Week12 Assignment

Deadlock

	Allocation	Max
	ABCD	ABCD
P1	0210	2310
P2	0101	0122
P3	0010	1011
P4	1100	1211

(1) Is the operating system in a safe state? Why? [10 pts]

Yes, running the safety algorithm, the sequence <P3, P2, P4, P1> can finish it

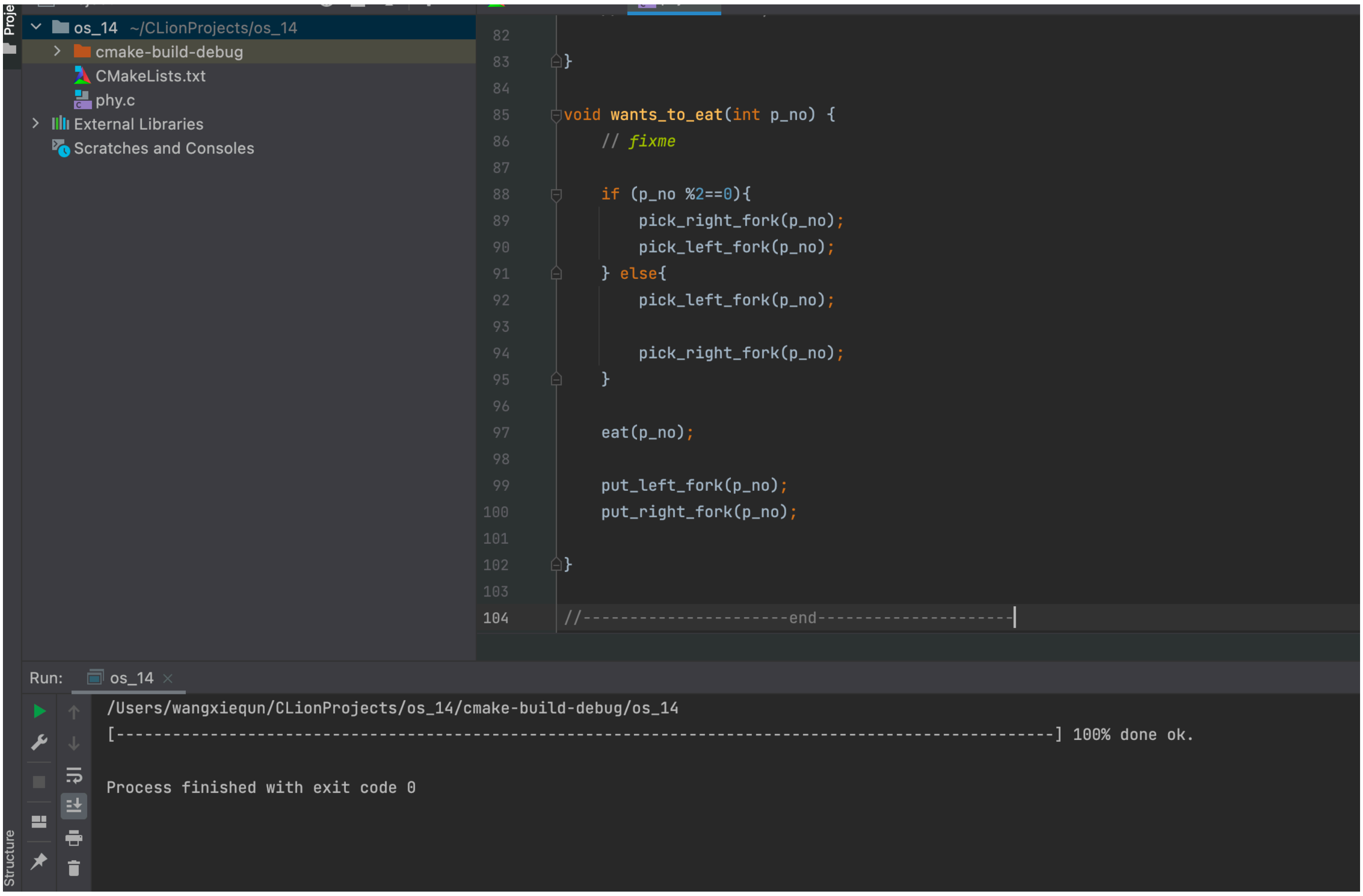
(2) If P4 requests (0,0,1,1), please run the Banker's algorithm to determine if the request should be granted. [10 pts]

1. Request4<Need4
2. Requset4<Available
3. Then it becomes an unsafe state so not be granted since when meets P3, then it deadlocks

(3) Let's assume P4's request was granted anyway (regardless of the answer to question 2). If then the processes request additional resources as follows, is the system in a deadlock state? Why? [10 pts]

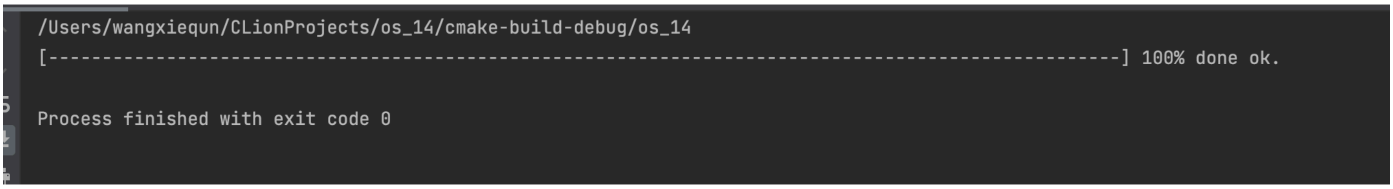
Yes, the answer showed above.

Dining philosophers problem

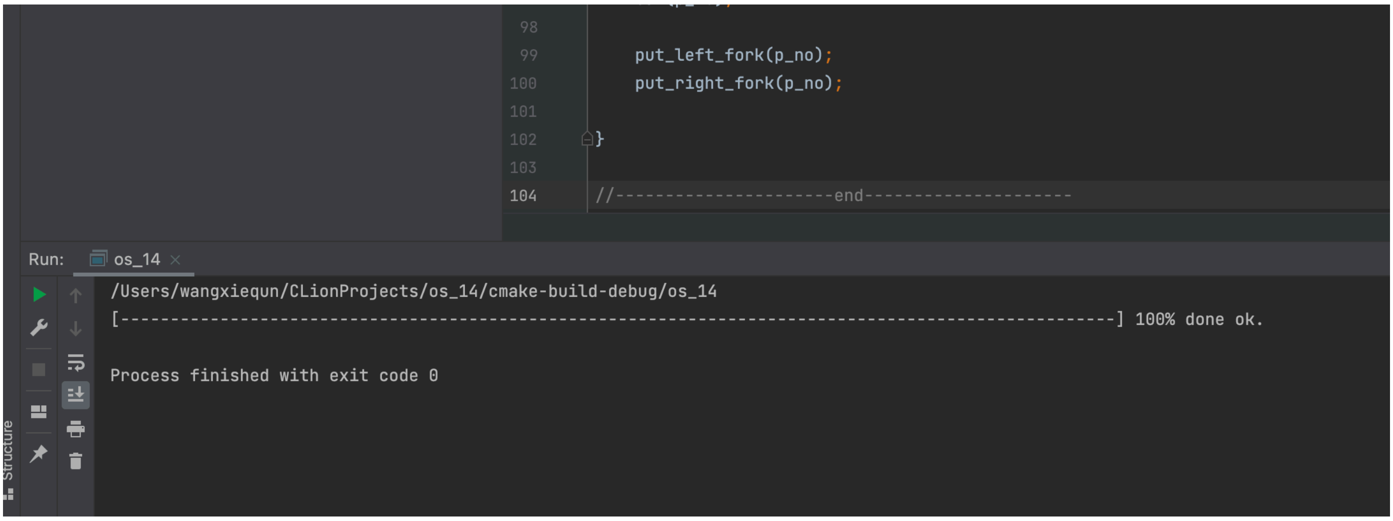


```c

}

void wants_to_eat(int p_no) {
    // fixme

    if (p_no %2==0){
        pick_right_fork(p_no);
        pick_left_fork(p_no);
    } else{
        pick_left_fork(p_no);

        pick_right_fork(p_no);
    }

    eat(p_no);

    put_left_fork(p_no);
    put_right_fork(p_no);

}

//---------------------end--------------------
```

```
/Users/wangxiequn/CLionProjects/os_14/cmake-build-debug/os_14
[----------------------------------------------------------------------------------------------------] 100% done ok.

Process finished with exit code 0
```

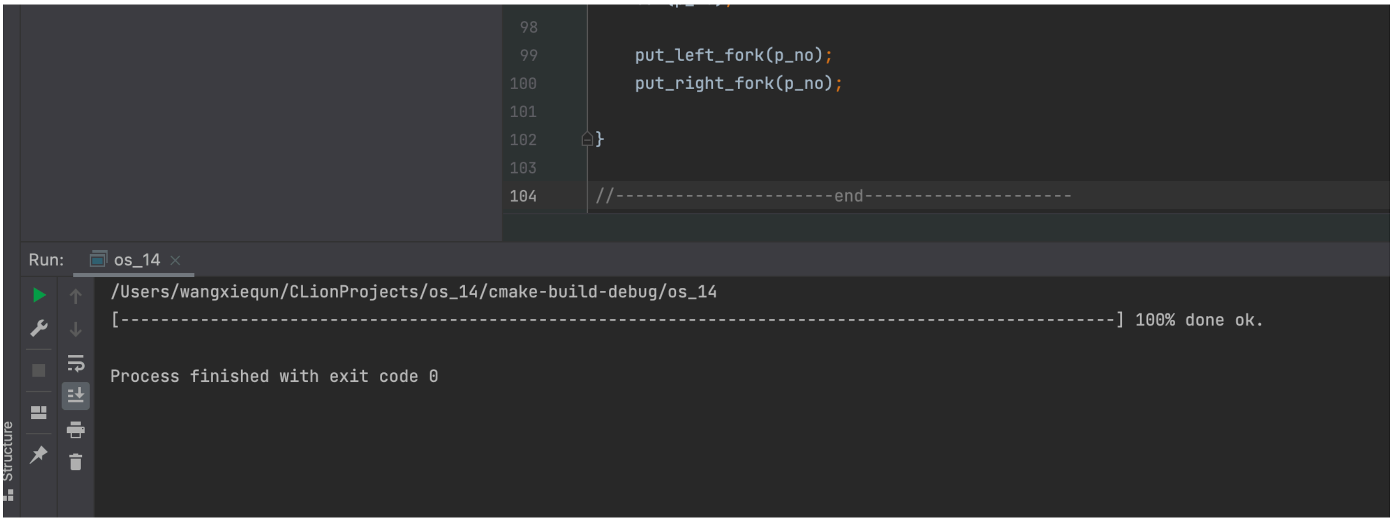


```
/Users/wangxiequn/CLionProjects/os_14/cmake-build-debug/os_14
[----------------------------------------------------------------------------------------------------] 100% done ok.

Process finished with exit code 0
```



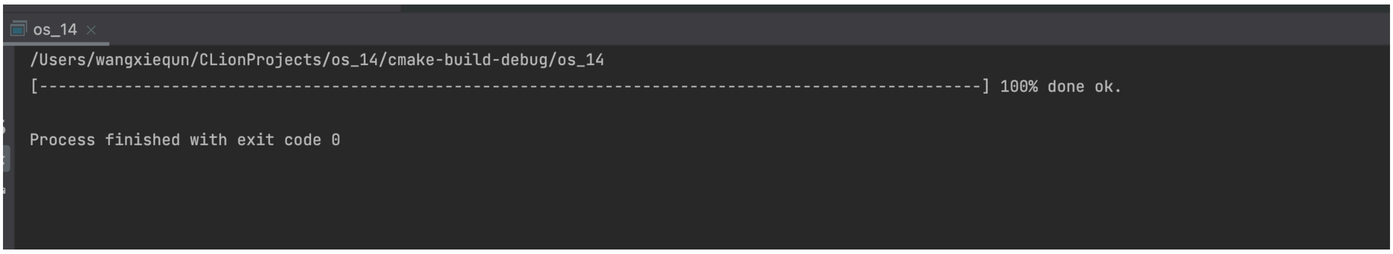
```c

    put_left_fork(p_no);
    put_right_fork(p_no);

}

//---------------------end--------------------
```

```
/Users/wangxiequn/CLionProjects/os_14/cmake-build-debug/os_14
[----------------------------------------------------------------------------------------------------] 100% done ok.

Process finished with exit code 0
```



```c

    put_left_fork(p_no);
    put_right_fork(p_no);

}

//---------------------end--------------------
```

```
/Users/wangxiequn/CLionProjects/os_14/cmake-build-debug/os_14
[----------------------------------------------------------------------------------------------------] 100% done ok.

Process finished with exit code 0
```

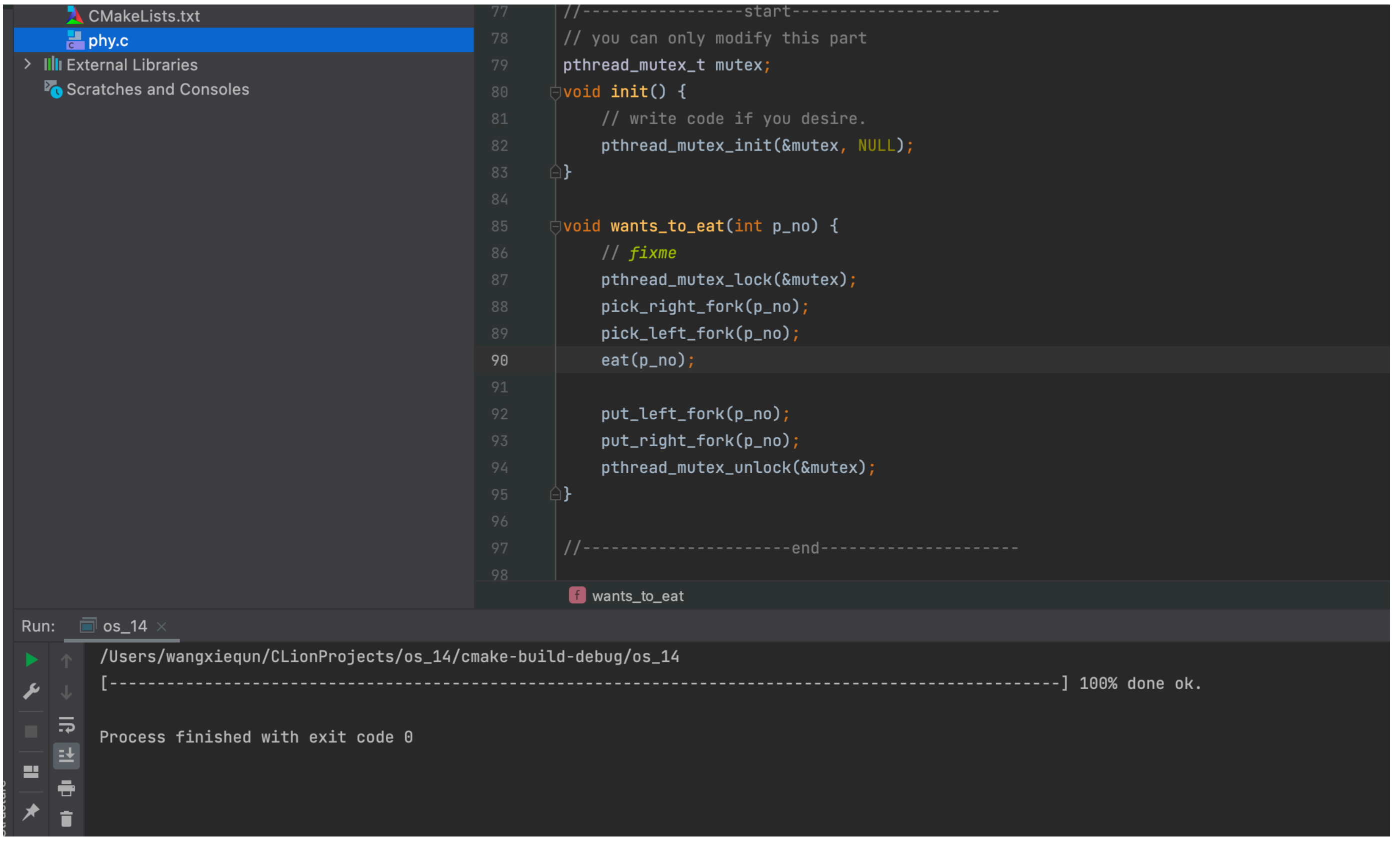


```
/Users/wangxiequn/CLionProjects/os_14/cmake-build-debug/os_14
[----------------------------------------------------------------------------------------------------] 100% done ok.

Process finished with exit code 0
```

The first method:

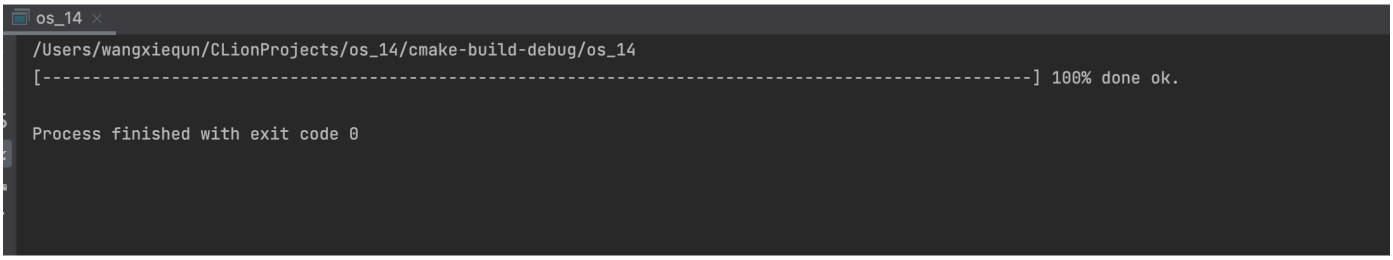
The odd-numbered philosopher is required to pick up the chopsticks on his left and then to his right, while the even-numbered philosopher is the opposite, so that a philosopher can always get two chopsticks to complete the meal, thus freeing up the resources it occupies

The second is that only one philosopher can eat at same time

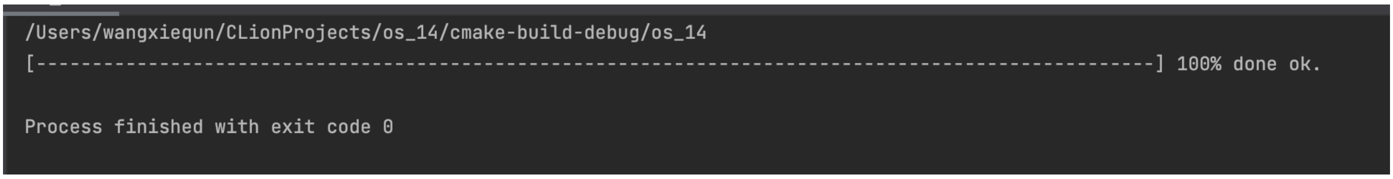


```c
//----------------start--------------------
// you can only modify this part
pthread_mutex_t mutex;
void init() {
    // write code if you desire.
    pthread_mutex_init(&mutex, NULL);
}

void wants_to_eat(int p_no) {
    // fixme
    pthread_mutex_lock(&mutex);
    pick_right_fork(p_no);
    pick_left_fork(p_no);
    eat(p_no);

    put_left_fork(p_no);
    put_right_fork(p_no);
    pthread_mutex_unlock(&mutex);
}

//--------------------end--------------------

```

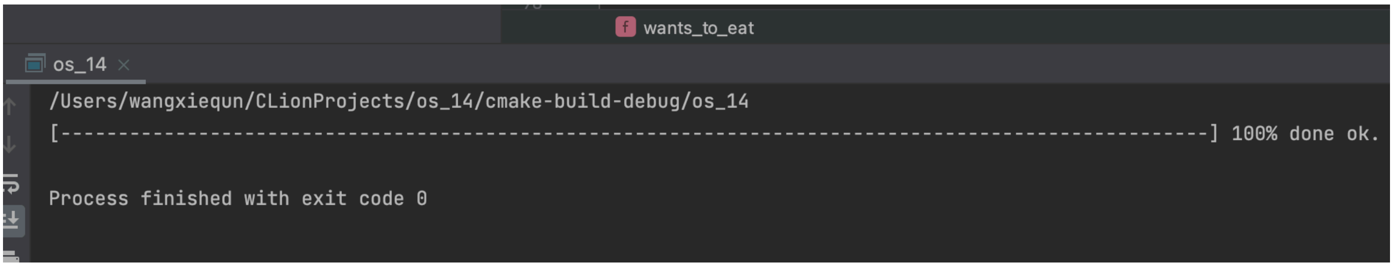
```
/Users/wangxiequn/CLionProjects/os_14/cmake-build-debug/os_14
[----------------------------------------------------------------------------------------------------] 100% done ok.

Process finished with exit code 0
```



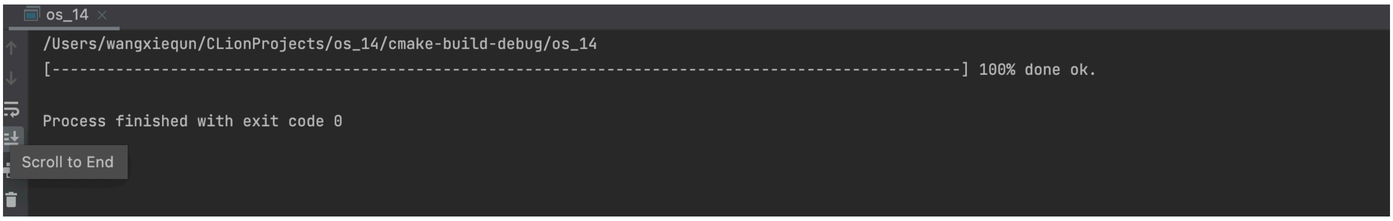
```
/Users/wangxiequn/CLionProjects/os_14/cmake-build-debug/os_14
[----------------------------------------------------------------------------------------------------] 100% done ok.

Process finished with exit code 0
```



```
/Users/wangxiequn/CLionProjects/os_14/cmake-build-debug/os_14
[----------------------------------------------------------------------------------------------------] 100% done ok.

Process finished with exit code 0
```



```
/Users/wangxiequn/CLionProjects/os_14/cmake-build-debug/os_14
[----------------------------------------------------------------------------------------------------] 100% done ok.

Process finished with exit code 0
```

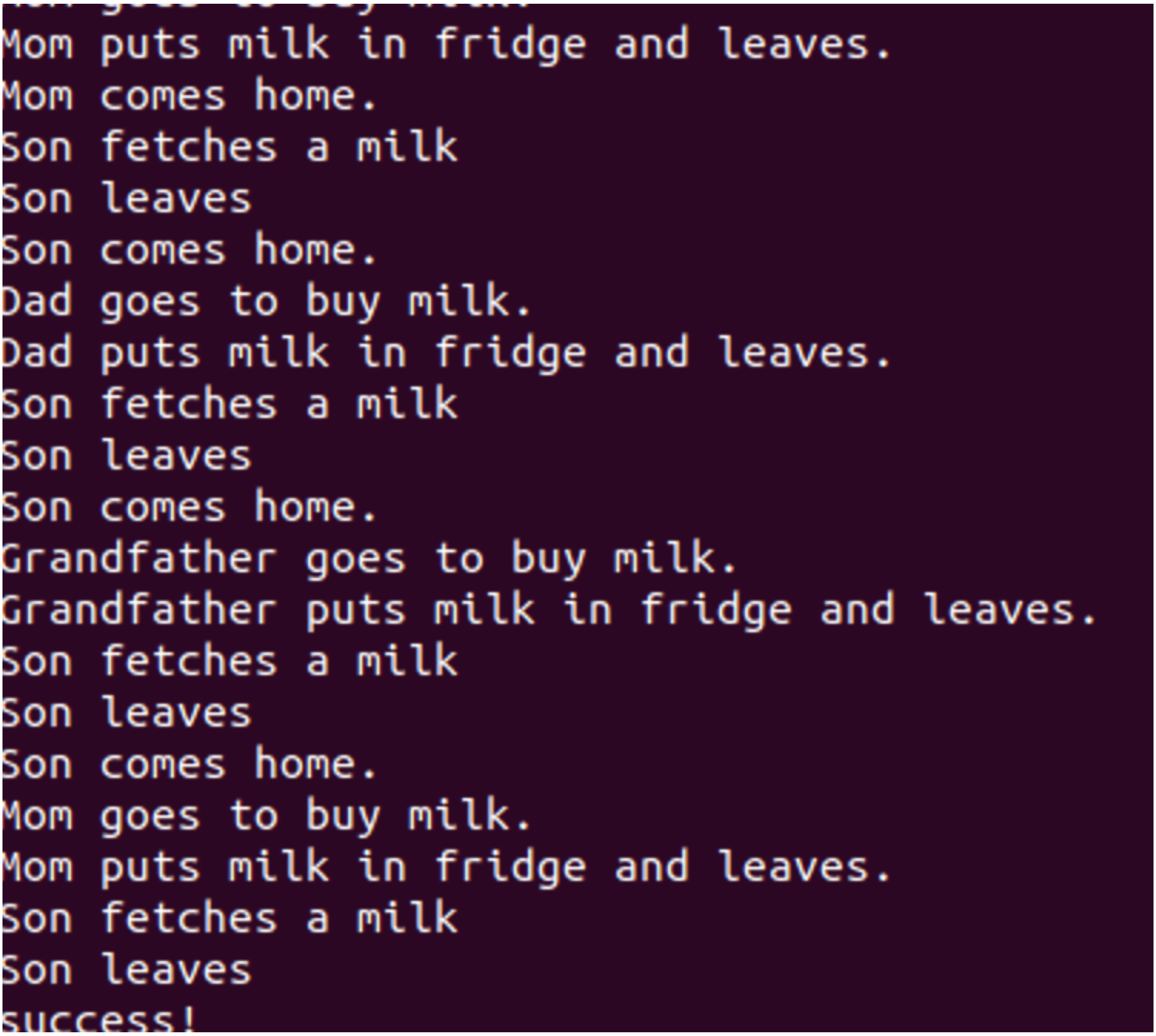


```
/Users/wangxiequn/CLionProjects/os_14/cmake-build-debug/os_14
[----------------------------------------------------------------------------------------------------] 100% done ok.

Process finished with exit code 0
```

The too much milk problem

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


sem_t sem;
sem_t sem1;
pthread_mutex_t fri_lock;

void *mom(int *num){
    for(int i=0;i<10;i++){

        printf("Mom comes home.\n");
        sleep(rand()%2+1);
        printf("Mom goes to buy milk.\n");

        *num += 1;

        sem_wait(&sem1);
        sem_post(&sem);
        if (*num > 2){
            printf("What a waste of food! The fridge can not hold so much milk!\n");
            while(1){ printf("TAT");}
        }
        printf("Mom puts milk in fridge and leaves.\n");
    }

}

void *dad(int *num){
    for(int i=0;i<10;i++){
        printf("Dad comes home.\n");


        sleep(rand()%2+1);
        printf("Dad goes to buy milk.\n");
        *num += 1;
        sem_wait(&sem1);
        sem_post(&sem);
        if (*num > 2){
            printf("What a waste of food! The fridge can not hold so much milk!\n");
            while(1){ printf("TAT");}
        }
        printf("Dad puts milk in fridge and leaves.\n");
    }
}

void *grandfather(int *num){
    for(int i=0;i<10;i++){
        printf("Grandfather comes home.\n");


        sleep(rand()%2+1);
        printf("Grandfather goes to buy milk.\n");
        *num += 1;
        sem_post(&sem);
        sem_wait(&sem1);
        if (*num > 2){
            printf("What a waste of food! The fridge can not hold so much milk!\n");
            while(1){ printf("TAT");}

        }
        printf("Grandfather puts milk in fridge and leaves.\n");
    }
}

void *son(int *num){
    for(int i = 0; i < 30 ; i++){
```

```c
        printf("Son comes home.\n");


        sem_post(&sem1);

        sem_wait(&sem);
        if(*num == 0){
            printf("The fridge is empty!\n");
            while(1){ printf("TAT");}

        }
        printf("Son fetches a milk\n");
        *num -= 1;
        printf("Son leaves\n");
    }
}

int main(int argc, char * argv[]) {
    srand(time(0));
    sem_init(&sem,0,0);
    sem_init(&sem1,0,0);
    sem_post(&sem1);
    sem_post(&sem1);
//    printf("%d",sem1);
    int num_milk = 0;
    pthread_t p1, p2, p3, p4;
    pthread_mutex_init(&fri_lock,NULL);

    // Create two threads (both run func)
    pthread_create(&p1, NULL, mom, &num_milk);
    pthread_create(&p2, NULL, dad, &num_milk);
    pthread_create(&p3, NULL, grandfather, &num_milk);
    pthread_create(&p4, NULL, son, &num_milk);

    // Wait for the threads to end.
    pthread_join(p1, NULL);
    pthread_join(p2, NULL);
    pthread_join(p3, NULL);
    pthread_join(p4, NULL);

    printf("success!\n");
    sem_destroy(&sem);
    sem_destroy(&sem1);
}
```



```
Mom goes to buy milk.
Mom puts milk in fridge and leaves.
Mom comes home.
Son fetches a milk
Son leaves
Son comes home.
Dad goes to buy milk.
Dad puts milk in fridge and leaves.
Son fetches a milk
Son leaves
Son comes home.
Grandfather goes to buy milk.
Grandfather puts milk in fridge and leaves.
Son fetches a milk
Son leaves
Son comes home.
Mom goes to buy milk.
Mom puts milk in fridge and leaves.
Son fetches a milk
Son leaves
success!
```

Set two semaphore first is sem is 0 the second sem1 is 2. Everytime buy milk let sem add 1 and sem1 reduce 1 and everytime drink milk let sem reduce 1 and sem1 add to let milk is in 0-2