电子科技大学信息与软件工程学院

**实 验 报 告**

学 号 2023091602014

姓 名 张烨涛

（实验） 课程名称 操作系统原理与实践

理论教师 刘瑶

实验教师 刘瑶

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：张烨涛 学号：2023091602014 指导教师：刘瑶**

**实验地点：信软楼303 实验时间：2025.04.28**

1. **实验名称：**

实验一：生产者消费者问题的实现

1. **实验学时：**

3h

1. **实验目的：**
2. 掌握进程、线程的概念，熟悉相关的控制语。
3. 掌握进程、线程间的同步原理和方法。
4. 掌握进程、线程间的互斥原理和方法。
5. 掌握使用信号量原语解决进程、线程间互斥和同步方法。
6. **实验原理：**

n个缓冲区的缓冲池作为一个临界资源：

· 当生产者任务从数据源—文件中读取数据后将会申请一个缓冲区，并将此数据放入缓冲区中。

· 消费者任务从一个缓冲区中取走数据，并将其中的内容打印输出。

当一个生产者任务正在访问缓冲区时，其他生产者和消费者任务不能访问缓冲区。

当一个消费者任务正在访问缓冲区时，其他其他生产者和消费者任务不能访问缓冲区（使用互斥量实现对缓冲池的互斥访问）。

生产者任务在向缓冲池放入数据之前需要判断缓冲池中是否还有空的缓冲区，如果有则向空的缓冲区写入，如果没有则等待。

消费者任务在从缓冲池读取数据之前需要判断缓冲池中是否有已经写入数据的缓冲区，如果有则读取已经写入数据的缓冲区，如果没有则等待（使用两个信号量，一个代表空缓冲区的数量，一个代表已经写入数据的缓冲区数量）。

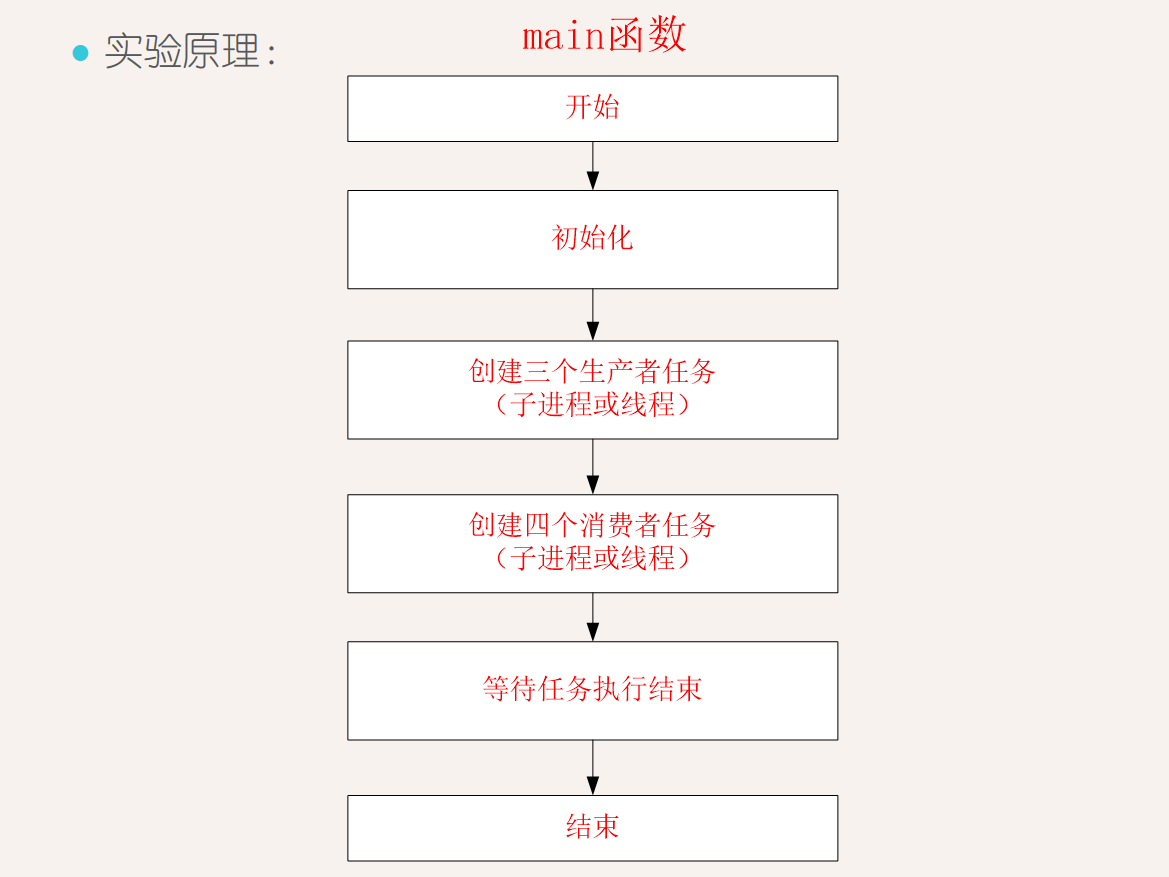


图 4-1 main函数流程

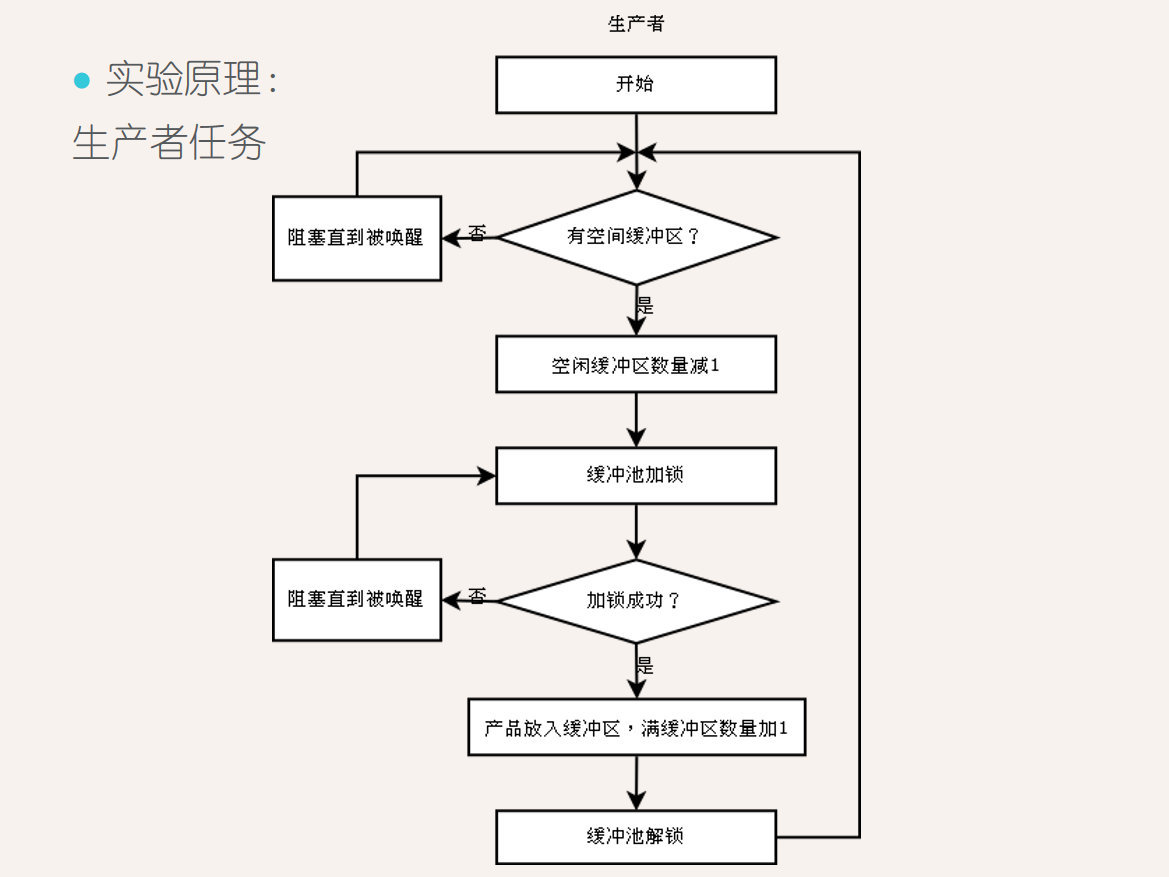


图 4-2 生产者流程

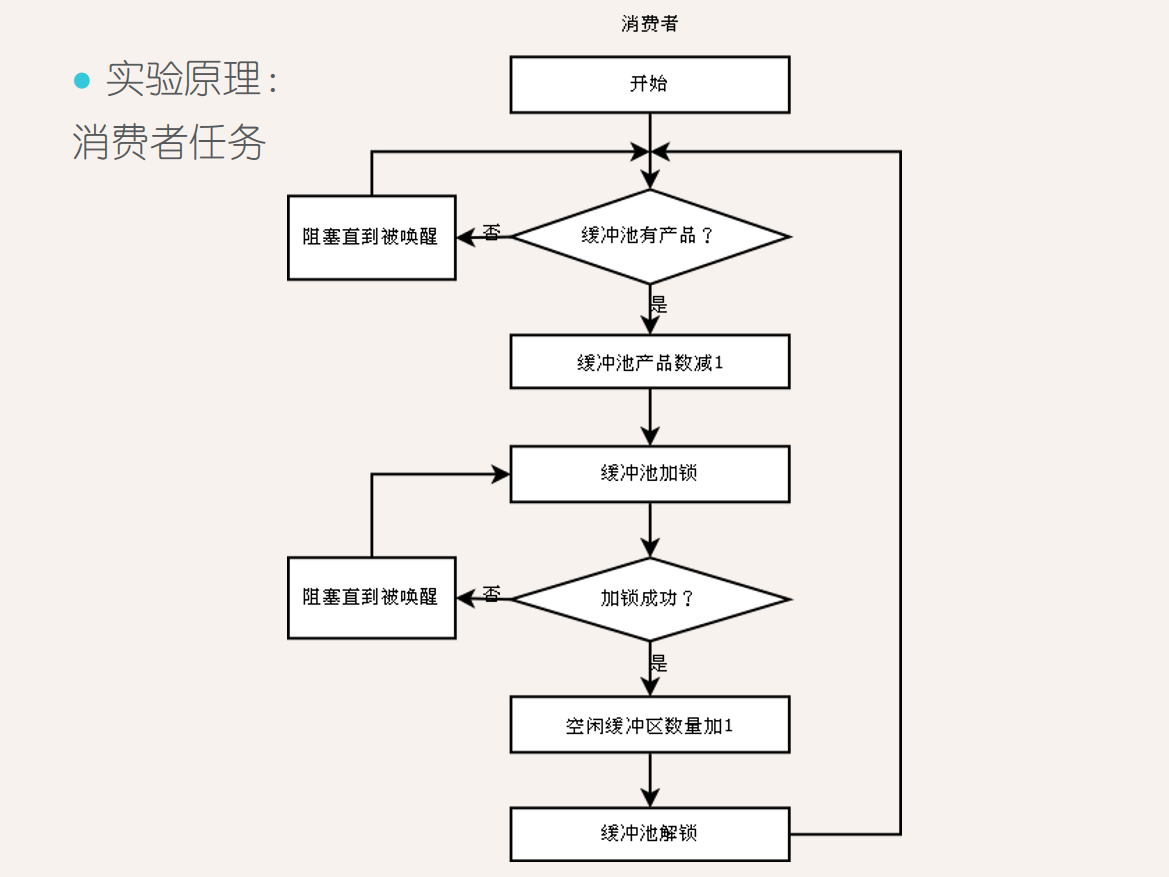


图 4-3 消费者流程

1. **实验内容：**
2. 有一群生产者任务在生产产品，并将这些产品提供给消费者任务去消费。为使生产者任务与消费者进程能并发执行，在两者之间设置了一个具有n个缓冲区的缓冲池：

· 生产者任务从文件中读取一个数据，并将它存放到一个缓冲区中

· 消费者任务从一个缓冲区中取走数据，并输出此数据（printf）

· 生产者和消费者之间必须保持同步原则：不允许消费者任务到一个空缓冲区去取产品；也不允许生产者任务向一个已装满产品且尚未被取走的缓冲区中投放产品。

1. 创建3个进程（或者线程）作为生产者任务，4个进程（或者线程）作为消费者任务
2. 创建一个文件作为数据源，文件中事先写入一些内容作为数据（字符串或者数值）
3. 生产者和消费者任务（进程或者线程）都具有相同的优先级
4. **实验器材（设备、元器件）：**

· 学生每人一台PC，安装WindowsXP/2000操作系统。

· 局域网络环境。

· 个人PC安装VMware虚拟机和Ubuntu系统。

1. **实验步骤：**
2. 分配具有 n 个缓冲区的缓冲池，作为共享资源。
3. 定义两个资源型信号量 empty 和 full，empty 信号量表示当前空的缓冲区数量，full 表示当前满的缓冲区数量。
4. 定义互斥信号量 mutex，当某个进程访问缓冲区之前先获取此信号量，在对缓冲区的操作完成后再释放此互斥信号量。以此实现多个进程对共享资源的互斥访问。
5. 创建 3 个进程（或者线程）作为生产者，4 个进程（或者线程）作为消费者。创建一个文件作为数据源，文件中事先写入一些内容作为内容。
6. 编写代码实现生产者进程的工作内容，即从文件中读取数据，然后申请一个 empty 信号量，和互斥信号量，然后进入临界区操作将读取的数据放入此缓冲区中。并释放 empty 信号量和互斥信号量。
7. 编写代码实现消费者者进程的工作内容，即先申请一个 full 信号量，和互斥信号量，然后进入临界区操作从缓冲区中读取数据并打印输出。
8. **实验结果与分析（含重要数据结果分析或核心代码流程分析）**

基于生产者消费者问题模型，我们进行多线程程序编程，实现了从文件中读取数据，放入缓冲区中，再从缓冲区中取出数据进行消费的过程。其中，涉及到了多线程之间共享变量，因此使用信号量和互斥锁来保证线程安全。程序主要逻辑梳理如下：

1. 定义缓冲区：定义了 in、out 和 buff 三个全局变量。其中，in 表示下一个生产者要放置产品的位置，out 表示下一个消费者需要访问的产品的位置，buff 则用于存储产品。
2. 初始化信号量：定义了 empty 和 full 两个信号量和 mutex 互斥锁。empty信号量表示缓冲区空闲的数量，full信号量表示缓冲区已满的数量，mutex互斥锁用于保证同一时间只有一个线程来访问缓冲区。
3. 创建线程：定义了 customers 和 producers 两个数组用于存储线程，分别对应生产者和消费者。通过循环创建线程，并将线程 id 作为参数传入线程函数。
4. 生产产品：在 product 函数中，首先打开指定文件，并循环读取文件中的数据。如果文件已读到末尾，则重新从文件开始处读取。然后，将读取到的数据放入缓冲区中，并输出提示信息。
5. 消费产品：在 consume 函数中，从缓冲区中取出产品，并输出提示信息。
6. 等待线程结束：在主函数中，调用 pthread\_join 来等待所有线程完成任务。
7. 销毁信号量：在结束程序之前，销毁信号量及互斥锁，释放资源。

程序通过使用信号量和互斥锁，实现了线程之间的同步互斥，避免了线程安全问题。同时通过生产者消费者模型，实现了多个线程对于同一个数据源的访问，并且对数据进行了缓存，从而实现了生产与消费的解耦。

以下是程序的源代码：

#include <stdio.h>  
#include <stdlib.h>  
#include <pthread.h>  
#include <semaphore.h>  
  
#define NUM\_PRODUCER 3  
#define NUM\_CONSUMER 4  
#define BUFFER\_SIZE 5  
#define FILE\_PATH "./source.txt"  
  
FILE \*fp;  
int data = 0;  
  
int in = 0,  
    out = 0,  
    buff[BUFFER\_SIZE] = {0};  
  
// semaphore  
sem\_t empty, full;  
pthread\_mutex\_t mutex;  
pthread\_mutex\_t file\_mutex;  
  
pthread\_t consumers[NUM\_CONSUMER];  
pthread\_t producers[NUM\_PRODUCER];  
int consumer\_id[NUM\_CONSUMER];  
int producer\_id[NUM\_PRODUCER];  
  
void PrintBuffer() {  
    for (int i = 0; i < BUFFER\_SIZE; i++)  
    {  
        if (buff[i] == 0)  
            printf("ยท");  
        else  
            printf("%d ", buff[i]);  
    }  
    printf(" [in:%d out:%d]\n", in, out);  
}  
  
void \*producer(void \*id) {  
    printf("Producer %d\n", \*(int \*)id);  
  
    while (1) {  
        sem\_wait(&empty);  
        pthread\_mutex\_lock(&mutex);  
        pthread\_mutex\_lock(&file\_mutex);  
        int stat = fscanf(fp, "%d", &data);  
        pthread\_mutex\_unlock(&file\_mutex);  
  
        if (stat == EOF)  
        {  
            buff[in] = -1;  
            sem\_post(&full);  
            pthread\_mutex\_unlock(&mutex);  
            break;  
        }  
  
        buff[in] = data;  
        printf("Producer %d produced [%d] in buffer[%d]\n",  
        \*(int \*)id, buff[in], in);  
  
        in = (in + 1) % BUFFER\_SIZE;  
        PrintBuffer();  
  
        pthread\_mutex\_unlock(&mutex);  
        sem\_post(&full);  
  
    }  
    fclose(fp);  
    pthread\_exit(NULL);  
}  
  
void \*consumer(void \*id) {  
    printf("Consumer %d\n", \*(int \*)id);  
    while (1)  
    {  
        sem\_wait(&full);  
        pthread\_mutex\_lock(&mutex);  
  
        int product = buff[out];  
        if (product == -1) {  
            pthread\_mutex\_unlock(&mutex);  
            pthread\_exit(NULL);  
            break;  
        }  
  
        buff[out] = 0;  
        printf("Consumer %d consumed [%d] in buffer[%d]\n",  
            \*(int \*)id, product, out);  
  
        out = (out + 1) % BUFFER\_SIZE;  
        PrintBuffer();  
  
        pthread\_mutex\_unlock(&mutex);  
        sem\_post(&empty);  
    }  
}  
  
void Init() {  
    sem\_init(&empty, 0, BUFFER\_SIZE);  
    sem\_init(&full, 0, 0);  
    pthread\_mutex\_init(&mutex, NULL);  
    pthread\_mutex\_init(&file\_mutex, NULL);  
  
    fp = fopen(FILE\_PATH, "r");  
  
    for (int i =0; i<NUM\_PRODUCER; i++) {  
        producer\_id[i] = i+1;  
        pthread\_create(&producers[i], NULL, producer, &producer\_id[i]);  
    }  
    for (int i =0; i<NUM\_CONSUMER; i++) {  
        consumer\_id[i] = i+1;  
        pthread\_create(&consumers[i], NULL, consumer, &consumer\_id[i]);  
    }  
}  
  
void Destroy() {  
    sem\_destroy(&empty);  
    sem\_destroy(&full);  
    pthread\_mutex\_destroy(&mutex);  
    pthread\_mutex\_destroy(&file\_mutex);  
}  
  
int main(void) {  
    printf("Hello, World!\n");  
  
    Init();  
    for (int i = 0; i < NUM\_PRODUCER; i++) {  
        pthread\_join(producers[i], NULL);  
    }  
    for (int i = 0; i < NUM\_CONSUMER; i++)  
    {  
        pthread\_join(consumers[i], NULL);  
    }  
  
    Destroy();  
    printf("All done");  
  
  
    return 0;  
}

在Ubuntu 22.04.1 LTS环境下运行结果如下图。

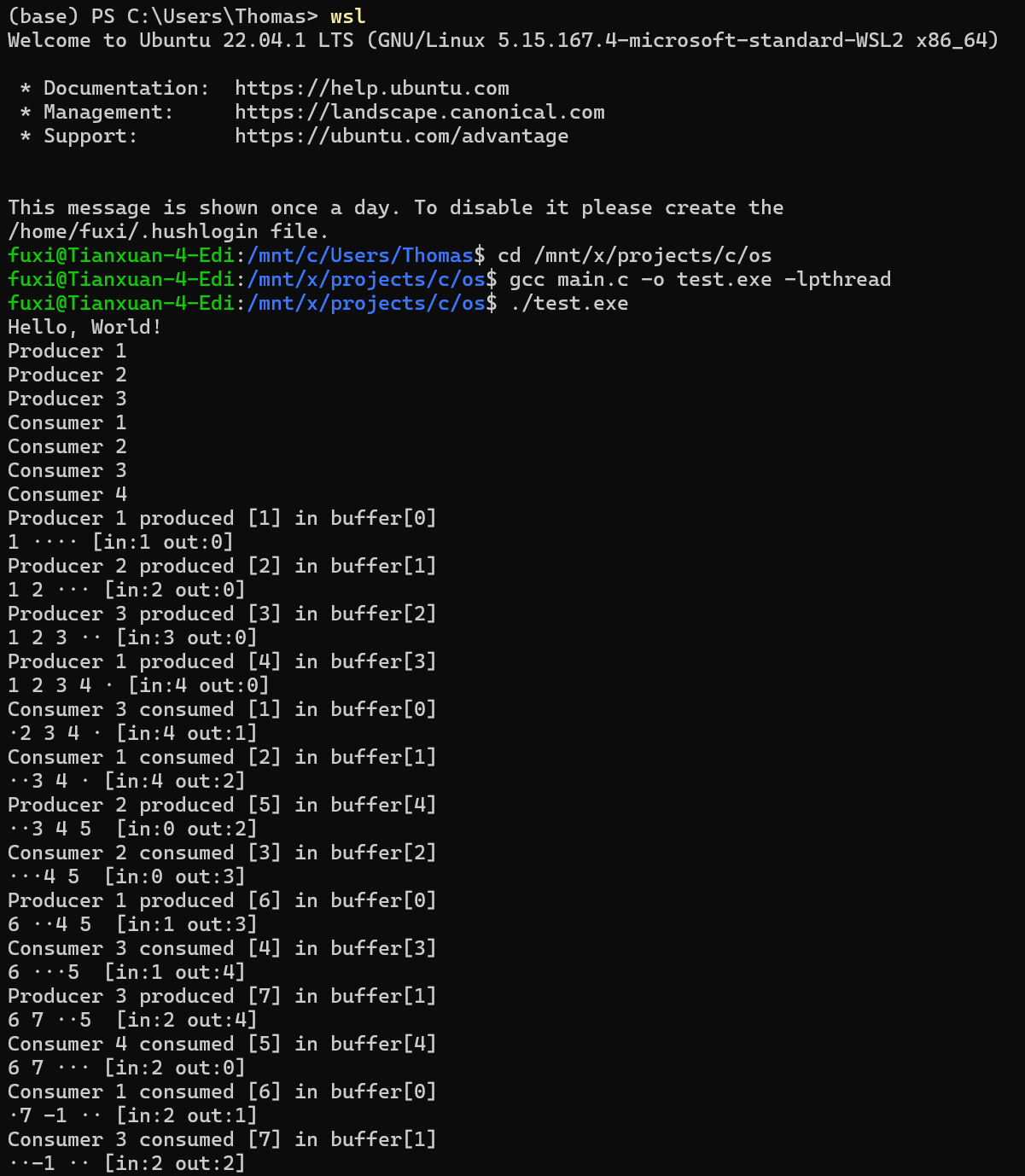


图 8-1 生产者/消费者问题代码运行结果

随着程序的运行，生产和消费的过程不断重复，缓冲区中的产品不断更新。总体来看，程序能够有效地实现多线程协同工作并保证线程安全，达到实验目标。

另外，本实验经过实验老师和助教的检查，得到了A+的评分，包括哲学家就餐问题。学家就餐问题代码运行结果如下。

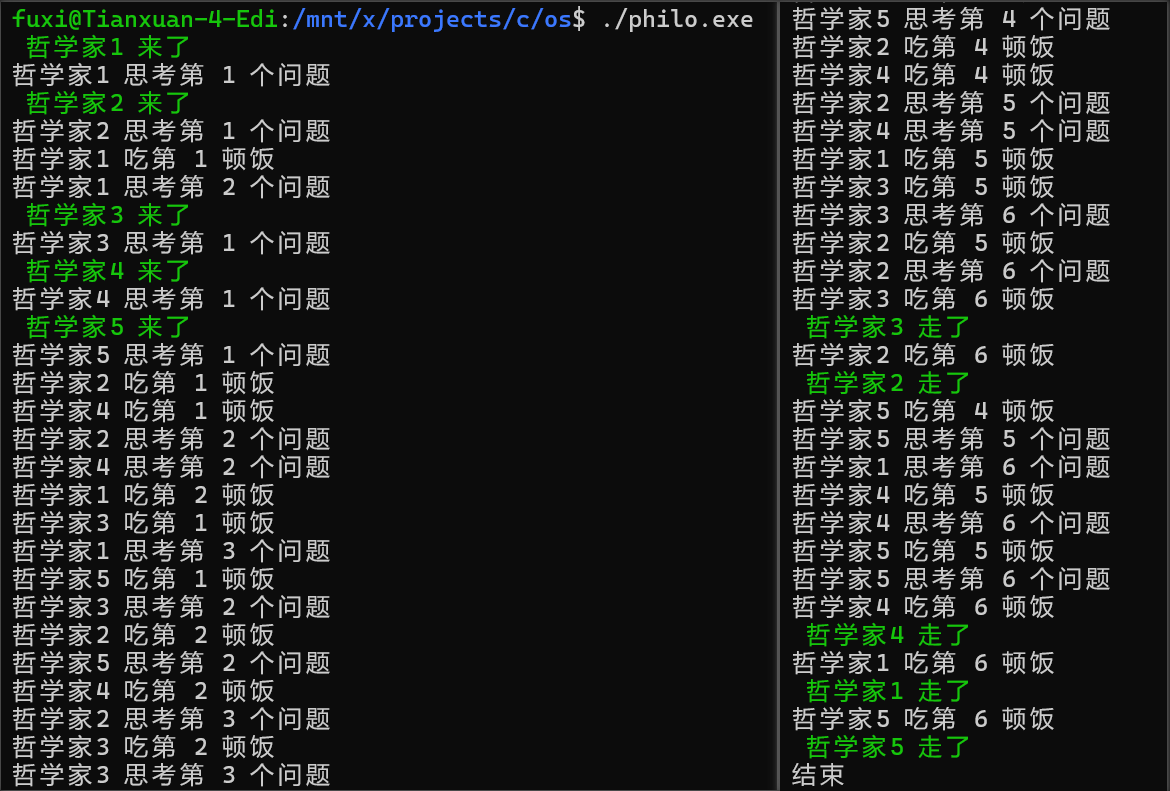


图 8-2 哲学家就餐问题代码运行结果

1. **总结及心得体会：**

生产者消费者问题是计算机科学中一个重要的问题，也是操作系统在进程管理中的核心。在本次实验中，我深入学习了生产者消费者问题的概念、原理和实现方式，也更好地理解了在并发程序中，如何通过共享内存缓存区，实现多个进程之间的协作。

通过本实验，我深入学习了如何使用 pthread 库来创建线程、销毁线程等操作，以及如何使用 sem\_init()、sem\_wait()、sem\_post()等函数来初始化、等待和释放信号。

总的来说，本次实验让我更好地理解了生产者消费者问题的概念、原理和应用，同时也加强了我对多进程、多线程、进程同步和互斥等技术的理解。同时，本次实验也让我更加深刻地认识到了并发编程所涉及的线程安全的重要性。通过这次实验，我掌握了一些基本的并发编程技巧，也提高了我对并发编程的兴趣和热情。

1. **对本实验过程及方法、手段的改进建议：**

**报告评分：**

**指导教师签字：**