北京郵電大學

实验报告：实验 3 进程同步实验

学院：计算机学院（国家示范性软件学院）

专业：计算机科学与技术

班级：2022211305

学号：2022211683

姓名：张晨阳

2024 年 11 月 22 号

目录

1.实验概述

1.1. 实验内容

生产者消费者问题（Producer-consumer problem），也称有限缓冲问题（Bounded-buffer problem），是一个多线程同步问题的经典案例。该问题描述了两个共享固定大小缓冲区的线程在实际运行时会发生的问题。生产者的主要作用是生成一定量的数据放到缓冲区中，然后重复此过程。与此同时，消费者也在缓冲区消耗这些数据。该问题的关键就是要保证生产者不会在缓冲区已经装满时加入数据，消费者也不会在缓冲区为空时消耗数据。

1.2. 实验要求

(1) 缓冲区

(a) 缓冲区存储结构建议采用固定大小的数组表示，并作为环形队列处理。

(b) 缓冲区的访问算法按照课本 6.6.1 节图 6.10、图 6.11 进行设计。

(2) 主函数 main()

(a) 主函数需要创建一定数量的生产者线程与消费者线程。线程创建完毕后，主函数将睡眠一段时间，并在唤醒时终止应用程序。

(b) 主函数需要从命令行接受三个参数：睡眠时长、生产者线程数量、消费者线程数量。

(3) 生产者与消费者线程

(a) 生产者线程：随机睡眠一段时间，向缓冲区插入一个随机数。

(b) 消费者线程：随机睡眠一段时间，从缓冲区去除一个随机数。

1.3. 实验环境

- Visual Studio Code 1.94.2
- Windows 11
- c11

2.程序设计说明

2.1. 问题分析

生产者-消费者问题简介

生产者-消费者问题是经典的多线程并发问题之一，涉及两个线程：

- **生产者**：负责生成数据并将其存入共享缓冲区。
- **消费者**：从共享缓冲区中提取数据并进行消费。

两者需要通过共享缓冲区进行通信，同时需要解决以下问题：

1. **同步问题**：
   - 生产者不能向已满的缓冲区中插入数据。
   - 消费者不能从空缓冲区中提取数据。
2. **互斥问题**：
   - 生产者和消费者不能同时访问缓冲区，防止数据竞争。
   - 必须使用互斥机制保证缓冲区操作的原子性。

本实验要求通过多线程机制实现生产者-消费者问题，主要目标：

1. 设计一个固定大小的共享缓冲区，作为生产者和消费者之间的通信桥梁。
2. 使用信号量解决同步和互斥问题：
   - empty 信号量：记录缓冲区中空槽的数量。
   - full 信号量：记录缓冲区中已占用槽的数量。
   - mutex 信号量：实现对缓冲区的互斥访问。
3. 实现多生产者、多消费者的场景，并模拟随机的生产和消费时间。

2.2. 总体设计

设计目标

程序通过多线程和信号量机制，模拟生产者和消费者的并发操作，确保：

- 多线程间的同步和互斥问题得到妥善解决。
- 缓冲区中的数据流动能够正确反映生产和消费过程。

核心模块设计

程序分为以下几个核心模块：

1. **生产者模块**：生成一个随机数据，等待空槽可用后将数据插入缓冲区。
2. **消费者模块**：等待有数据可用后从缓冲区中取出数据，并模拟消费过程。
3. **随机数生成模块**：使用高精度计时器初始化随机种子，确保每次运行生成的随机数序列不同。
4. **主控模块**：负责程序初始化、线程创建和最终的资源清理。

工作流程

1. **初始化**：

- 创建信号量：empty（初始值为缓冲区大小）、full（初始值为 0）、mutex（初始值为 1）。
- 初始化缓冲区的指针 in 和 out。

2. **线程创建**：

- 根据命令行参数，启动指定数量的生产者和消费者线程。

3. **线程操作**：

- 生产者线程生成数据并插入缓冲区，更新 in 指针。
- 消费者线程取出数据并消费，更新 out 指针。
- 信号量控制线程间的同步和互斥。

4. **终止和清理**：

- 主线程在指定时间后终止所有生产者和消费者线程。
- 销毁信号量，释放资源。

2.3. 数据结构与变量定义

1. 共享缓冲区

```
#define BUFFER_SIZE 5
int buffer[BUFFER_SIZE];
int in = 0, out = 0;
```

- buffer：固定大小的数组，用作共享缓冲区。
- in：生产者插入数据的位置索引，循环使用。
- out：消费者提取数据的位置索引，循环使用。

2. 信号量

```
sem_t empty, full, mutex;
```

- empty：记录缓冲区中空槽数量，初始值为 BUFFER_SIZE。
- full：记录缓冲区中已占用槽数量，初始值为 0。
- mutex：互斥信号量，用于保护对缓冲区的并发访问，初始值为 1。

3. 线程

```
pthread_t producers[num_producers], consumers[num_consumers];
int producer_ids[num_producers], consumer_ids[num_consumers];
```

- pthread_t：线程的标识符，用于管理生产者和消费者线程。
- producer_ids、consumer_ids：每个线程的唯一标识，用于输出和调试。

4. 随机数生成

```
void initialize_random_seed();
int true_random(int min, int max);
```

- initialize_random_seed：每个线程独立初始化随机数种子。
- true_random：生成指定范围 [min, max] 内的随机整数。

2.4. 模块设计

1. 生产者模块

函数：void* producer_function(void* arg)

功能描述：

1) 生成一个随机数（代表生产的物品）。
2) 等待缓冲区的空槽（sem_wait(&empty)）。
3) 获取互斥锁，插入数据到缓冲区（sem_wait(&mutex)）。
4) 释放互斥锁，通知缓冲区已增加数据 sem_post(&mutex) 和 sem_post(&full)。
5) 模拟随机延迟（Sleep）。

流程：

1) 等待空槽可用（empty > 0）。
2) 加锁，插入数据。
3) 解锁，通知有新数据可用。

2. 消费者模块

函数：void* consumer_function(void* arg)

功能描述：

1) 等待缓冲区中有数据（sem_wait(&full)）。
2) 获取互斥锁，提取数据（sem_wait(&mutex)）。
3) 释放互斥锁，通知缓冲区空槽增加 sem_post(&mutex) 和 sem_post(&empty)。
4) 模拟随机延迟（Sleep）。

流程：

1) 等待数据可用（full > 0）。
2) 加锁，提取数据。
3) 解锁，通知空槽增加。

3. 随机数生成模块

函数：void initialize_random_seed() 和 int true_random(int min, int max)

功能描述：

1) 初始化随机数种子，使用高精度计时器 QueryPerformanceCounter 确保种子独立性。
2) 生成范围内的随机整数，用于生产数据和模拟随机延迟。

4. 主控模块

主要逻辑如下：

1) 初始化信号量。
2) 创建指定数量的生产者和消费者线程。
3) 主线程睡眠指定时间后终止子线程。
4) 销毁信号量，释放资源。

3.程序执行结果

3.1. 编译与运行环境

操作系统：Windows 11

编译器： MinGW 编译器，gcc version 8.1.0

编译命令：

```
> gcc -o producer_consumer producer_consumer.c -lpthread
```

运行命令：

运行程序时的命令格式如下：

```
> ./producer_consumer <sleep_time> <num_producers> <num_consumers>
```

- <sleep_time>：程序运行的总时间（单位：秒）。
- <num_producers>：生产者线程数量。
- <num_consumers>：消费者线程数量。

3.2. 测试用例说明

测试用例 1：基本测试

参数：

```
sleep_time = 10
num_producers = 3
num_consumers = 2
```

目的：

- 验证程序在多生产者和多消费者情况下的正常运行。
- 确认缓冲区的容量限制有效，生产和消费顺序正确。

测试用例 2：高并发测试

参数：

```
sleep_time = 8
num_producers = 5
num_consumers = 5
```

目的：

- 验证高并发情况下的程序稳定性。
- 确认信号量和互斥锁是否正确同步生产者和消费者的行为。

测试用例 3：极限测试

参数：

```
sleep_time = 15
num_producers = 15
num_consumers = 1
```

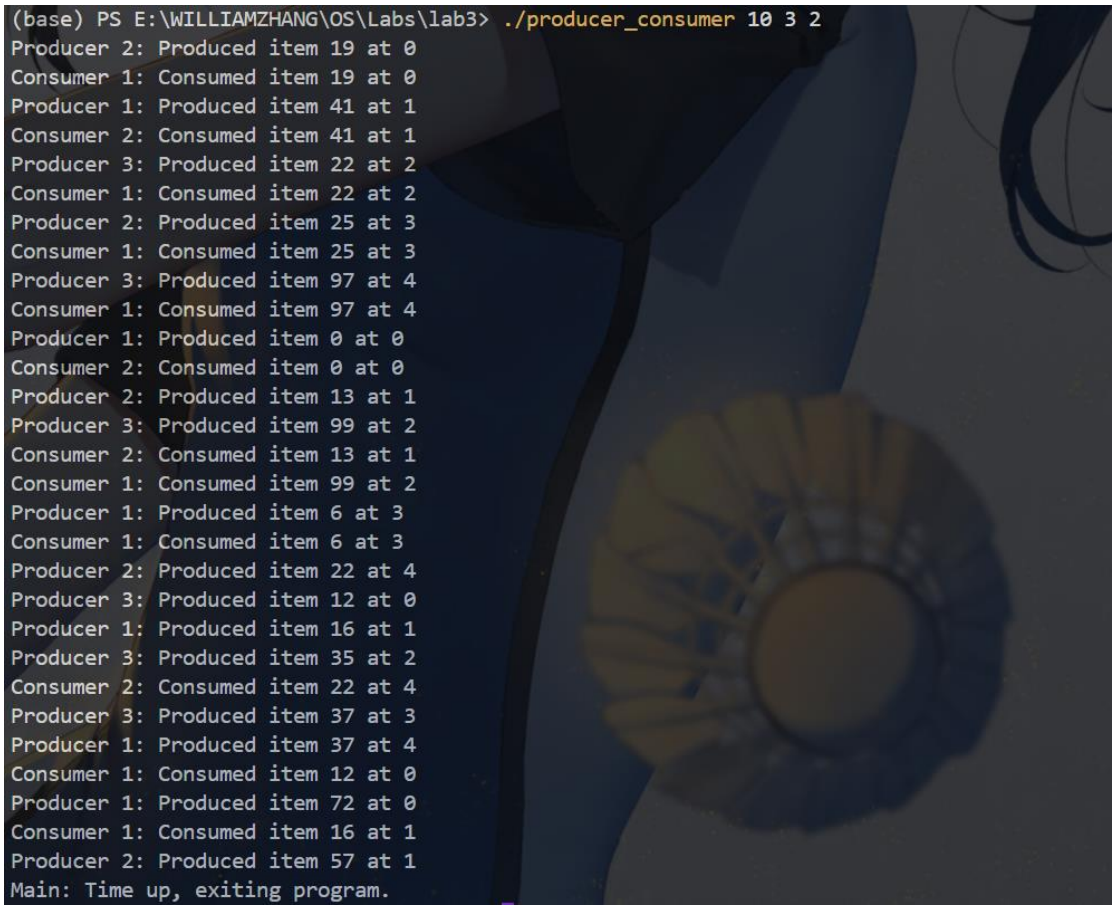
目的：

- 验证消费者远少于生产者时缓冲区是否正常工作。
- 测试 empty 和 full 信号量的边界情况。

3.3. 测试结果与分析

测试用例 1：基本测试

运行截图：

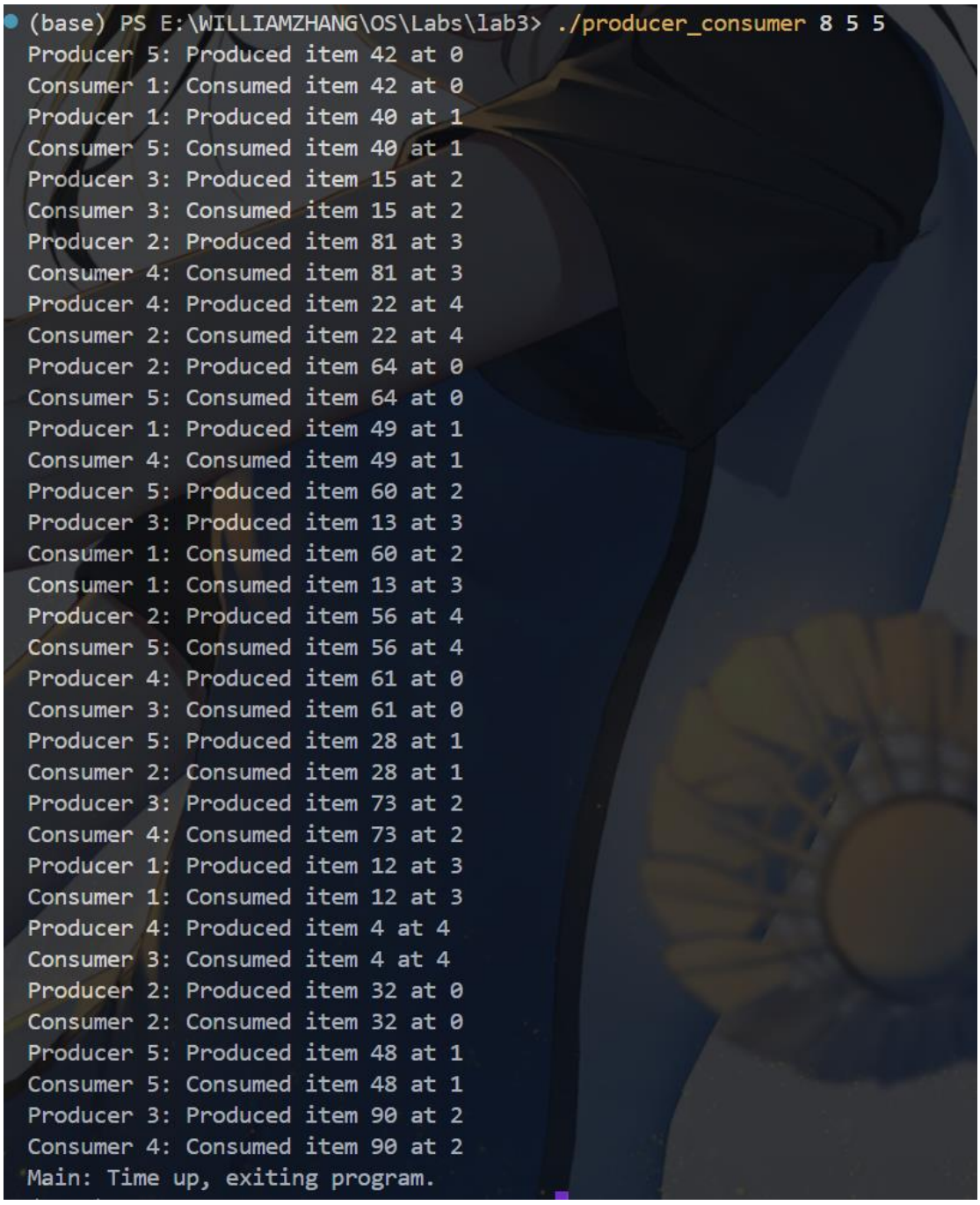


运行分析：

- **正常性**：生产者生成的物品编号符合随机生成的范围 [0, 99]。
- **同步性**：消费者仅在缓冲区中有数据时消费，未出现消费空数据的情况。
- **互斥性**：多个线程并发时缓冲区操作未出现竞态条件。

测试用例 2：高并发测试

运行截图：

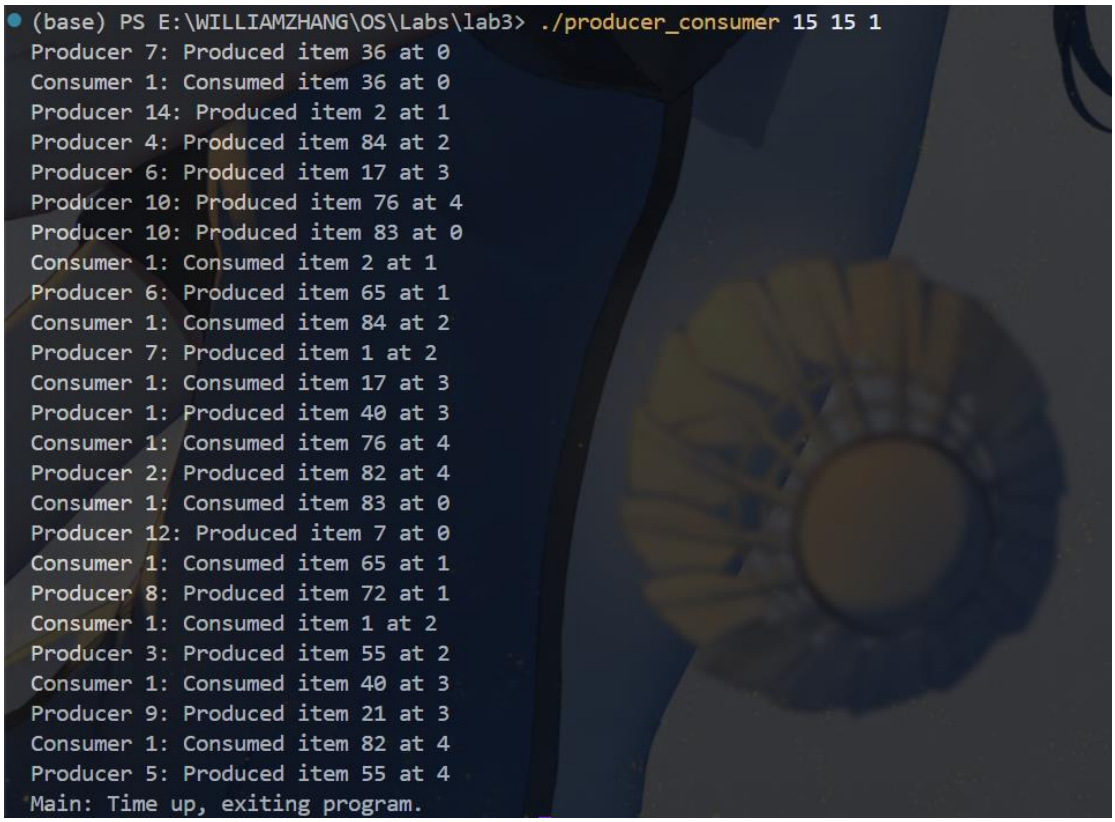


运行分析：

- **性能**：高并发情况下，生产者和消费者行为交替进行，程序无死锁或阻塞。
- **正确性**：缓冲区容量未超过定义的大小，物品进出顺序正确。
- **随机性**：生产者和消费者的行为具有较强的随机性，延迟时间分布合理。

测试用例 3：极限测试

运行截图：



运行分析：

- **缓冲区边界**：消费者速度较慢时，生产者正确等待空槽可用，未出现溢出错误。
- **信号量作用**：empty 和 full 信号量在缓冲区满时阻塞生产者，在缓冲区空时阻塞消费者，验证了同步机制的有效性。
- **线程协调**：尽管生产者数量远大于消费者，但缓冲区的状态保持稳定。

4.心得总结

通过本次实验，我深入学习并实践了经典的生产者-消费者问题，全面了解了线程间同步与互斥机制的重要性。

实验中使用信号量 empty、full 和 mutex 解决了多线程操作共享缓冲区时可能出现的同步问题，以及数据竞争带来的错误。同时，通过实现一个环形缓冲区，我更深刻理解了共享数据结构的设计及其在并发环境中的应用。

在随机数生成方面，我尝试了基于高精度计时器的随机种子初始化，使得每次运行程序生成的随机数序列更加随机化，这提升了实验的真实感。在多线程的实现中，我体会到了线程调度和资源共享的复杂性。通过实验，我进一步理解了信号量的作用和工作原理，以及如何通过它来控制线程的访问顺序和共享资源的状态。

此外，实验过程中我深刻认识到测试和调试在多线程编程中的重要性。由于线程调度的不可预测性，测试和验证程序的正确性变得尤为重要。在不同的测试用例下，我发现程序的表现非常符合预期，这充分验证了代码的稳定性和设计的合理性。同时，我也认识到优化线程终止的方式是未来改进的方向，例如使用标志位代替 pthread_cancel，使线程能够优雅退出。

总体来说，这次实验不仅让我掌握了生产者-消费者问题的解决方法，还让我对多线程编程有了更深层次的理解。我学会了如何合理设计线程间的通信机制，如何利用信号量保障数据一致性，同时也意识到了多线程编程中的细节处理与优化空间。

在未来的学习和实践中，我将继续深入研究并运用这些技术，提升自己在并发编程方面的能力。