班级：2013211312　　姓名：楼燕　　学号2012211451

实验3 进程同步

实验要求：

见书p236

代码说明：

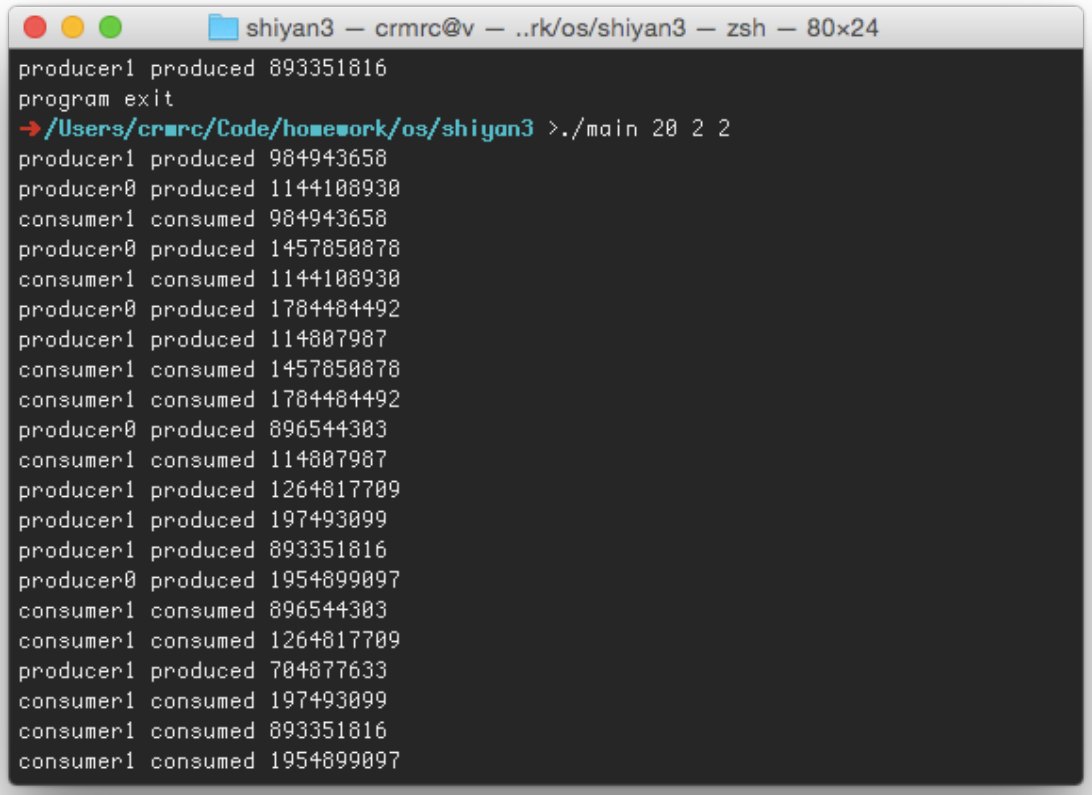
buffer.h 文件里带一个全局的 buffer变量声明 和两个对buffer变量进行操作的接口函数声明：

```
int insert_item(buffer_item item);
int remove_item(buffer_item* item);
```

buffer.c 是对接口和全局变量的实现和定义。

main.c 是主程序，基本跟书上内容没差, osx不能用sem_init就只能用sem_open代替了。

效果演示：



代码：

```
//buffer.h

#ifndef __BUFFER_H__
#define __BUFFER_H__
```

班级：2013211312 姓名：楼燕 学号2012211451

```
#include <unistd.h>

#define BUFFER_SIZE 5

typedef int buffer_item;
typedef int buffer_idx;
typedef struct {
    buffer_item buf[BUFFER_SIZE];
    size_t len;
    buffer_idx head;
    buffer_idx end;
} buffer_t;

extern buffer_t buffer;

int insert_item(buffer_item item);
int remove_item(buffer_item* item);

#endif


//buffer.c

#include "buffer.h"

buffer_t buffer = {
    .len = 0,
    .head = 0,
    .end = 0
};

int insert_item(buffer_item item)
{
    if (buffer.len >= BUFFER_SIZE){
        return -1;
    }else{
        buffer.len++;
        buffer.buf[buffer.end] = item;
        buffer.end = (buffer.end + 1) % BUFFER_SIZE;
        return 0;
    }

    return -1;
}

int remove_item(buffer_item* item)
{
    if (buffer.len <= 0){
        return -1;
    }else{
        buffer.len--;
        (*item) = buffer.buf[buffer.head];
        buffer.head = (buffer.head + 1) % BUFFER_SIZE;
        return 0;
    }
    return -1;
```

```
}



//main.c

#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include "buffer.h"

void *producer(void* pnumber);
void *consumer(void* pnumber);

pthread_mutex_t mutex;
sem_t *empty, *full;

const char *empty_name = "Empty!";
const char *full_name = "Full!"; //Die!!


int main(int argc, char const *argv[])
{
    // wtf,osx sem_init 不能用
    // sem_init(&empty, 0, 5);
    // sem_init(&full, 0, 0);

    int i;
    int sleep_time = atoi(argv[1]);
    int pro_thread_count = atoi(argv[2]);
    int con_thread_count = atoi(argv[3]);

    if ((empty = sem_open(empty_name, O_CREAT, 0644, 5)) == SEM_FAILED) {
        perror("semaphore initilization");
        exit(1);
    }

    if ((full = sem_open(full_name, O_CREAT, 0644, 0)) == SEM_FAILED) {
        perror("semaphore initilization");
        exit(1);
    }

    pthread_t pro, con;
    int *pnumber = malloc(sizeof(int));

    for (i = 0; i < pro_thread_count; ++i)
    {
        *pnumber = i;
        pthread_create(&pro, NULL, producer, (void*)pnumber);
    }

    for (i = 0; i < con_thread_count; ++i)
    {
        *pnumber = i;
        pthread_create(&con, NULL, consumer, (void*)pnumber);
```

```
    }



    // pthread_join(pro, NULL);
    // pthread_join(con, NULL);

    sleep(sleep_time);
    pthread_mutex_destroy(&mutex);

    // sem_destroy(&empty);
    // sem_destroy(&full);

    sem_unlink(empty_name);
    sem_unlink(full_name);

    printf("program exit\n");


    return 0;
}

void *producer(void* pnumber)
{
    buffer_item item;
    int number = *((int*)pnumber);

    while(1) {
        sleep(rand() % 3);
        item = rand();

        sem_wait(empty);
        pthread_mutex_lock(&mutex);

        printf("producer%d produced %d\n", number, item);
        if (insert_item(item)){
            fprintf(stderr, "report error condition");
        }

        pthread_mutex_unlock(&mutex);
        sem_post(full);
    }
}

void *consumer(void* pnumber)
{
    buffer_item item;
    int number = *((int*)pnumber);


    while(1) {
        sleep(rand() % 3);
        item = rand();

        sem_wait(full);
        pthread_mutex_lock(&mutex);
```

```
        if (remove_item(&item)){
            fprintf(stderr, "report error condition");
        }else{
            printf("consumer%d consumed %d\n", number, item);
        }

        pthread_mutex_unlock(&mutex);
        sem_post(empty);
    }
}
```