CS302 OS Week12 Assignment - Report

Name: 刘仁杰
SID: 11911808

1. Deadlock

Need Matrix:

	A	B	C	D
P1	2	1	0	0
P2	0	0	2	1
P3	1	0	0	1
P4	0	1	1	1

(1) Is the operating system in a safe state? Why？

	Available	Status
T0	(1,0,1,2)	P3 done
T1	(1,0,2,2)	P2 done
T2	(1,1,2,3)	P4 done
T3	(2,2,2,3)	P1 done

So, the OS is in a safe state.

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

If P4 is granted with (0,0,1,1), then:

Need Matrix:

	A	B	C	D
P1	2	1	0	0
P2	0	0	2	1
P3	1	0	0	1
P4	0	1	0	0

	Available	Status
T0	(1,0,0,1)	P3 done
T1	(1,0,1,1)	no process can be done

This time the OS is in dead lock. So, the request from P4 should not be granted.

(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?

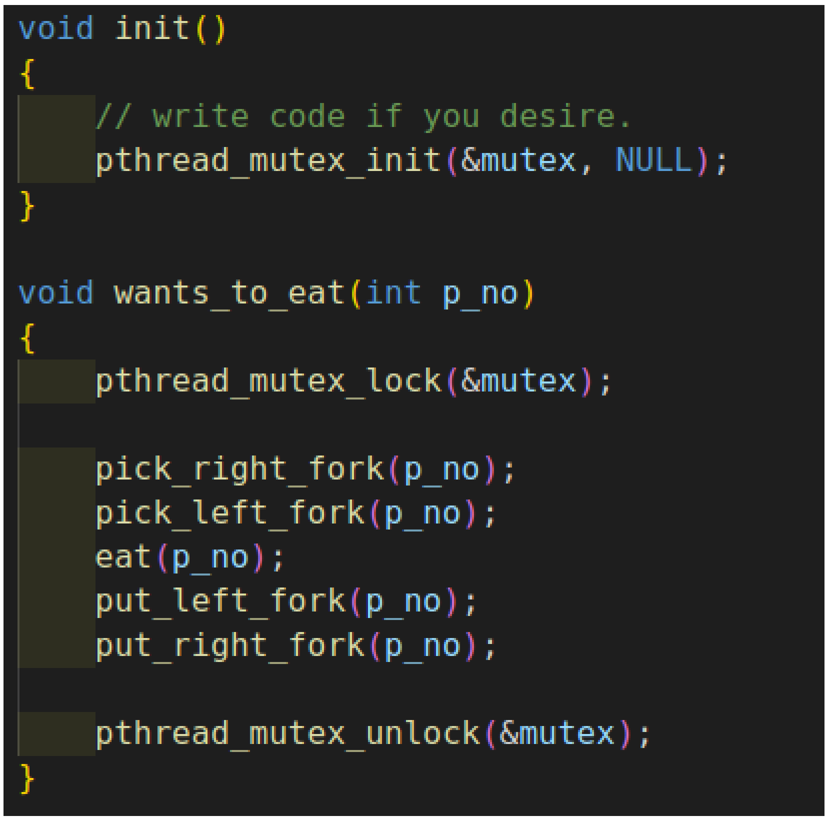
If P4 is granted with (0,0,1,1), then available is (1,0,0,1).

	Available	Status
T0	(1,0,0,1)	P3 done
T1	(1,0,1,1)	P2 done
T2	(1,1,1,2)	P4 done
T3	(2,2,1,2)	P1 done

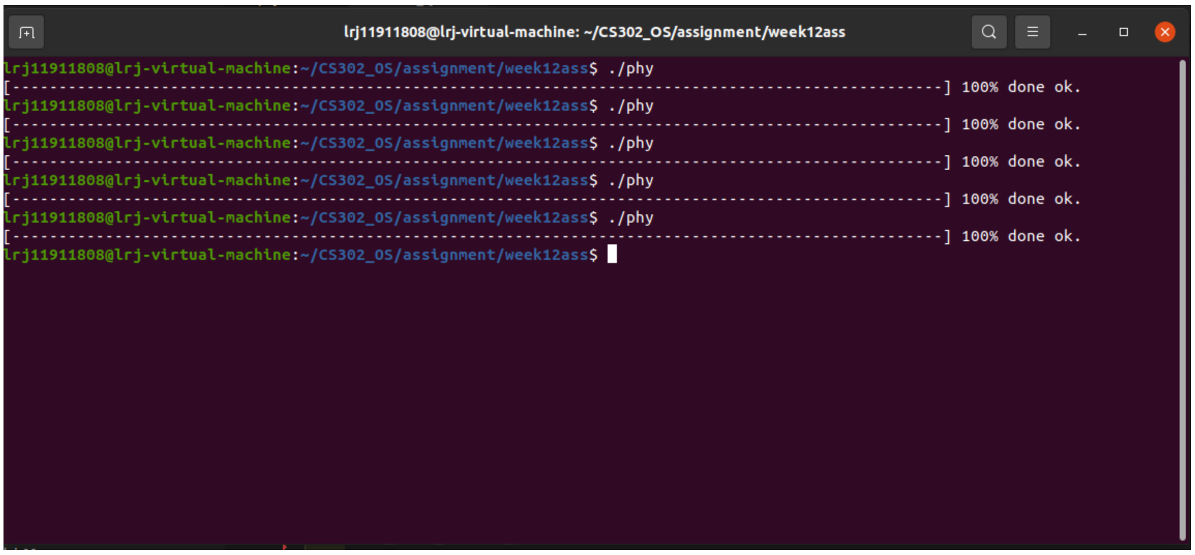
All processes are done, the system is not in a deadlock state.

2. Dining philosophers problem

- The first solution is that we just let one philosopher to eat at one time, so there are no cyclic use of resources.

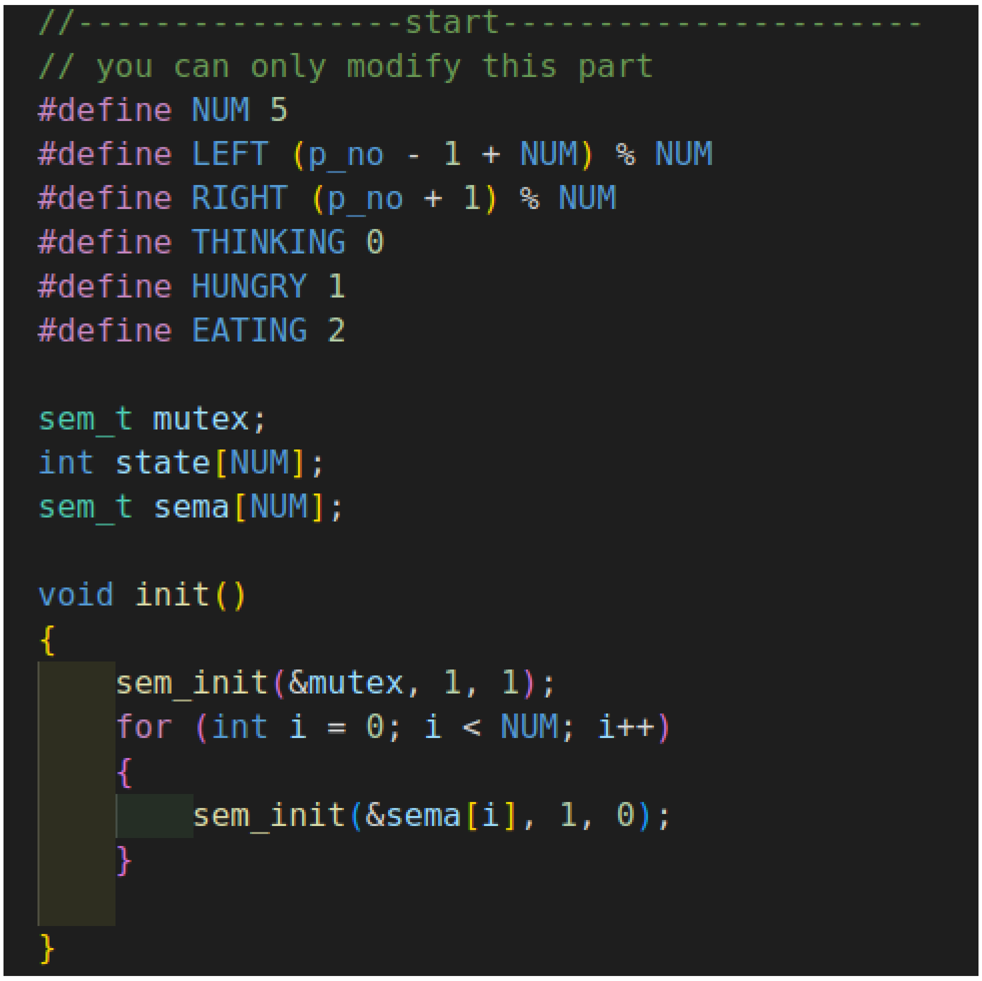


```
void init()
{
    // write code if you desire.
    pthread_mutex_init(&mutex, NULL);
}

void wants_to_eat(int p_no)
{
    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);
}
```

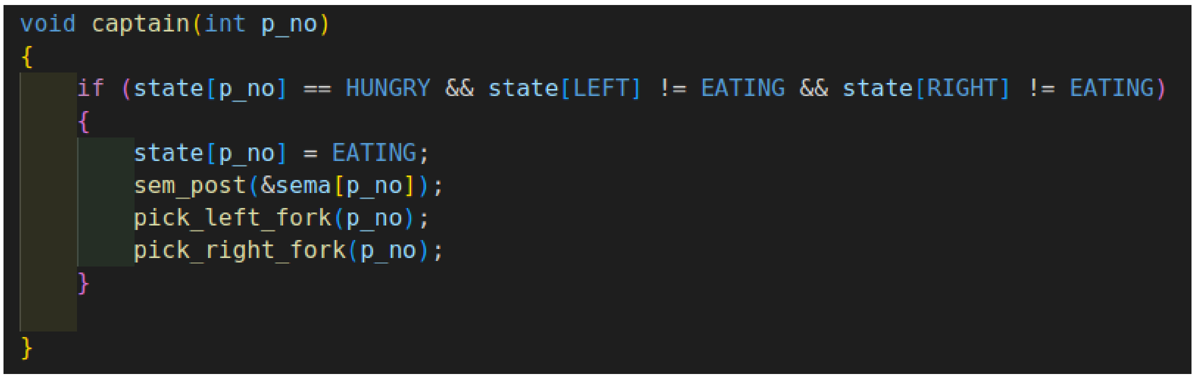


```
lrj11911808@lrj-virtual-machine:~/CS302_OS/assignment/week12ass$ ./phy
[----------------------------------------------------------------------------------------------------] 100% done ok.
lrj11911808@lrj-virtual-machine:~/CS302_OS/assignment/week12ass$ ./phy
[----------------------------------------------------------------------------------------------------] 100% done ok.
lrj11911808@lrj-virtual-machine:~/CS302_OS/assignment/week12ass$ ./phy
[----------------------------------------------------------------------------------------------------] 100% done ok.
lrj11911808@lrj-virtual-machine:~/CS302_OS/assignment/week12ass$ ./phy
[----------------------------------------------------------------------------------------------------] 100% done ok.
lrj11911808@lrj-virtual-machine:~/CS302_OS/assignment/week12ass$ ./phy
[----------------------------------------------------------------------------------------------------] 100% done ok.
lrj11911808@lrj-virtual-machine:~/CS302_OS/assignment/week12ass$ 
```

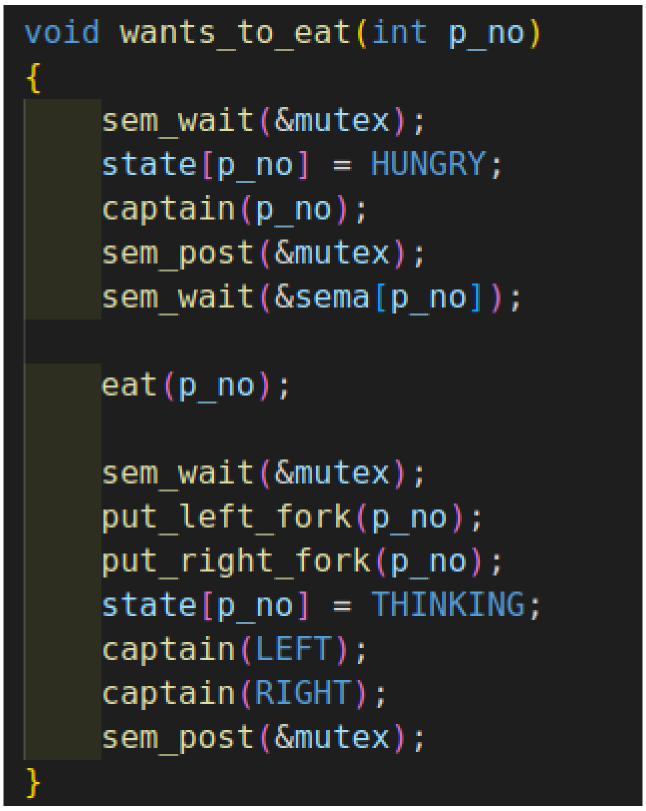
- The second solution is that we can let as many as philosophers to eat but one philosopher can eat only when his left neighbors and right neighbors are not eating. Moreover, when a philosopher ends eating, he will notify his left and right neighbors and ask them if they want to eat.



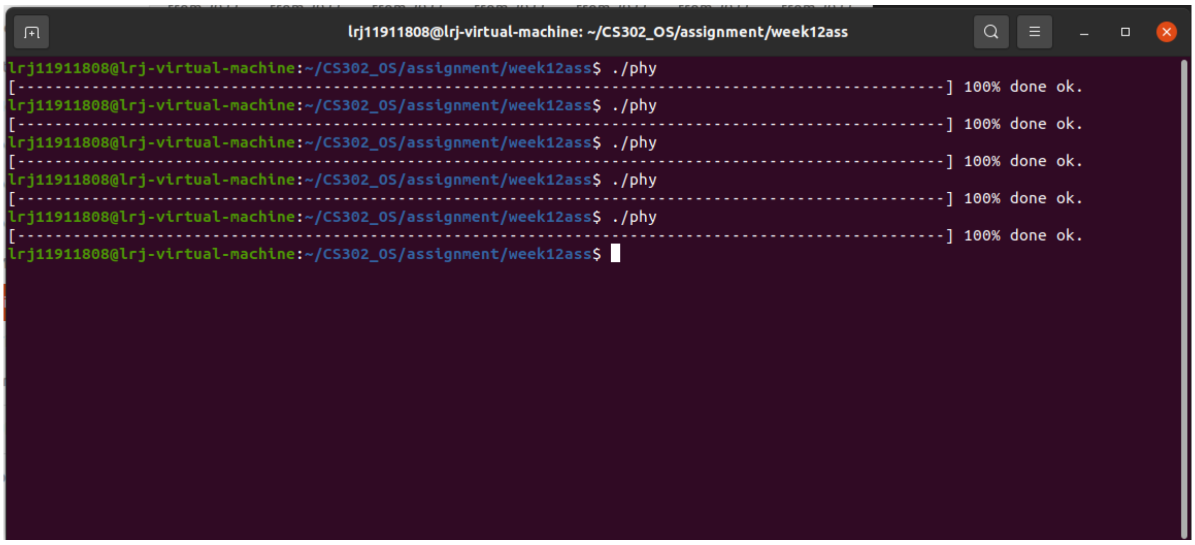
```
//-----------------start----------------------
// you can only modify this part
#define NUM 5
#define LEFT (p_no - 1 + NUM) % NUM
#define RIGHT (p_no + 1) % NUM
#define THINKING 0
#define HUNGRY 1
#define EATING 2

sem_t mutex;
int state[NUM];
sem_t sema[NUM];

void init()
{
    sem_init(&mutex, 1, 1);
    for (int i = 0; i < NUM; i++)
    {
        sem_init(&sema[i], 1, 0);
    }

}
```



```
void captain(int p_no)
{
    if (state[p_no] == HUNGRY && state[LEFT] != EATING && state[RIGHT] != EATING)
    {
        state[p_no] = EATING;
        sem_post(&sema[p_no]);
        pick_left_fork(p_no);
        pick_right_fork(p_no);
    }

}
```



```
void wants_to_eat(int p_no)
{
    sem_wait(&mutex);
    state[p_no] = HUNGRY;
    captain(p_no);
    sem_post(&mutex);
    sem_wait(&sema[p_no]);

    eat(p_no);

    sem_wait(&mutex);
    put_left_fork(p_no);
    put_right_fork(p_no);
    state[p_no] = THINKING;
    captain(LEFT);
    captain(RIGHT);
    sem_post(&mutex);
}
```



lrj11911808@lrj-virtual-machine: ~/CS302_OS/assignment/week12ass

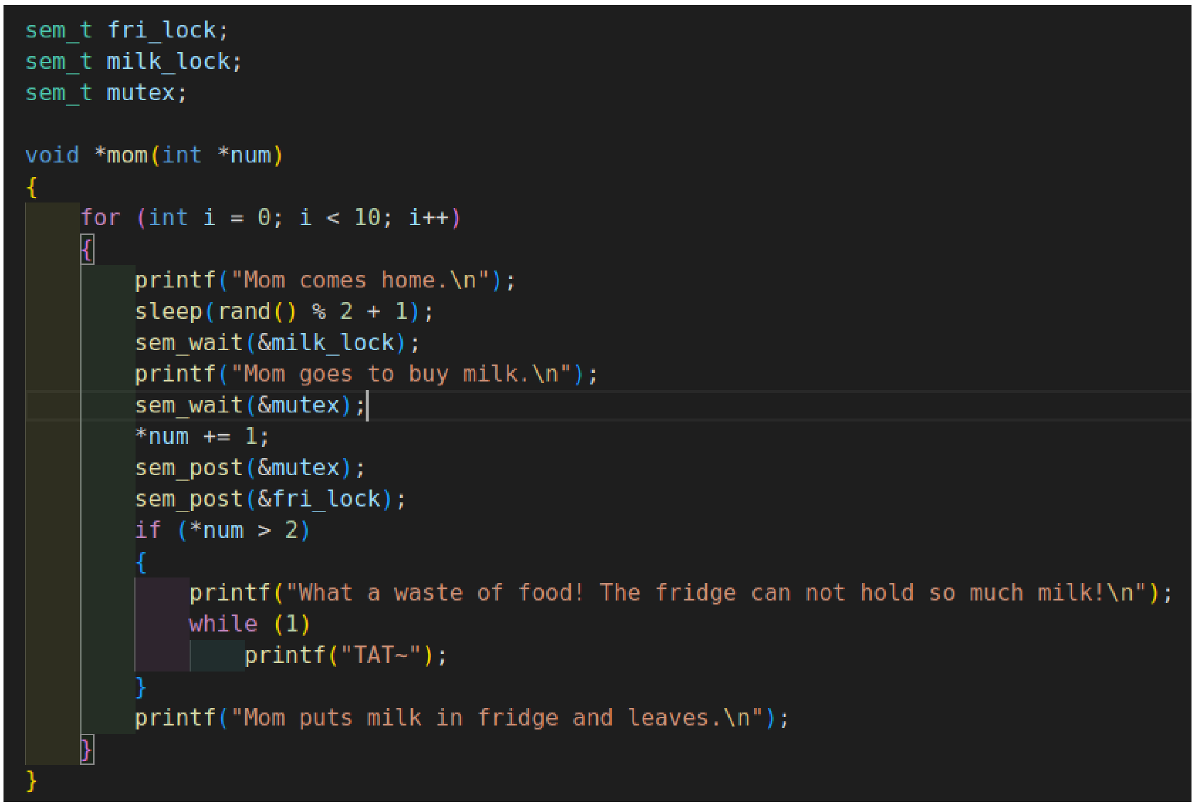
```
lrj11911808@lrj-virtual-machine:~/CS302_OS/assignment/week12ass$ ./phy
[----------------------------------------------------------------------------------------------------] 100% done ok.
lrj11911808@lrj-virtual-machine:~/CS302_OS/assignment/week12ass$ ./phy
[----------------------------------------------------------------------------------------------------] 100% done ok.
lrj11911808@lrj-virtual-machine:~/CS302_OS/assignment/week12ass$ ./phy
[----------------------------------------------------------------------------------------------------] 100% done ok.
lrj11911808@lrj-virtual-machine:~/CS302_OS/assignment/week12ass$ ./phy
[----------------------------------------------------------------------------------------------------] 100% done ok.
lrj11911808@lrj-virtual-machine:~/CS302_OS/assignment/week12ass$ ./phy
[----------------------------------------------------------------------------------------------------] 100% done ok.
lrj11911808@lrj-virtual-machine:~/CS302_OS/assignment/week12ass$ 
```

3. The too much milk problem.

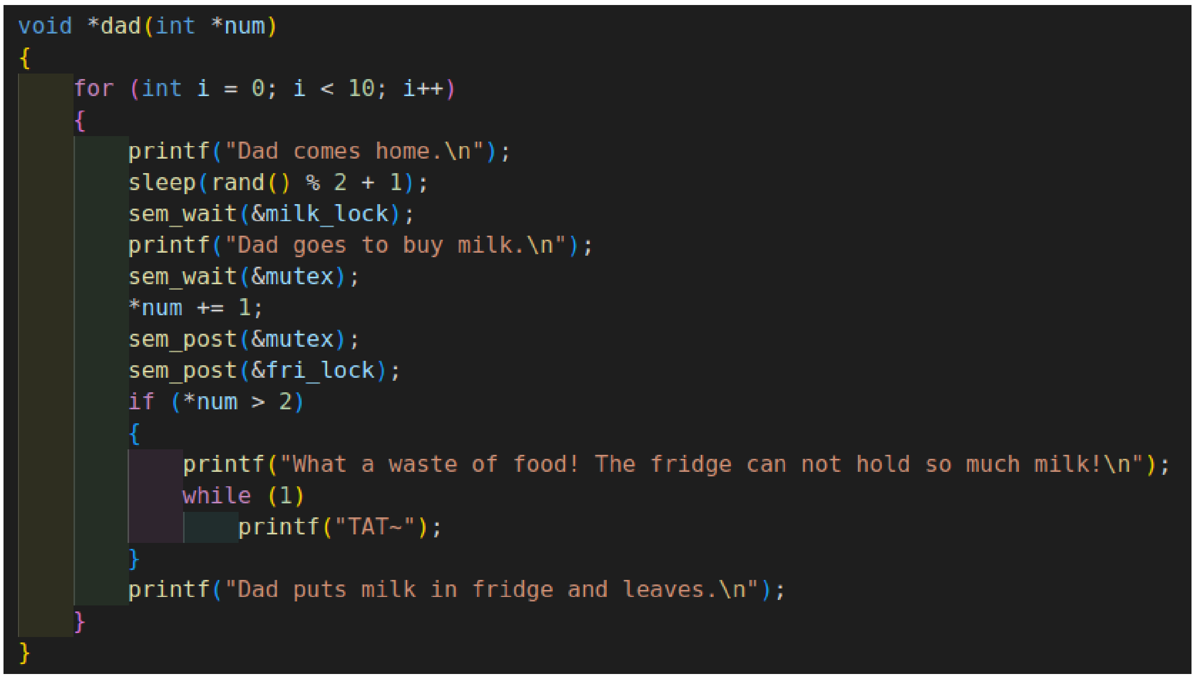
Design idea:

- To solve this problem, we can use three semaphores to prevent deadlock and guarantee the syncronization.
- The first semaphore is named `mutex` which is to guarantee the only unique access to the modification of milk numbers at a time.

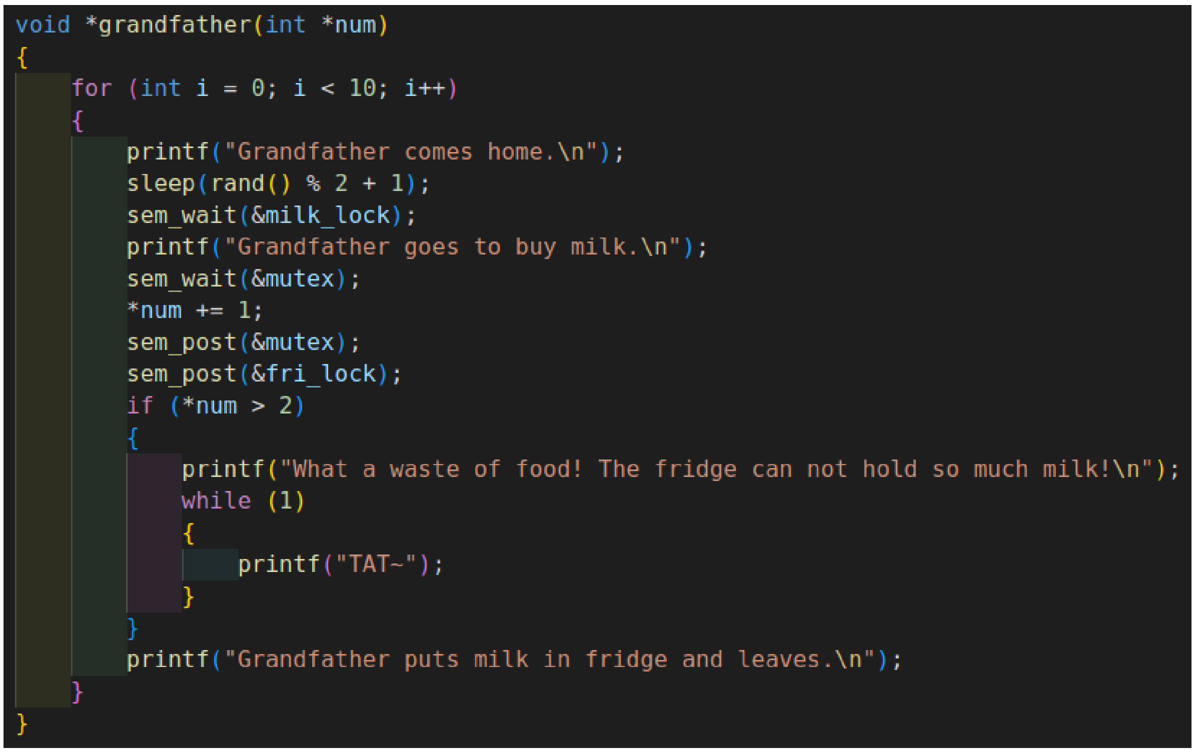
- The second semaphore is named `fri_lock` which is used to indicate how many bottles of milk are currently in the fridge. Every time someone buys a bottle of milk, the `fri_lock` will be posted while son will first wait for a `fri_lock` semaphore before he opens the fridge.
- The third semaphore is name `milk_lock` which is used to indicate how many bottles of milk should the buyers buy. Every time someone wants to buy a bottle of milk, he will first wait for a `milk_lock` semaphore. Also, whenever son fetches a bottle a milk, he will post a `milk_lock` semaphore.



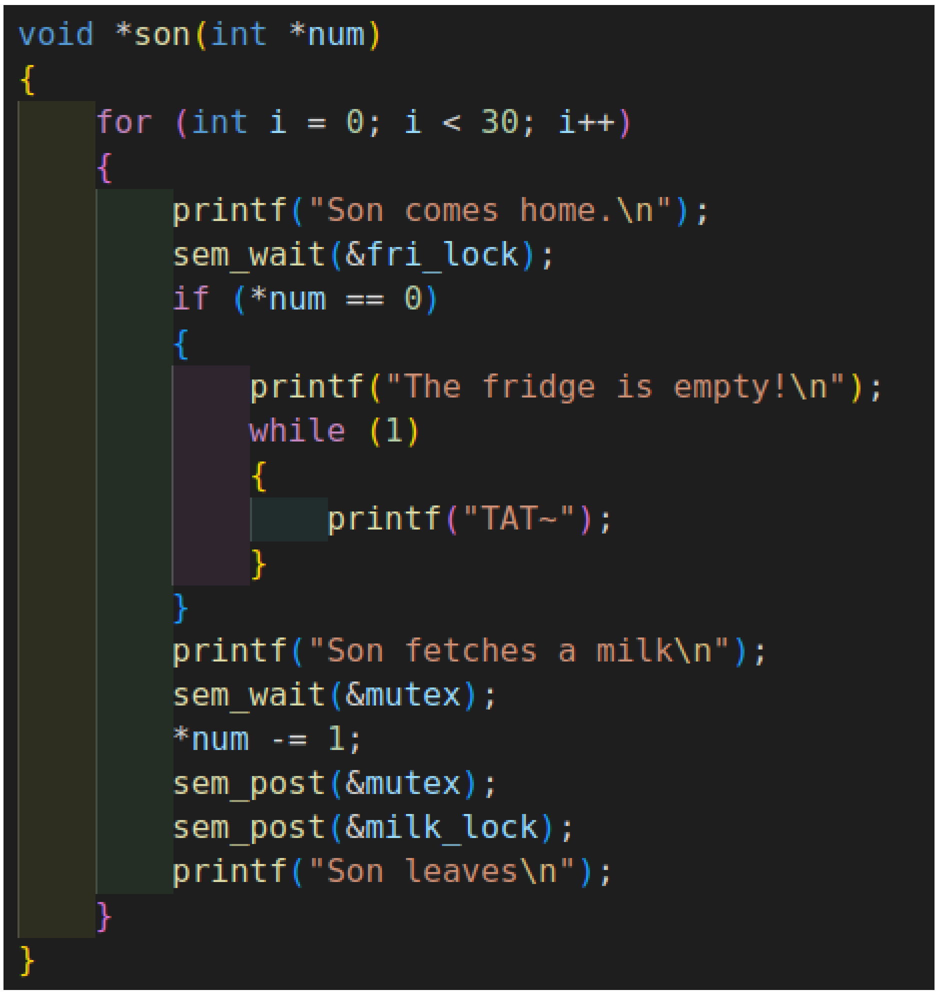
```c
sem_t fri_lock;
sem_t milk_lock;
sem_t mutex;

void *mom(int *num)
{
    for (int i = 0; i < 10; i++)
    {
        printf("Mom comes home.\n");
        sleep(rand() % 2 + 1);
        sem_wait(&milk_lock);
        printf("Mom goes to buy milk.\n");
        sem_wait(&mutex);
        *num += 1;
        sem_post(&mutex);
        sem_post(&fri_lock);
        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");
    }
}
```



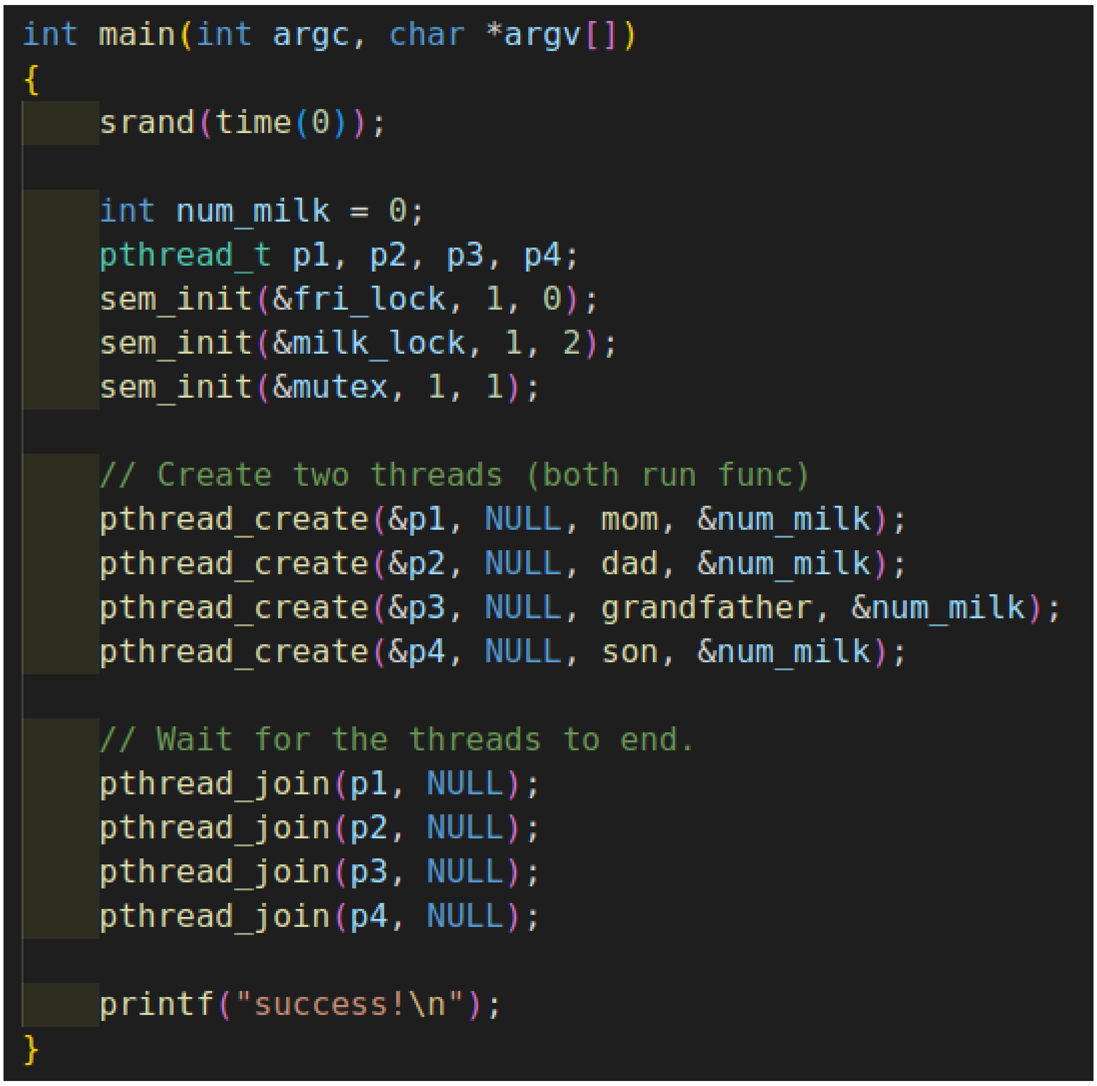
```c
void *dad(int *num)
{
    for (int i = 0; i < 10; i++)
    {
        printf("Dad comes home.\n");
        sleep(rand() % 2 + 1);
        sem_wait(&milk_lock);
        printf("Dad goes to buy milk.\n");
        sem_wait(&mutex);
        *num += 1;
        sem_post(&mutex);
        sem_post(&fri_lock);
        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");
    }
}
```



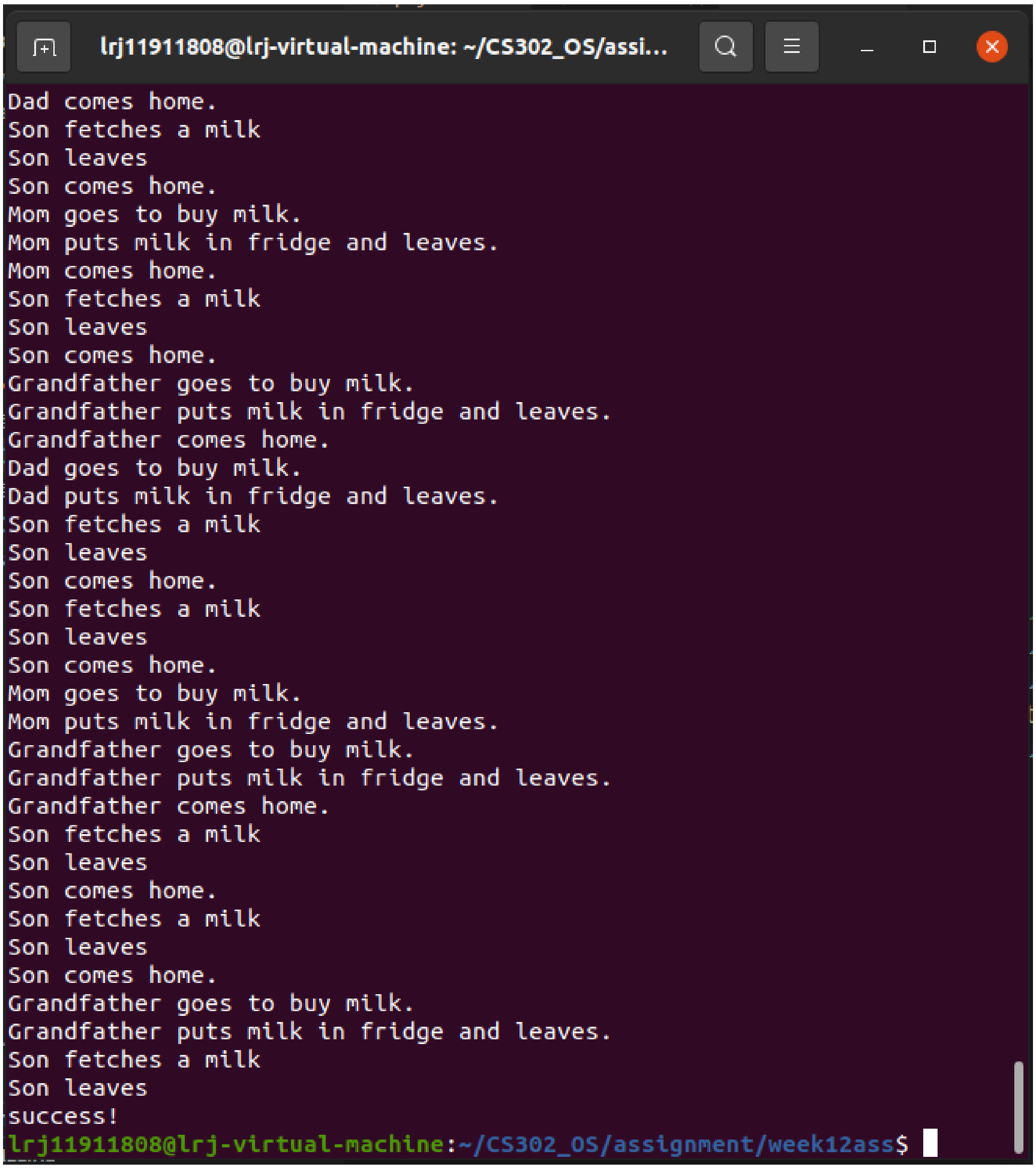
```c
void *grandfather(int *num)
{
    for (int i = 0; i < 10; i++)
    {
        printf("Grandfather comes home.\n");
        sleep(rand() % 2 + 1);
        sem_wait(&milk_lock);
        printf("Grandfather goes to buy milk.\n");
        sem_wait(&mutex);
        *num += 1;
        sem_post(&mutex);
        sem_post(&fri_lock);
        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");
    }
}
```



```c
void *son(int *num)
{
    for (int i = 0; i < 30; i++)
    {
        printf("Son comes home.\n");
        sem_wait(&fri_lock);
        if (*num == 0)
        {
            printf("The fridge is empty!\n");
            while (1)
            {
                printf("TAT~");
            }
        }
        printf("Son fetches a milk\n");
        sem_wait(&mutex);
        *num -= 1;
        sem_post(&mutex);
        sem_post(&milk_lock);
        printf("Son leaves\n");
    }
}
```



```c
int main(int argc, char *argv[])
{
    srand(time(0));

    int num_milk = 0;
    pthread_t p1, p2, p3, p4;
    sem_init(&fri_lock, 1, 0);
    sem_init(&milk_lock, 1, 2);
    sem_init(&mutex, 1, 1);

    // 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");
}
```



lrj11911808@lrj-virtual-machine: ~/CS302_OS/assi...

```
Dad comes home.
Son fetches a milk
Son leaves
Son comes home.
Mom goes to buy milk.
Mom puts milk in fridge and leaves.
Mom comes home.
Son fetches a milk
Son leaves
Son comes home.
Grandfather goes to buy milk.
Grandfather puts milk in fridge and leaves.
Grandfather comes home.
Dad goes to buy milk.
Dad puts milk in fridge and leaves.
Son fetches a milk
Son leaves
Son comes home.
Son fetches a milk
Son leaves
Son comes home.
Mom goes to buy milk.
Mom puts milk in fridge and leaves.
Grandfather goes to buy milk.
Grandfather puts milk in fridge and leaves.
Grandfather comes home.
Son fetches a milk
Son leaves
Son comes home.
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
success!
lrj11911808@lrj-virtual-machine:~/CS302_OS/assignment/week12ass$
```