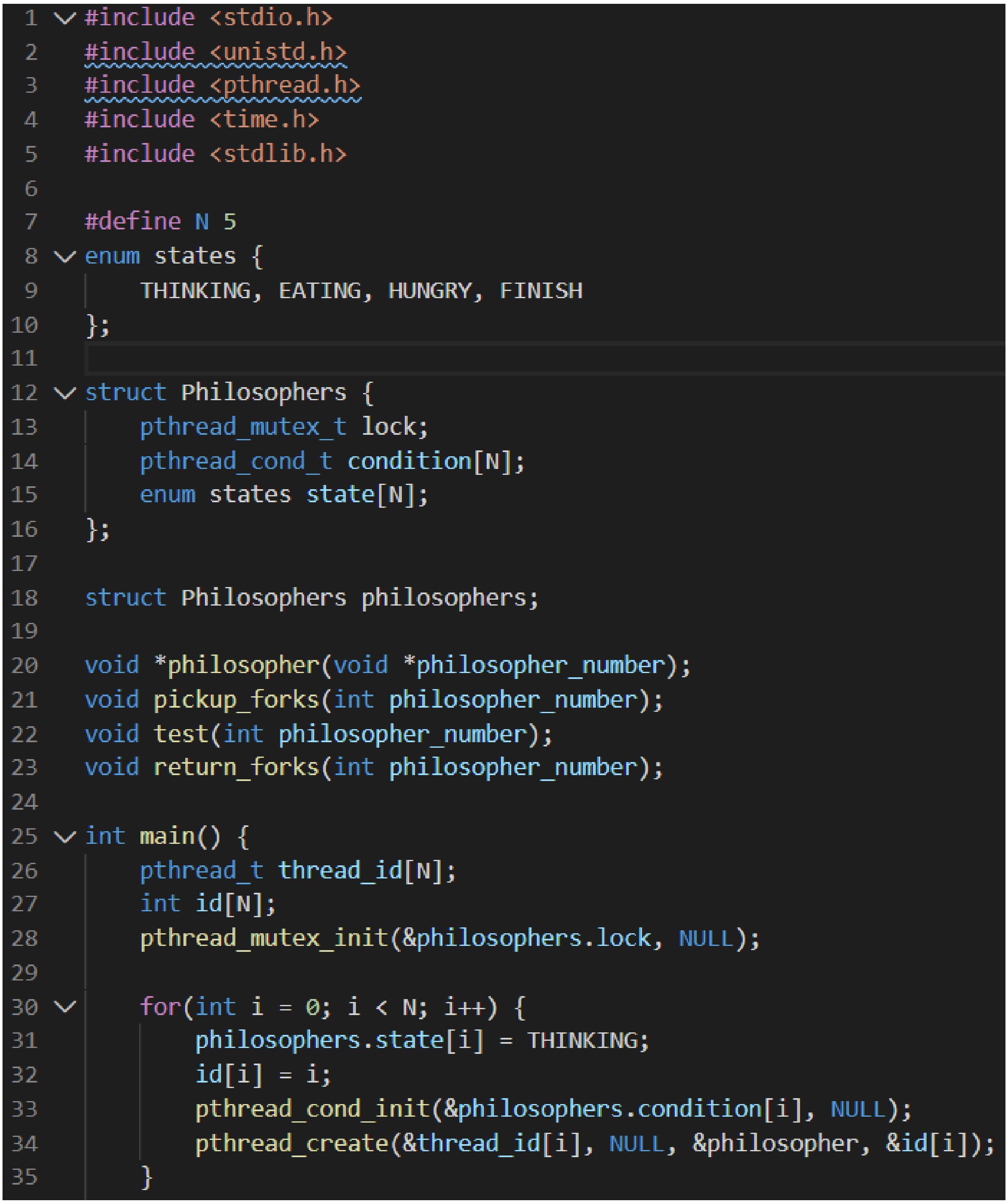
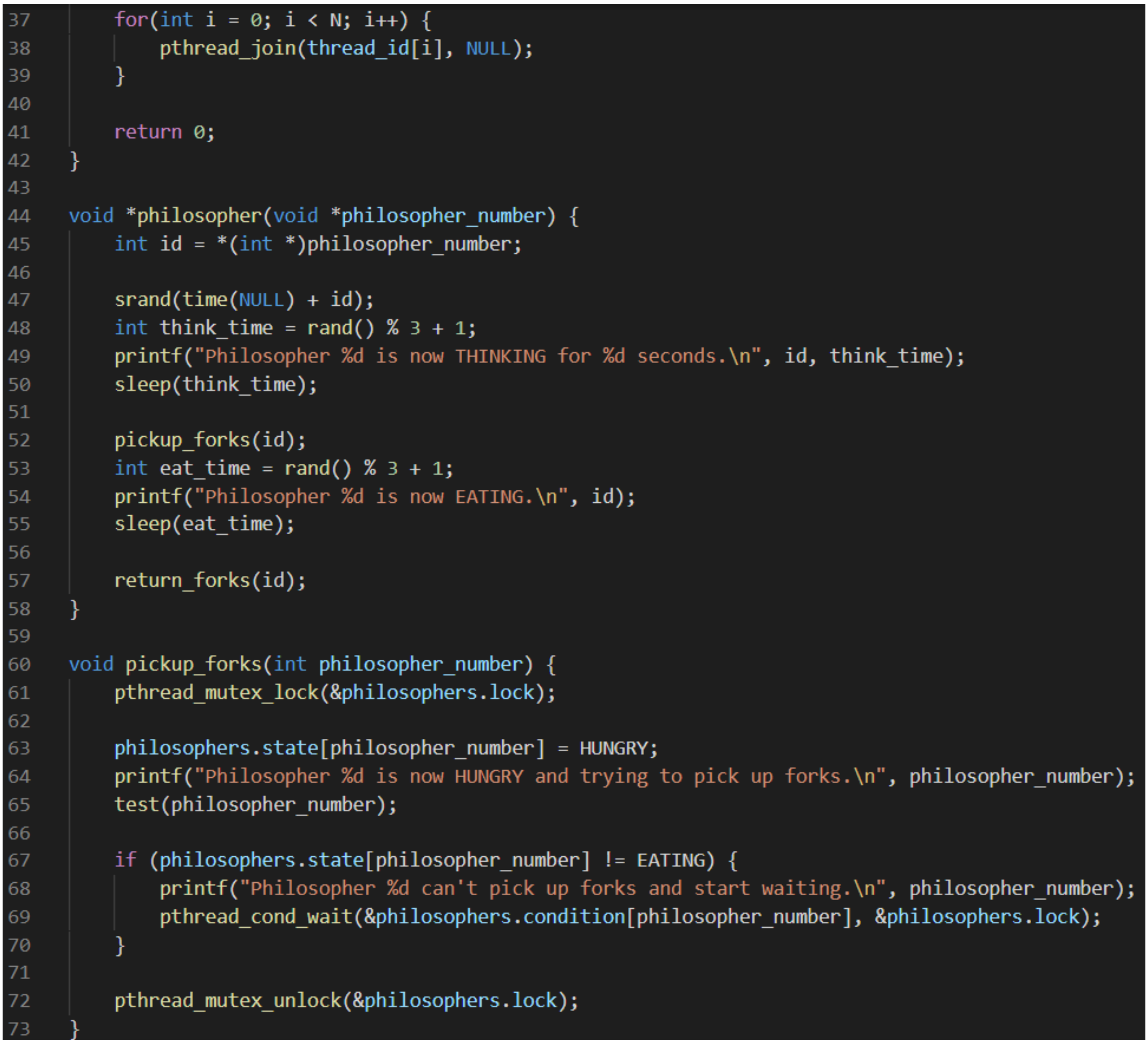
106030012 廖昱瑋 工院21 Hw4

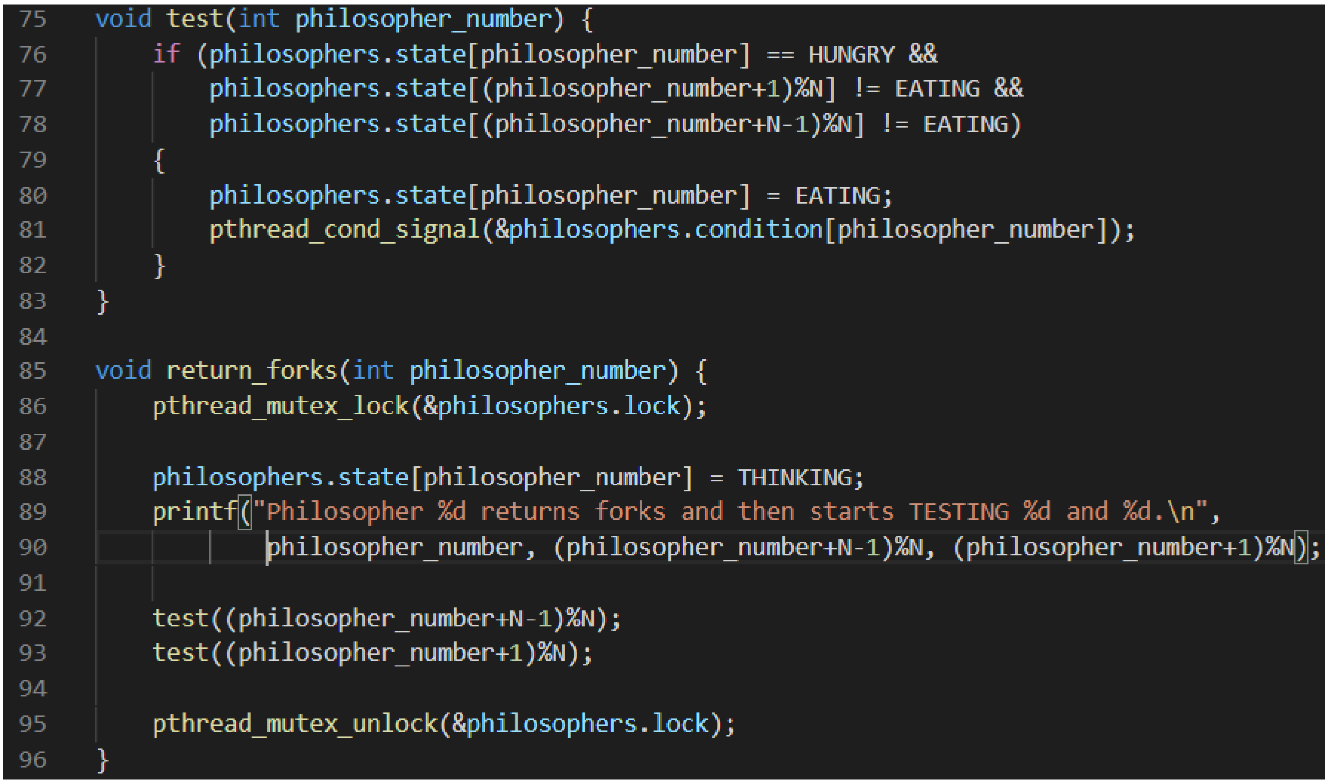
一、code 截圖



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

#define N 5
enum states {
    THINKING, EATING, HUNGRY, FINISH
};

struct Philosophers {
    pthread_mutex_t lock;
    pthread_cond_t condition[N];
    enum states state[N];
};

struct Philosophers philosophers;

void *philosopher(void *philosopher_number);
void pickup_forks(int philosopher_number);
void test(int philosopher_number);
void return_forks(int philosopher_number);

int main() {
    pthread_t thread_id[N];
    int id[N];
    pthread_mutex_init(&philosophers.lock, NULL);

    for(int i = 0; i < N; i++) {
        philosophers.state[i] = THINKING;
        id[i] = i;
        pthread_cond_init(&philosophers.condition[i], NULL);
        pthread_create(&thread_id[i], NULL, &philosopher, &id[i]);
    }
```



```c
    for(int i = 0; i < N; i++) {
        pthread_join(thread_id[i], NULL);
    }

    return 0;
}

void *philosopher(void *philosopher_number) {
    int id = *(int *)philosopher_number;

    srand(time(NULL) + id);
    int think_time = rand() % 3 + 1;
    printf("Philosopher %d is now THINKING for %d seconds.\n", id, think_time);
    sleep(think_time);

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

    return_forks(id);
}

void pickup_forks(int philosopher_number) {
    pthread_mutex_lock(&philosophers.lock);

    philosophers.state[philosopher_number] = HUNGRY;
    printf("Philosopher %d is now HUNGRY and trying to pick up forks.\n", philosopher_number);
    test(philosopher_number);

    if (philosophers.state[philosopher_number] != EATING) {
        printf("Philosopher %d can't pick up forks and start waiting.\n", philosopher_number);
        pthread_cond_wait(&philosophers.condition[philosopher_number], &philosophers.lock);
    }

    pthread_mutex_unlock(&philosophers.lock);
}
```



```c
void test(int philosopher_number) {
    if (philosophers.state[philosopher_number] == HUNGRY &&
        philosophers.state[(philosopher_number+1)%N] != EATING &&
        philosophers.state[(philosopher_number+N-1)%N] != EATING)
    {
        philosophers.state[philosopher_number] = EATING;
        pthread_cond_signal(&philosophers.condition[philosopher_number]);
    }
}

void return_forks(int philosopher_number) {
    pthread_mutex_lock(&philosophers.lock);

    philosophers.state[philosopher_number] = THINKING;
    printf("Philosopher %d returns forks and then starts TESTING %d and %d.\n",
            philosopher_number, (philosopher_number+N-1)%N, (philosopher_number+1)%N);

    test((philosopher_number+N-1)%N);
    test((philosopher_number+1)%N);

    pthread_mutex_unlock(&philosophers.lock);
}
```

二、說明

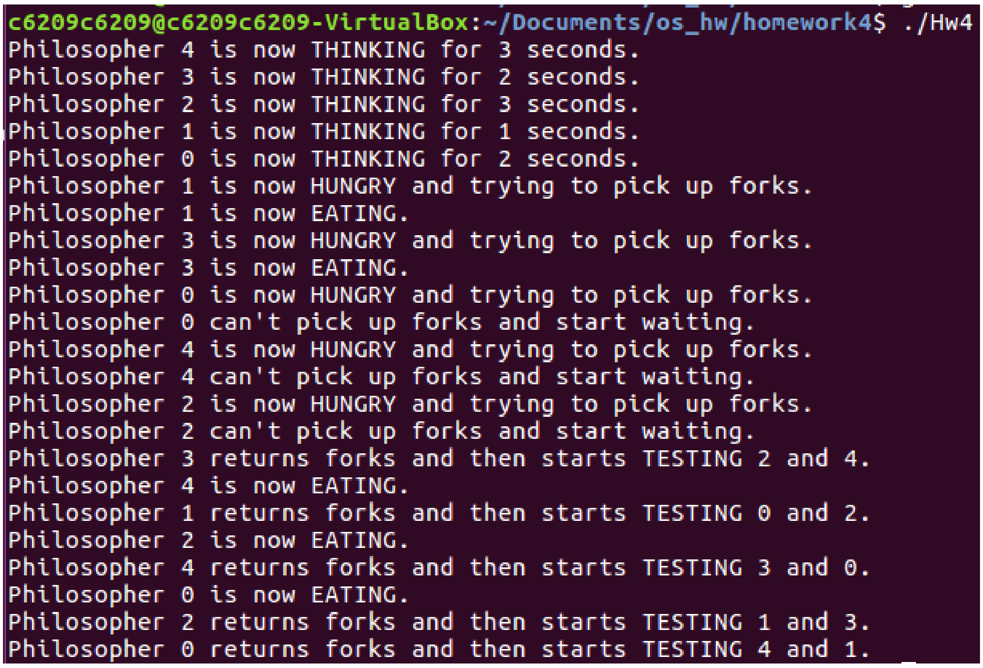
宣告struct Philosophers包含lock、condition[5]跟state[5]，condition用來存是否philosopher正在waiting、state用來存philosopher的狀態(THINKING、EATING、HUNGRY)。一開始先設每個state為THINKING，並且initial lock及condition，然後依序為每個philosophers create thread。

在philosopher function裡，先random出think_time，讓process sleep，再進入pickup_forks。因為怕會有左右兩邊同時要拿起筷子的事情發生，pickup_forks function整個用mutex_lock包起來，並在裡面設state為HUNGRY，接著用test測試可不可以拿筷子，若不能拿筷子便用pthread_cond_wait等待直到上一個philosopher吃完觸發pthread_cond_signal。

再test function裡，測試自己是不是HUNGRY並且左右兩邊philosopher都不是正在吃，這樣就可以把自己的state改成EATING，並呼叫pthread_cond_signal，代表已經拿到筷子了。

接著random出eat_time，讓process sleep，再return_forks。return_forks中為了確保放下筷子後test左右兩邊產生大家互搶的情況，所以也整個用mutex_lock包起來。在return_forks裡，先將state設回THINKING，接著依序test左邊及右邊的人，看他們可不可以拿筷子。最後全部做完再用pthread_join將thread關掉。

三、結果



```
c6209c6209@c6209c6209-VirtualBox:~/Documents/os_hw/homework4$ ./Hw4
Philosopher 4 is now THINKING for 3 seconds.
Philosopher 3 is now THINKING for 2 seconds.
Philosopher 2 is now THINKING for 3 seconds.
Philosopher 1 is now THINKING for 1 seconds.
Philosopher 0 is now THINKING for 2 seconds.
Philosopher 1 is now HUNGRY and trying to pick up forks.
Philosopher 1 is now EATING.
Philosopher 3 is now HUNGRY and trying to pick up forks.
Philosopher 3 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 4 is now HUNGRY and trying to pick up forks.
Philosopher 4 can't pick up forks and start waiting.
Philosopher 2 is now HUNGRY and trying to pick up forks.
Philosopher 2 can't pick up forks and start waiting.
Philosopher 3 returns forks and then starts TESTING 2 and 4.
Philosopher 4 is now EATING.
Philosopher 1 returns forks and then starts TESTING 0 and 2.
Philosopher 2 is now EATING.
Philosopher 4 returns forks and then starts TESTING 3 and 0.
Philosopher 0 is now EATING.
Philosopher 2 returns forks and then starts TESTING 1 and 3.
Philosopher 0 returns forks and then starts TESTING 4 and 1.
```