**操作系统实验1**

1. **实验目的**

分别在Linux和Windows系统下编写生产者-消费者问题的程序，来掌握多线程、信号量、互斥锁、缓冲区等的应用。

1. **实验过程**
2. Linux系统下

编写程序buffer.h：

定义数据类型buffer\_item（int型），定义缓冲区大小BUFFER\_SIZE，定义缓冲区buffer（循环队列），定义缓冲区头尾指针front、rear。

定义遍历打印缓冲区元素的函数print\_buffer()。

定义向缓冲区插入元素的函数insert\_item(buffer\_item item)：若队列未满，则向队尾插入元素，尾指针后移一位。

定义从缓冲区删除元素的函数remove\_item(buffer\_item\* item)：若队列未空，则头指针后移一位。

/\* buffer.h \*/

typedef int buffer\_item;

#define BUFFER\_SIZE 5

/\* the buffer \*/

buffer\_item buffer[BUFFER\_SIZE];

/\* pointers of the buffer \*/

int front = 0, rear = 0;

void print\_buffer()

{

    printf("buffer: ");

    int i = 0;

    if (front == rear)

        printf("empty");

    else

        for (i = front; i != rear; i = (i + 1) % BUFFER\_SIZE)

            printf("%d\t", buffer[i]);

    printf("\n\n");

}

int insert\_item(buffer\_item item)

{

    /\* if the queue is full, return -1 \*/

    if((rear + 1) % BUFFER\_SIZE == front)

        return -1;

    /\* insert item into buffer, return 0 \*/

    buffer[rear] = item;

    rear = (rear + 1) % BUFFER\_SIZE;

    return 0;

}

int remove\_item(buffer\_item\* item)

{

    /\* if the queue is empty, return -1 \*/

    if (front == rear)

        return -1;

    /\* remove an object from buffer and place it in item, return 0 \*/

    \*item = buffer[front];

    front = (front + 1) % BUFFER\_SIZE;

    return 0;

}

编写程序main.c：

定义信号量empty、full，定义互斥锁mutex。

定义生产者线程入口函数：随机睡眠时间（0-5s）后，等待信号量和互斥锁开始生产（产生0-9随机数并插入缓冲区队尾），生产结束后释放信号量和互斥锁，打印缓冲区内容。

定义消费者线程入口函数：随机睡眠时间（0-5s）后，等待信号量和互斥锁开始消费（从缓冲区队首删除元素），消费结束后释放信号量和互斥锁，打印缓冲区内容。

从控制台传入4个参数：argv[1]为主程序在终止前睡眠的时间，argv[2]为生产者数量，argv[3]为消费者数量。

按照数量定义线程标识符、线程返回值（使用malloc动态分配）。

初始化缓冲区。

按照数量创建生产者、消费者线程。

主程序睡眠指定的时间后结束运行。

/\* main.c \*/

#include <pthread.h>

#include <semaphore.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <unistd.h>

#include <stdbool.h>

#include <malloc.h>

#include "buffer.h"

int p\_id, c\_id;

sem\_t empty, full;

pthread\_mutex\_t mutex;

/\* threads call these functions \*/

void\* producer();

void\* consumer();

int main(int argc, char \*argv[])

{

    srand(time(0));

    int i = 0;

    /\* get command line arguments \*/

    int sleep\_time = atoi(argv[1]);

    int p\_num = atoi(argv[2]);

    int c\_num = atoi(argv[3]);

    if (argc != 4)

    {

        fprintf(stderr, "usage: a.out <integer value>\n");

        return -1;

    }

    if (sleep\_time < 0 || p\_num < 0 || c\_num < 0)

    {

        fprintf(stderr, "all 3 arguments must be >= 0\n");

        return -1;

    }

    printf("sleep time: %d, producer number: %d, consumer number: %d\n\n", sleep\_time, p\_num, c\_num);

    /\* threads \*/

    int mutex\_lock = pthread\_mutex\_init(&mutex, NULL);

    pthread\_t\* p = (pthread\_t\*)malloc(sizeof(pthread\_t) \* p\_num);

    pthread\_t\* c = (pthread\_t\*)malloc(sizeof(pthread\_t) \* c\_num);

    /\* return value of threads \*/

    int\* p\_ret = (int\*)malloc(sizeof(int) \* p\_num);

    int\* c\_ret = (int\*)malloc(sizeof(int) \* c\_num);

    /\* initialize buffer \*/

    buffer\_item temp = 0;

    while (insert\_item(temp) != -1);

    buffer[rear] = 0;

    rear = (rear + 1) % BUFFER\_SIZE;

    printf("initial ");

    print\_buffer();

    /\* create producer thread(s) \*/

    for (i = 0; i < p\_num; ++i)

    {

        p\_ret[i] = pthread\_create(&p[i], NULL, producer, NULL);

        if (p\_ret[i] != 0)

        {

            printf("failed to create producer\_%d\n", i);

            exit(1);

        }

    }

    /\* create consumer thread(s) \*/

    for (i = 0; i < c\_num; ++i)

    {

        c\_ret[i] = pthread\_create(&c[i], NULL, consumer, NULL);

        if (c\_ret[i] != 0)

        {

            printf("failed to create consumer\_%d\n", i);

            exit(1);

        }

    }

    /\* sleep \*/

    sleep(sleep\_time);

    /\* exit \*/

    return 0;

}

void \*producer()

{

    int id = ++p\_id;

    int ret = sem\_init(&empty, 0, BUFFER\_SIZE);

    while (true)

    {

        /\* sleep for a random period of time \*/

        int time = rand() % 5;

        sleep(time);

        /\* wait \*/

        sem\_wait(&empty);

        sem\_getvalue(&empty, &ret);

        pthread\_mutex\_lock(&mutex);

        /\* generate a random number \*/

        buffer\_item item = rand() % 10;

        /\* insert to buffer \*/

        if (insert\_item(item) == -1)

            fprintf(stderr, "failed to insert\n");

        else

            printf("producer\_%d produced %d after sleeping %d\n", id, item, time);

        /\* signal \*/

        pthread\_mutex\_unlock(&mutex);

        sem\_post(&full);

        /\* print buffer \*/

        print\_buffer();

    }

}

void \*consumer()

{

    int id = ++c\_id;

    int ret = sem\_init(&empty, 0, BUFFER\_SIZE);

    while (true)

    {

        /\* sleep for a random period of time \*/

        int time = rand() % 5;

        sleep(time);

        /\* wait \*/

        sem\_wait(&full);

        sem\_getvalue(&empty, &ret);

        pthread\_mutex\_lock(&mutex);

        /\* remove from buffer \*/

        buffer\_item item = 0;

        if (remove\_item(&item) == -1)

            fprintf(stderr, "failed to remove\n");

        else

            printf("consumer\_%d consumed %d after sleeping %d\n", id, item, time);

        /\* signal \*/

        pthread\_mutex\_unlock(&mutex);

        sem\_post(&empty);

        /\* print buffer \*/

        print\_buffer();

    }

}

1. Windows系统下

buffer.h同Linux系统下，略。

main.c与Linux系统下有不同。

头文件去除<pthread.h>、<semaphore.h>，添加<windows.h>。

信号量、互斥锁的类型为HANDLE，线程标识符的类型为DWORD，线程入口函数返回值类型为DWORD WINAPI。

互斥锁的创建为CreateMutex()，信号量的创建为CreateSemaphore()。

信号量、互斥锁的等待为WaitForSingleObject()，互斥锁的释放为ReleaseMutex()，信号量的释放为ReleaseSemaphore()。

与Linux系统下不同，创建线程的返回值不为0时是创建成功。

睡眠函数sleep()改为Sleep()，并注意参数的单位是毫秒。

/\* main.c \*/

#include <windows.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <stdbool.h>

#include <malloc.h>

#include "buffer.h"

int p\_id, c\_id;

HANDLE empty, full;

HANDLE mutex;

/\* threads call these functions \*/

DWORD WINAPI producer();

DWORD WINAPI consumer();

int main(int argc, char \*argv[])

{

    srand(time(0));

    int i = 0;

    /\* get command line arguments \*/

    int sleep\_time = atoi(argv[1]);

    int p\_num = atoi(argv[2]);

    int c\_num = atoi(argv[3]);

    if (argc != 4)

    {

        fprintf(stderr, "usage: a.out <integer value>\n");

        return -1;

    }

    if (sleep\_time < 0 || p\_num < 0 || c\_num < 0)

    {

        fprintf(stderr, "all 3 arguments must be >= 0\n");

        return -1;

    }

    printf("sleep time: %d, producer number: %d, consumer number: %d\n\n", sleep\_time, p\_num, c\_num);

    /\* create mutex and semaphores \*/

    mutex = CreateMutex(NULL, FALSE, NULL);

    empty = CreateSemaphore(NULL, 5, 5, NULL);

    full = CreateSemaphore(NULL, 0, 5, NULL);

    /\* threads \*/

    DWORD \*p = (DWORD \*)malloc(sizeof(DWORD) \* p\_num);

    DWORD \*c = (DWORD \*)malloc(sizeof(DWORD) \* c\_num);

    /\* return value of threads \*/

    HANDLE \*p\_ret = (HANDLE)malloc(sizeof(HANDLE) \* p\_num);

    HANDLE \*c\_ret = (HANDLE)malloc(sizeof(HANDLE) \* c\_num);

    /\* initialize buffer \*/

    buffer\_item temp = 0;

    while (insert\_item(temp) != -1);

    buffer[rear] = 0;

    rear = (rear + 1) % BUFFER\_SIZE;

    printf("initial ");

    print\_buffer();

    /\* create producer thread(s) \*/

    for (i = 0; i < p\_num; ++i)

    {

        p\_ret[i] = CreateThread(NULL, 0, producer, NULL, 0, &p[i]);

        if (p\_ret[i] == 0)

        {

            printf("failed to create producer\_%d\n", i);

            exit(1);

        }

        CloseHandle(p\_ret[i]);

        Sleep(100);

    }

    /\* create consumer thread(s) \*/

    for (i = 0; i < c\_num; ++i)

    {

        c\_ret[i] = CreateThread(NULL, 0, consumer, NULL, 0, &c[i]);

        if (c\_ret[i] == 0)

        {

            printf("failed to create consumer\_%d\n", i);

            exit(1);

        }

        CloseHandle(c\_ret[i]);

        Sleep(100);

    }

    /\* sleep \*/

    Sleep(sleep\_time \* 1000);

    /\* exit \*/

    return 0;

}

DWORD WINAPI producer()

{

    int id = ++p\_id;

    while (true)

    {

        /\* sleep for a random period of time \*/

        int time = rand() % 5;

        Sleep(time \* 1000);

        /\* wait \*/

        WaitForSingleObject(empty, INFINITE);

        WaitForSingleObject(mutex, INFINITE);

        /\* generate a random number \*/

        buffer\_item item = rand() % 10;

        /\* insert to buffer \*/

        if (insert\_item(item) == -1)

            fprintf(stderr, "failed to insert\n");

        else

            printf("producer\_%d produced %d after sleeping %d\n", id, item, time);

        /\* signal \*/

        ReleaseMutex(mutex);

        ReleaseSemaphore(full, 1, NULL);

        /\* print buffer \*/

        print\_buffer();

    }

}

DWORD WINAPI consumer()

{

    int id = ++c\_id;

    while (true)

    {

        /\* sleep for a random period of time \*/

        int time = rand() % 5;

        Sleep(time \* 1000);

        /\* wait \*/

        WaitForSingleObject(full, INFINITE);

        WaitForSingleObject(mutex, INFINITE);

        /\* remove from buffer \*/

        buffer\_item item = 0;

        if (remove\_item(&item) == -1)

            fprintf(stderr, "failed to remove\n");

        else

            printf("consumer\_%d consumed %d after sleeping %d\n", id, item, time);

        /\* signal \*/

        ReleaseMutex(mutex);

        ReleaseSemaphore(empty, 1, NULL);

        /\* print buffer \*/

        print\_buffer();

    }

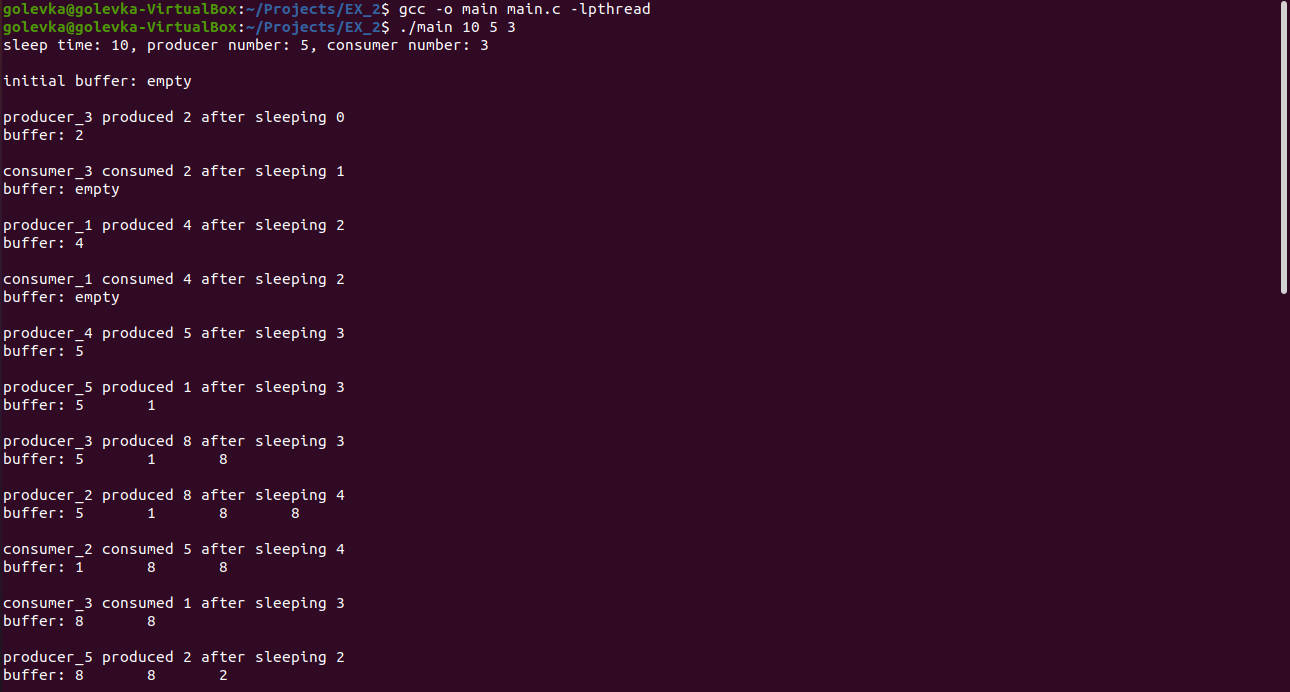
}

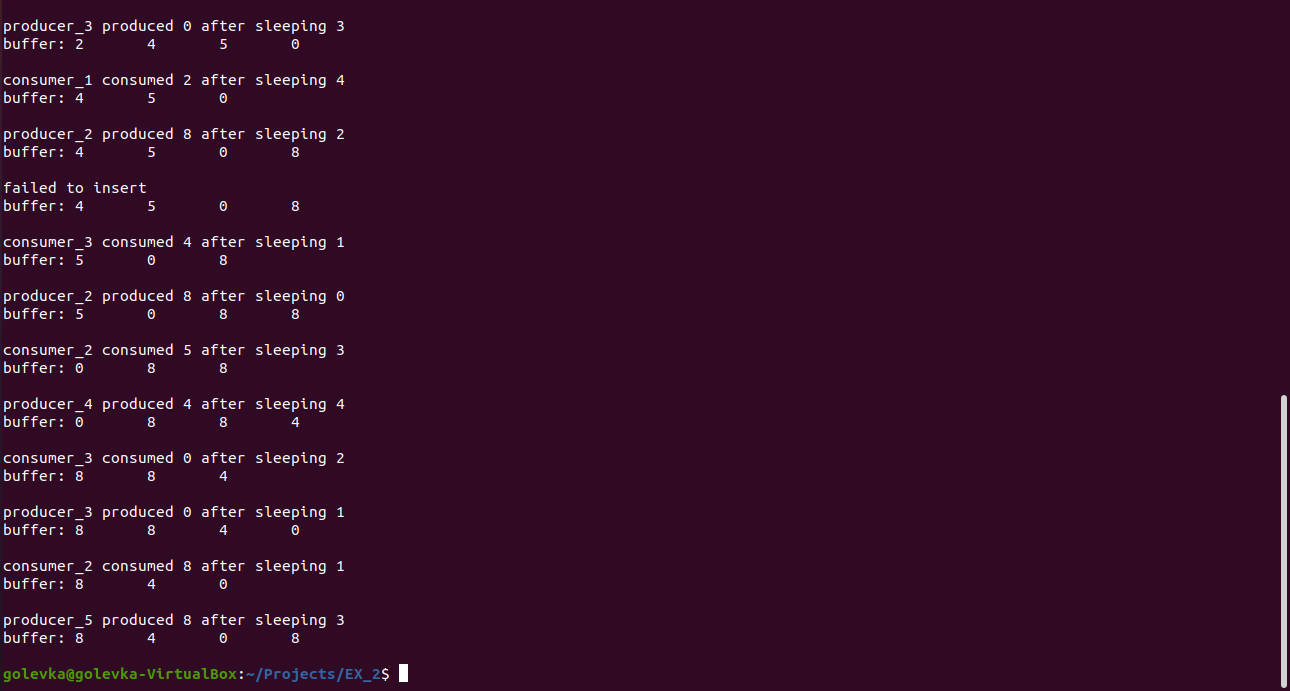
1. **实验结果**
2. Linux系统下

在控制台输入gcc -o main main.c -lpthread，编译生成main文件。

在控制台输入./main 10 5 3，设定主程序运行时间为10s，有5个生产者，3个消费者。

运行结果如下，可看到不同的生产者、消费者线程的生产、消费情况和缓冲区状态，程序在运行10s后结束：





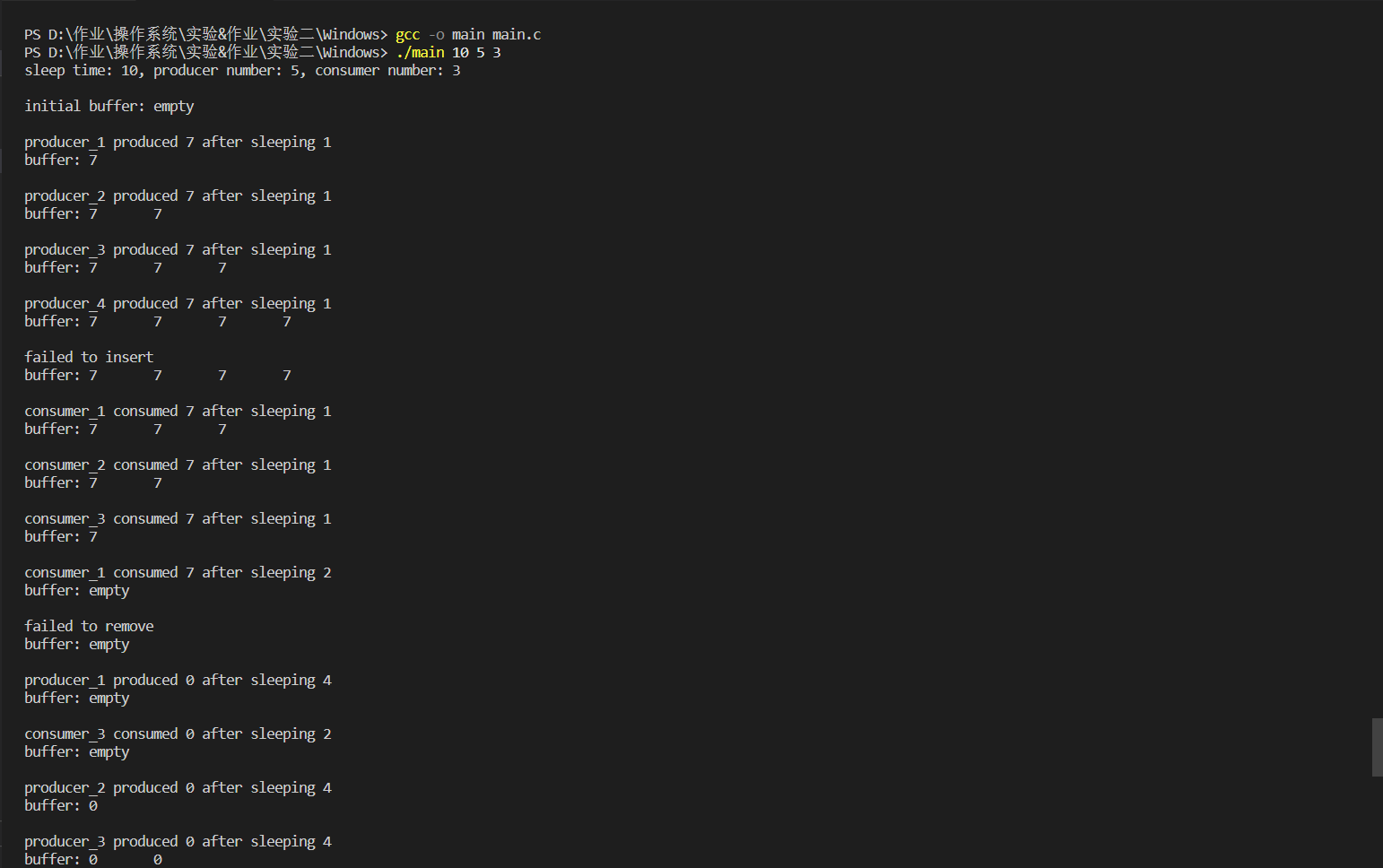
1. Windows系统下

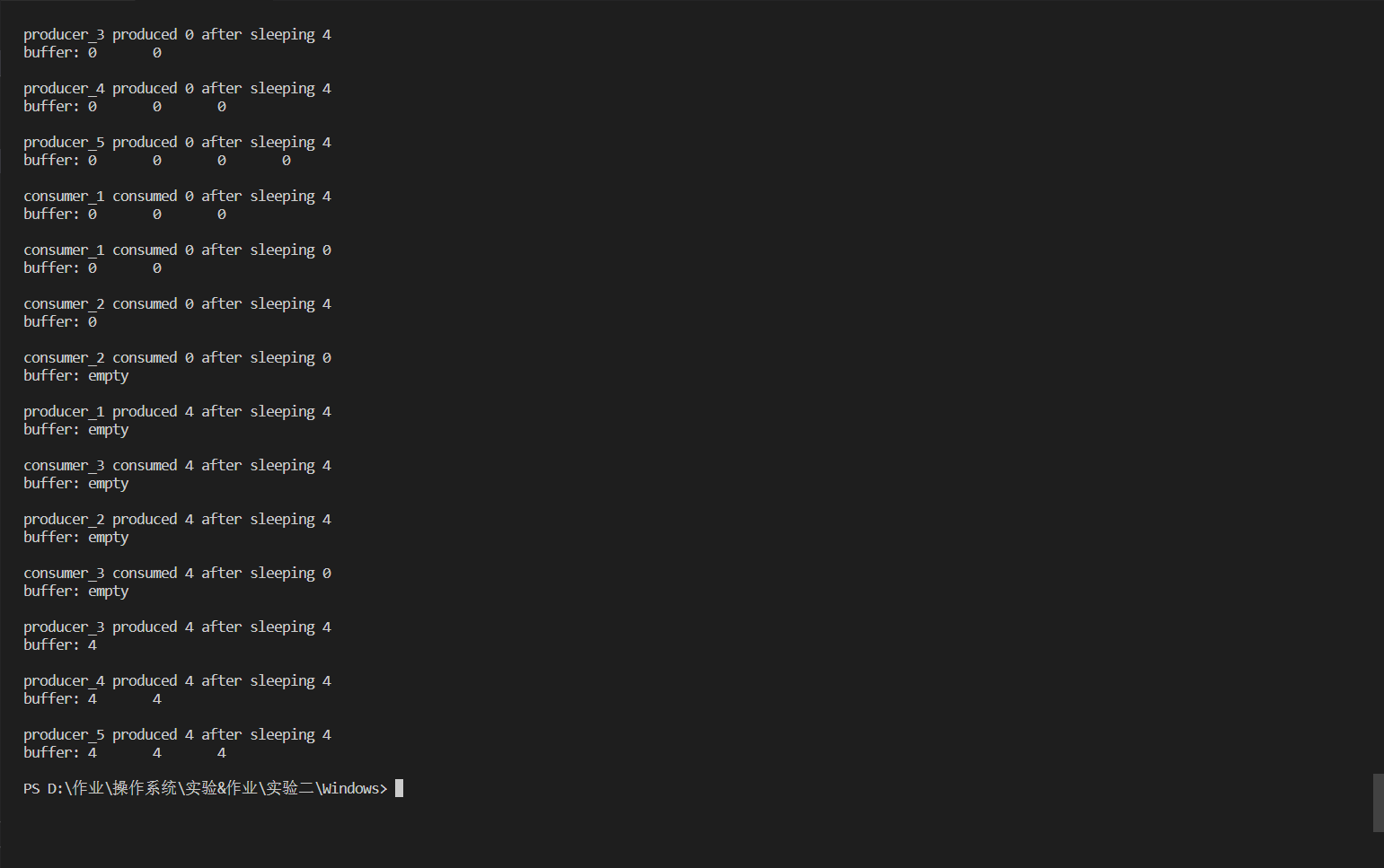
在控制台输入gcc -o main main.c，编译生成main.exe文件。

在控制台输入./main 10 5 3，设定主程序运行时间为10s，有5个生产者，3个消费者。

运行结果如下，可看到不同的生产者、消费者线程的生产、消费情况和缓冲区状态，程序在运行10s后结束。

目前的问题是，不同的生产者线程在产生随机数时，总是产生相同的结果，多次运行结果相同，但随机数种子是没有问题的，在主函数中打印随机数也正常，还没有找到问题在哪。





1. **实验小结**

通过本实验，可以更加深入的了解到信号量、互斥锁在多线程任务中的用途，以及深入了解多线程任务中不同线程的运行情况，对pthread、semaphore中的一些函数、类型更加熟悉。