Exercise 5 : Thread Synchronization

1. ให้บอกว่าโปรแกรมมีข้อผิดพลาดที่ใด และความผิดพลาดเกิดขึ้นจากเหตุใด

```
nrbaboo@ubuntu:~/Desktop/OS/Ex5$ ./nr

 Job 1 started

 Job 2 started

 Job 2 finished

 Job 2 finished
nrbaboo@ubuntu:~/Desktop/OS/Ex5$
```

ตอบ

ข้อผิดพลาด จากผลการรันพบว่า โปรแกรมไม่ได้แสดงผลว่า Job 1 finished แต่แสดง Job 2 finished ออกมา 2 ครั้ง จึงสันนิฐานได้ว่า ตัวแปรที่แสดงจำนวนครั้งของ Job มีการคำนวณที่ผิดพลาด

สาเหตุที่เกิด เนื่องจากใช้ตัวแปร counter เป็นแบบ global เมื่อเข้าไปทำ thread 2 แล้ว ส่งผลให้ตัวแปรมีค่าเปลี่ยนแปลงไป เมื่อจบการทำงานของ thread 1 จึงทำให้แสดงผลเป็นค่าล่าสุดออกมา

2. แก้ปัญหาที่เกิดขึ้น แสดงโปรแกรมและผลการทำงานที่แก้ไขแล้ว

```c
pthread_t thread_id[2];
int counter;
pthread_mutex_t lock;

void* doThing(void *arg)
{
        pthread_mutex_lock(&lock);

        unsigned long i = 0;
        counter += 1;
        printf("\n Job %d started\n", counter);
        for(i=0; i<(0xFFFFFFFF);i++);
        printf("\n Job %d finished\n", counter);

        pthread_mutex_unlock(&lock);

        return NULL;
}
```

```c
int main(void)
{
        int i = 0;
        int err;
        if (pthread_mutex_init(&lock, NULL) != 0)
        {
                printf("\n mutex init failed\n");
                return 1;
        }
        while(i < 2)
        {
                err = pthread_create(&(thread_id[i]), NULL, &doThing, NULL);
                if (err != 0)
                        printf("\ncan't create thread :[%s]", strerror(err));
                i++;
        }
        pthread_join(thread_id[0], NULL);
        pthread_join(thread_id[1], NULL);
        pthread_mutex_destroy(&lock);
        return 0;
}
```

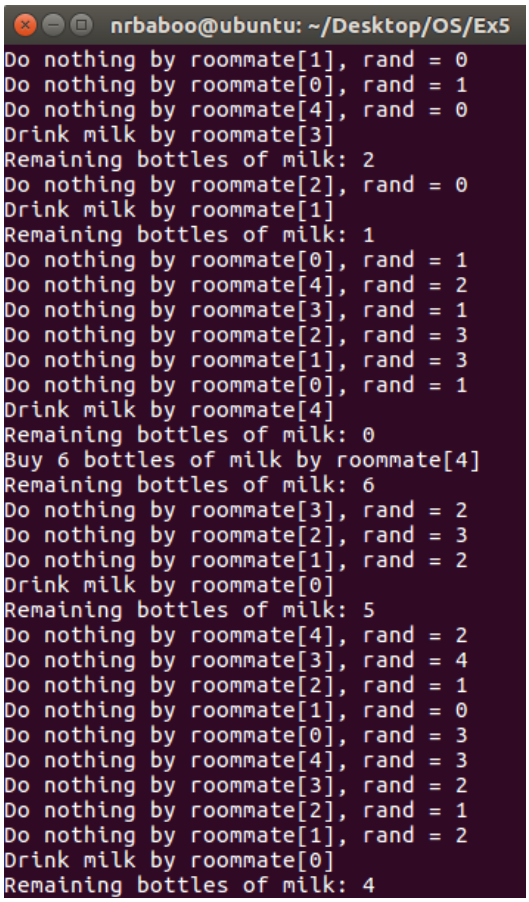
```
nrbaboo@ubuntu:~/Desktop/OS/Ex5$ ./nr

 Job 1 started

 Job 1 finished

 Job 2 started

 Job 2 finished
```

3. ให้แก้ไขโปรแกรมตามข้อ 1 โดยใช้ semaphore แสดงโปรแกรมและผลการทำงานที่แก้ไขแล้ว

```c
sem_t s;
void* doThing(void *arg)
{
        //pthread_mutex_lock(&lock);
        sem_wait(&s);

        unsigned long i = 0;
        counter += 1;
        printf("\n Job %d started\n", counter);
        for(i=0; i<(0xFFFFFFFF);i++);
        printf("\n Job %d finished\n", counter);

        //pthread_mutex_unlock(&lock);
        sem_post(&s);

        return NULL;
}
int main(void)
{
        int i = 0;
        int err;
        if ( sem_init(&s, 0, 1) != 0 )
        {
                // Error: initialization failed
        }
        while(i < 2)
        {
                err = pthread_create(&(thread_id[i]), NULL, &doThing, NULL);
                if (err != 0)
                        printf("\ncan't create thread :[%s]", strerror(err));
                i++;
        }

        pthread_join(thread_id[0], NULL);
        pthread_join(thread_id[1], NULL);
        //pthread_mutex_destroy(&lock);
        return 0;
```

```
nrbaboo@ubuntu:~/Desktop/OS/Ex5$ ./nr

 Job 1 started

 Job 1 finished

 Job 2 started

 Job 2 finished
```

4.




```c
void* drinkMilkAndBuyIfNeed(void* id)
{
        const int myid = (long)id;

        while(1) {
                pthread_mutex_lock( &mutex );
                r = rand() % 5;
                if(r == myid) {
                        milk--;
                        printf("Drink milk by roommate[%d]\n", myid);

                        printf("Remaining bottles of milk: %d\n", milk);
                        sleep(1);

                        if (milk == 0) {
                                r = rand() % 5 +2;
                                milk += r;
                                printf("Buy %d bottles of milk by roommate[%d]\n",r,myid);
                                printf("Remaining bottles of milk: %d\n", milk);
                                sleep(1);
                        }
                }
                else {
                        printf("Do nothing by roommate[%d], rand = %d\n", myid, r);
                        sleep(1);
                }
                pthread_mutex_unlock( & mutex );
                sleep(1);
        }
}
```

แนวคิดในการออกแบบโปรแกรม

ใช้ความรู้เรื่อง lock และ unlock ในข้อ 1.2 เพื่อทำให้แต่ละ thread ทำงานโดยรอ thread ก่อนหน้าก่อนช่วยในการคำนวณจำนวนนมที่เหลือได้ง่ายขึ้น

เมื่อเข้าไปใน thread จะทำการ random เลข 0-4 ถ้าเลขตรงกับ thread นั้นจะทำการดื่มนม (คิดเป็น 0.2%)

ถ้าหาก thread นั้นดื่มนมจนเหลือ 0 จะทำการไปซื้อนมมาใหม่ โดยสุ่มซื้อ ระหว่าง 2-6 ขวด