OS HW4 Dining Philosophers Problem

石育瑋 108062633

Implementation

Definition

```
#define P_NUM 5
#define LEFT (philosopher_number + P_NUM - 1) % P_NUM
#define RIGHT (philosopher_number + 1) % P_NUM
```

> *P_NUM 為philosopher總人數*
> *LEFT 為當前philosopher左邊的人*
> *RIGHT 為當前philosopher右邊的人*

Global variables

```
state_t p_state[5] = {THINKING, THINKING, THINKING, THINKING, THINKING};
pthread_mutex_t mutex;
pthread_cond_t cond_var[5];
pthread_t p[5];
int arg[5] = {0, 1, 2, 3, 4};
```

> *p_state 紀錄philosopher狀態*
> *mutex 控制state[]只能由一個thread去改變狀態*
> *cond_var 控制philosopher是否可吃飯*
> *p[5] thread pool*
> *arg[5] 紀錄philosopher number*

Initialization

```
void hw4_init() {
  pthread_mutex_init(&mutex, NULL);
  for (int i = 0; i < 5; i++) {
    pthread_cond_init(&cond_var[i], NULL);
  }
```

```
  for (int i = 0; i < 5; i++) {
    pthread_create(&p[i], NULL, philosopher, &arg[i]);
  }

  for (int i = 0; i < 5; i++) {
    pthread_join(p[i], NULL);
  }
}
```

> *初始化 `mutex` 與 `cond_var`，並讓thread pool執行philosopher function.*

Thread Entry Function

```
void philosopher(void *philosopher_number_p) {
  int philosopher_number = *(int *)philosopher_number_p;
  // thinking
  think(philosopher_number);
  // hungry
  pick_up_fork(philosopher_number);
  // eating
  eat(philosopher_number);
  // end eating
  return_fork(philosopher_number);
}
```

> *轉換輸入參數為int類型，並依序執行philosopher每個狀態函數（思考，餓肚子，吃飯，吃飽）*

Thinking

```
void think(int philosopher_number) {
  int thinking_time = (rand() % 3) + 1;
  printf("Philosopher %d is now THINKING for %d seconds.\n", philosopher_numbe
r, thinking_time);
  sleep(thinking_time);
}
```

> *思考時間為隨機1~3秒*

pick_up_fork

```
void pick_up_fork(int philosopher_number) {
  printf("Philosopher %d is now HUNGRY and trying to pick up forks.\n",
         philosopher_number);
  // set philosopher state
  pthread_mutex_lock(&mutex);
  p_state[philosopher_number] = HUNGRY;
  pthread_mutex_unlock(&mutex);

  // testing
  int flag = test(philosopher_number);
  if (flag == 0) {
    printf("Philosopher %d can't pick up forks and start waiting.\n",
           philosopher_number);
  }
  pthread_mutex_lock(&mutex);
  if (p_state[philosopher_number] != EATING) {
    pthread_cond_wait(&cond_var[philosopher_number], &mutex);
  }
  pthread_mutex_unlock(&mutex);
}
```

> *拿到mutex切換philosopher狀態為 `HUNGRY`，再測試左右叉子是否空閒*

test

```
int test(int philosopher_number) {
  if (p_state[philosopher_number] == HUNGRY && p_state[LEFT] != EATING &&
      p_state[RIGHT] != EATING) {
    p_state[philosopher_number] = EATING;
    pthread_cond_signal(&cond_var[philosopher_number]);
    return 1;
  }
```

```
  return 0;
}
```

> *測試左右有沒有人在吃飯，若沒有在吃飯，則代表左右叉子空閒，當前philosopher可以吃飯，發出signal，並return 1；若不能吃飯則return 0.*

eat

```
void eat(int philosopher_number) {
  printf("Philosopher %d is now EATING.\n", philosopher_number);
  int eating_time = (rand() % 3) + 1;
  sleep(eating_time);
}
```

> *隨機1~3秒去模擬吃飯*

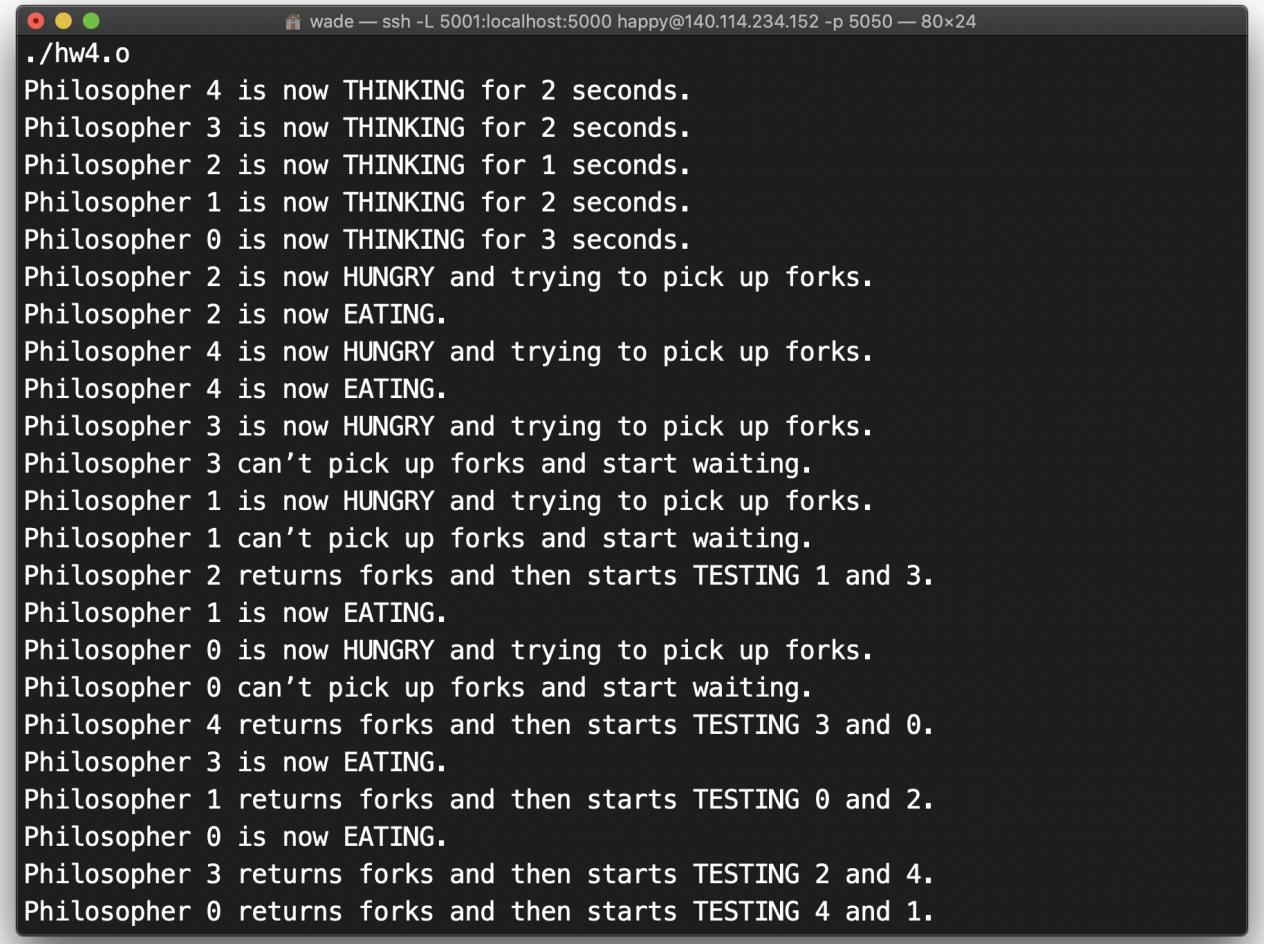
return_fork

```
void return_fork(int philosopher_number) {
  printf("Philosopher %d returns forks and then starts TESTING %d and %d.\n",
         philosopher_number, LEFT, RIGHT);
  pthread_mutex_lock(&mutex);
  p_state[philosopher_number] = THINKING;
  test(LEFT);
  test(RIGHT);
  pthread_mutex_unlock(&mutex);
}
```

> *吃飽了歸還叉子，設定自己的state為 `THINKING` ，並去測試左右兩位philosophers能否吃飯.*

Result

> *執行結果如下:*



Reference

1. https://blog.csdn.net/chengonghao/article/details/51779279?utm_medium=distribute.pc_relevant.none-task-blog-baidujs-4
2. https://blog.csdn.net/hairetz/article/details/4535920
3. http://blog.gitdns.org/2016/12/06/pthread/

Appendix - source code

Makefile

```
CC := gcc

OPS := -pthread

TARGET := hw4.c


all: hw4.o
```

```
.PHONY: clean all run style

hw4.o: $(TARGET)
    $(CC) $< $(OPS) -o [email protected]

run:
    ./hw4.o

clean:
    rm *.o

style:
    clang-format-6.0 -style=google -i *.c
```

hw4.c

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

#define P_NUM 5
#define LEFT (philosopher_number + P_NUM - 1) % P_NUM
#define RIGHT (philosopher_number + 1) % P_NUM

typedef enum state { THINKING, HUNGRY, EATING } state_t;

state_t p_state[5] = {THINKING, THINKING, THINKING, THINKING, THINKING};
pthread_mutex_t mutex;
pthread_cond_t cond_var[5];
pthread_t p[5];
int arg[5] = {0, 1, 2, 3, 4};

void hw4_init();
void philosopher(void *philosopher_number_p);
void pick_up_fork(int philosopher_number);
```

```
void return_fork(int philosopher_number);
void think(int philosopher_number);
void eat(int philosopher_number);
int test(int philosopher_number);

int main() {
  hw4_init();

  return 0;
}

void hw4_init() {
  pthread_mutex_init(&mutex, NULL);
  for (int i = 0; i < 5; i++) {
    pthread_cond_init(&cond_var[i], NULL);
  }

  for (int i = 0; i < 5; i++) {
    pthread_create(&p[i], NULL, philosopher, &arg[i]);
  }

  for (int i = 0; i < 5; i++) {
    pthread_join(p[i], NULL);
  }
}

void philosopher(void *philosopher_number_p) {
  int philosopher_number = *(int *)philosopher_number_p;
  // thinking
  think(philosopher_number);
  // hungry
  pick_up_fork(philosopher_number);
  // eating
  eat(philosopher_number);
  // end eating
  return_fork(philosopher_number);
}
```

```
void pick_up_fork(int philosopher_number) {
  printf("Philosopher %d is now HUNGRY and trying to pick up forks.\n",
         philosopher_number);
  // set philosopher state
  pthread_mutex_lock(&mutex);
  p_state[philosopher_number] = HUNGRY;
  pthread_mutex_unlock(&mutex);

  // testing
  int flag = test(philosopher_number);
  if (flag == 0) {
    printf("Philosopher %d can't pick up forks and start waiting.\n",
           philosopher_number);
  }
  pthread_mutex_lock(&mutex);
  if (p_state[philosopher_number] != EATING) {
    pthread_cond_wait(&cond_var[philosopher_number], &mutex);
  }
  pthread_mutex_unlock(&mutex);
}

void return_fork(int philosopher_number) {
  printf("Philosopher %d returns forks and then starts TESTING %d and %d.\n",
         philosopher_number, LEFT, RIGHT);
  pthread_mutex_lock(&mutex);
  p_state[philosopher_number] = THINKING;
  test(LEFT);
  test(RIGHT);
  pthread_mutex_unlock(&mutex);
}

int test(int philosopher_number) {
  if (p_state[philosopher_number] == HUNGRY && p_state[LEFT] != EATING &&
      p_state[RIGHT] != EATING) {
    p_state[philosopher_number] = EATING;
    pthread_cond_signal(&cond_var[philosopher_number]);
```

```
    return 1;
  }
  return 0;
}

void think(int philosopher_number) {
  int thinking_time = (rand() % 3) + 1;
  printf("Philosopher %d is now THINKING for %d seconds.\n", philosopher_numbe
r,
         thinking_time);
  sleep(thinking_time);
}

void eat(int philosopher_number) {
  printf("Philosopher %d is now EATING.\n", philosopher_number);
  int eating_time = (rand() % 3) + 1;
  sleep(eating_time);
}
```

stdout

```
Philosopher 4 is now THINKING for 2 seconds.
Philosopher 3 is now THINKING for 2 seconds.
Philosopher 2 is now THINKING for 1 seconds.
Philosopher 1 is now THINKING for 2 seconds.
Philosopher 0 is now THINKING for 3 seconds.
Philosopher 2 is now HUNGRY and trying to pick up forks.
Philosopher 2 is now EATING.
Philosopher 4 is now HUNGRY and trying to pick up forks.
Philosopher 4 is now EATING.
Philosopher 3 is now HUNGRY and trying to pick up forks.
Philosopher 3 can’t pick up forks and start waiting.
Philosopher 1 is now HUNGRY and trying to pick up forks.
Philosopher 1 can’t pick up forks and start waiting.
Philosopher 2 returns forks and then starts TESTING 1 and 3.
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 4 returns forks and then starts TESTING 3 and 0.
Philosopher 3 is now EATING.
Philosopher 1 returns forks and then starts TESTING 0 and 2.
Philosopher 0 is now EATING.
Philosopher 3 returns forks and then starts TESTING 2 and 4.
Philosopher 0 returns forks and then starts TESTING 4 and 1.
```