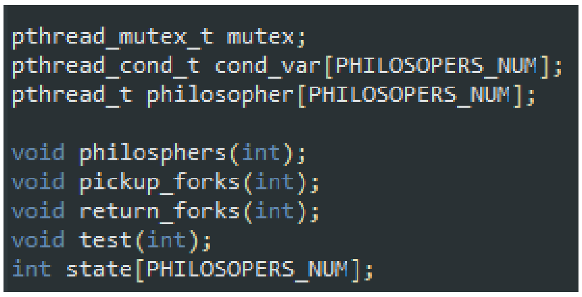
```c
#include <pthread.h>
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

#define PHILOSOPERS_NUM 5
#define THINKING 0
#define HUNGRY 1
#define EATING 2
#define WAITING 3
#define LEFT_PHILOSOPHER (philosopher_number+PHILOSOPERS_NUM-1) % PHILOSOPERS_NUM
#define RIGHT_PHILOSOPHER (philosopher_number+1) % PHILOSOPERS_NUM
```

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

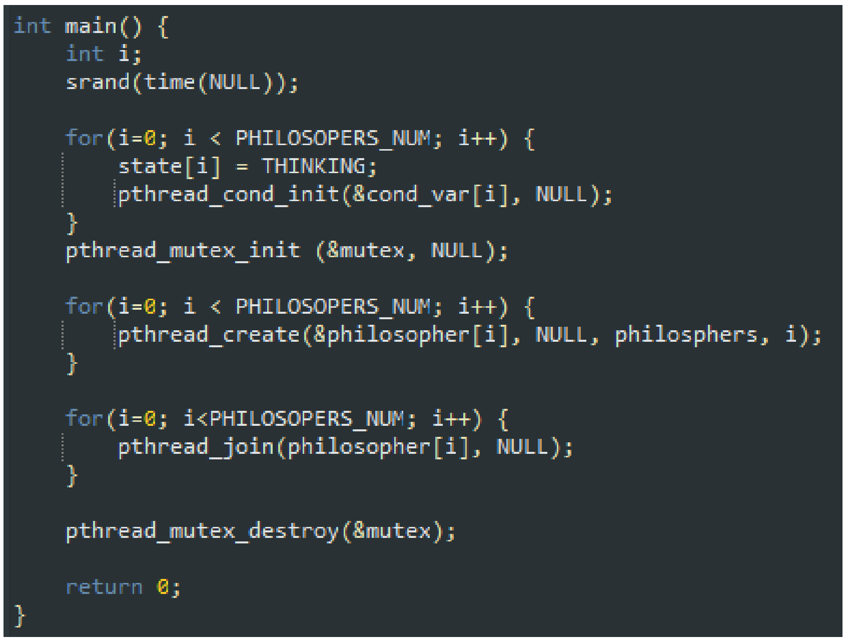
THINKING, HUNGRY, EATING, WAITING 代表四個不同的 state

LEFT_ PHILOSOPERSR 跟 RIGHT_ PHILOSOPERS 代表左邊和右邊的 philosopher



```c
pthread_mutex_t mutex;
pthread_cond_t cond_var[PHILOSOPERS_NUM];
pthread_t philosopher[PHILOSOPERS_NUM];

void philosphers(int);
void pickup_forks(int);
void return_forks(int);
void test(int);
int state[PHILOSOPERS_NUM];
```

宣告會用到的變數及 function。



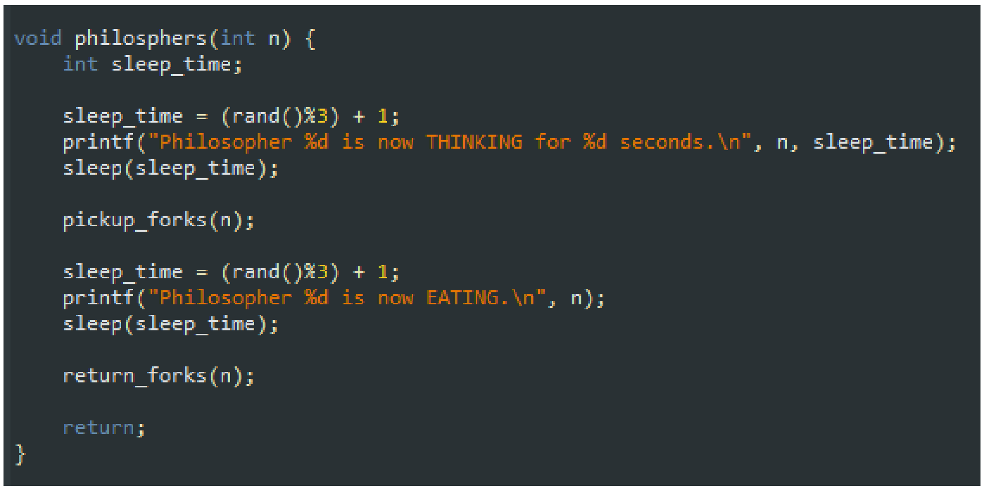
```c
int main() {
    int i;
    srand(time(NULL));

    for(i=0; i < PHILOSOPERS_NUM; i++) {
        state[i] = THINKING;
        pthread_cond_init(&cond_var[i], NULL);
    }
    pthread_mutex_init (&mutex, NULL);

    for(i=0; i < PHILOSOPERS_NUM; i++) {
        pthread_create(&philosopher[i], NULL, philosphers, i);
    }

    for(i=0; i<PHILOSOPERS_NUM; i++) {
        pthread_join(philosopher[i], NULL);
    }

    pthread_mutex_destroy(&mutex);

    return 0;
}
```

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。

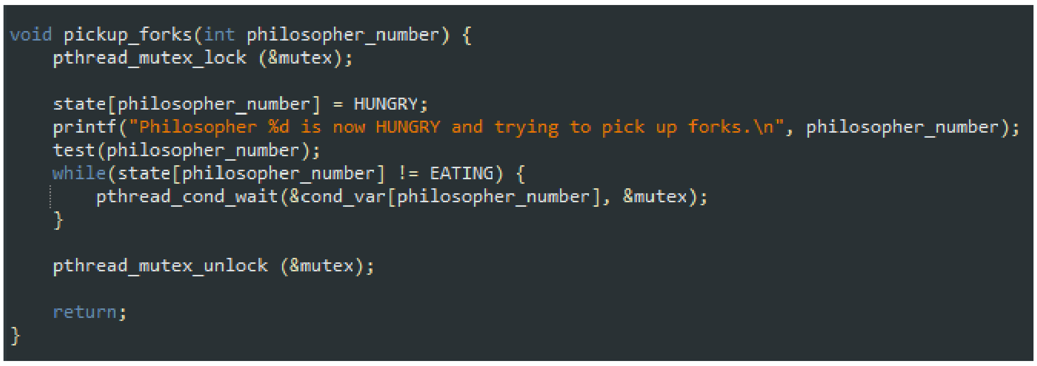
接下來 pthread_join 用來等待一個 thread 的結束。第一個參數為被等待的 thread 識別符號，第二個引數為一個使用者定義的指標，它可以用來儲存被等待 thread 的 return 值。這個 function 是一個 thread 阻塞的 function，呼叫它的 function 將一直等待到被等待的 thread 結束為止，當 function return 時，被等待 thread 的資源被收回。

最後使用 pthread_mutex_destroy 來 destroy pthread_mutex_t。



```c
void philosphers(int n) {
    int sleep_time;

    sleep_time = (rand()%3) + 1;
    printf("Philosopher %d is now THINKING for %d seconds.\n", n, sleep_time);
    sleep(sleep_time);

    pickup_forks(n);

    sleep_time = (rand()%3) + 1;
    printf("Philosopher %d is now EATING.\n", n);
    sleep(sleep_time);

    return_forks(n);

    return;
}
```

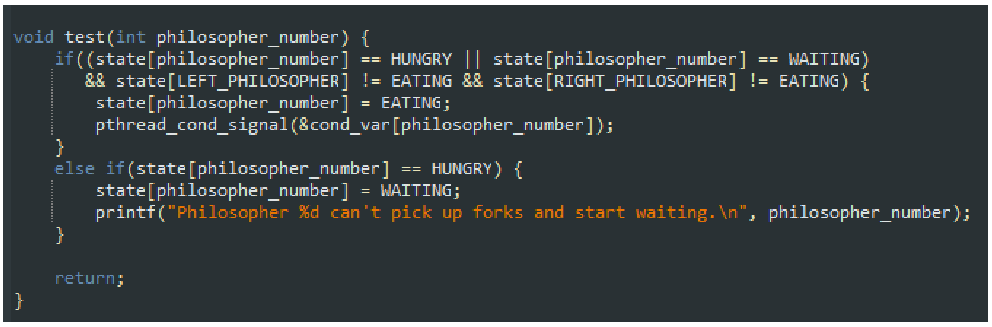
根據題目要求，把 sleep_time 設為 1~3 的 random number，模擬 thinking 的時間。thinking 結束後，就可以 pickup_forks。然後根據題目要求，再把 sleep_time 設為 1~3 的 random number，模擬 eating 的時間，eating 結束後，就可以 return_forks。



```c
void pickup_forks(int philosopher_number) {
    pthread_mutex_lock (&mutex);

    state[philosopher_number] = HUNGRY;
    printf("Philosopher %d is now HUNGRY and trying to pick up forks.\n", philosopher_number);
    test(philosopher_number);
    while(state[philosopher_number] != EATING) {
        pthread_cond_wait(&cond_var[philosopher_number], &mutex);
    }

    pthread_mutex_unlock (&mutex);

    return;
}
```

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。

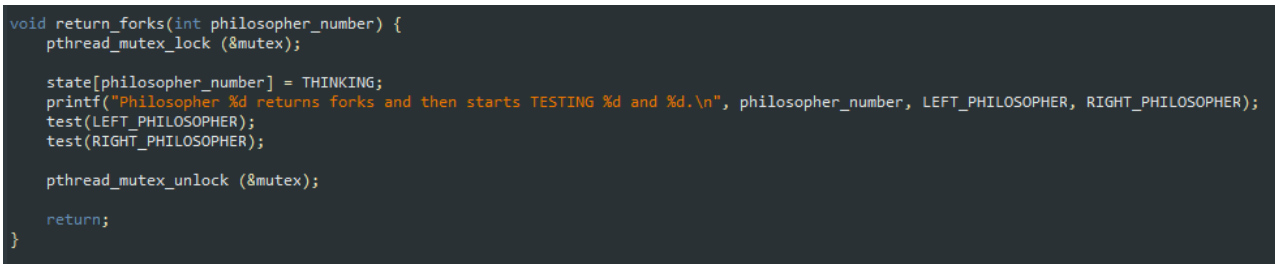
進去了之後，把 state 設為 HUNGRY 並 test 是否能夠開始吃。如果不能就開始 wait 直到用 pthread_cond_signal()喚醒它。最後 unlock，其他 thread 就可以進入了。



```c
void test(int philosopher_number) {
    if((state[philosopher_number] == HUNGRY || state[philosopher_number] == WAITING)
       && state[LEFT_PHILOSOPHER] != EATING && state[RIGHT_PHILOSOPHER] != EATING) {
        state[philosopher_number] = EATING;
        pthread_cond_signal(&cond_var[philosopher_number]);
    }
    else if(state[philosopher_number] == HUNGRY) {
        state[philosopher_number] = WAITING;
        printf("Philosopher %d can't pick up forks and start waiting.\n", philosopher_number);
    }

    return;
}
```

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。



```c
void return_forks(int philosopher_number) {
    pthread_mutex_lock (&mutex);

    state[philosopher_number] = THINKING;
    printf("Philosopher %d returns forks and then starts TESTING %d and %d.\n", philosopher_number, LEFT_PHILOSOPHER, RIGHT_PHILOSOPHER);
    test(LEFT_PHILOSOPHER);
    test(RIGHT_PHILOSOPHER);

    pthread_mutex_unlock (&mutex);

    return;
}
```

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。