北京郵電大學

Beijing University of Posts and Telecommunications

实 验 报 告

课 程 名 称	操作系统
题 目	进程同步实验-1
姓 名	沈原灏
学 号	2021211108
班 级	2021211306
指 导 教 师	李文生

2023 年 11 月

目录

一、实验概述

1.1 实验内容

基于 openEuler 操作系统，设计一个 C 语言程序，利用信号量机制解决有限缓冲的生产者-消费者问题。

具体内容参考教材《操作系统概念》第六章项目：生产者-消费者问题。

1.2 实验内容

基于 openEuler 操作系统，设计一个 C 语言程序，利用信号量机制解决有限缓冲的生产者-消费者问题。

(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

- OpenEuler 20.03(LTS) (由于虚拟机不太好编辑代码，采用了华为云的 ECS 服务器完成)
- MobaXterm：一款远程桌面管理软件，支持 ssh 远程连接服务器
- gcc 7.3.0

二、实验设计

2.1 API 概述

API	头文件	功能	返回值
`int sem_init(sem_t *sem, int pshared, unsigned int value)`	<semaphore.h>	初始化信号量	初始化成功返回 0，失败返回-1
`int sem_wait(sem_t *sem)`	<semaphore.h>	如果信号量的值大于零，则减量继续进行，函数立即返回。如果信号量当前的值为零，则调用将阻塞，直到有可能执行减量操作为止。	运行成功则返回 0，失败返回-1。
`int sem_post(sem_t *sem)`	<semaphore.h>	解锁信号量 sem。	运行成功则返回 0，失败返回-1。
`int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void*(*start_routine)(void *), void *arg)`	<pthread.h>	调用 pthread_create()函数的进程中创建一个新的线程。	函数运行成功返回 0，失败则返回一个错误编号。

2.2 数据结构设计

信号量采用 mutex, full 与 empty，mutex 进行互斥操作，full 表示缓冲区内的产品数，empty 表示缓冲区的空额数。

缓冲区采用固定大小的数组表示，并作为环形队列处理。

```
sem_t mutex, full, empty;
int buffer[BUFFER_SIZE];
int in = 0, out = 0;
```

2.3 main 程序设计

main 程序包括读取命令行参数，初始化信号量，创建线程，睡眠等操作。

```c
int main(int argc, char *argv[])
{
  // Parse command line arguments
  if (argc != 4)
  {
    printf("Usage: %s <sleep duration> <producer threads> <consumer
threads>\n", argv[0]);
    return -1;
  }
  int sleep_duration = atoi(argv[1]);
  int num_producers = atoi(argv[2]);
  int num_consumers = atoi(argv[3]);

  // Initialize semaphores
  sem_init(&mutex, 0, 1);
  sem_init(&full, 0, 0);
  sem_init(&empty, 0, BUFFER_SIZE);

  // Create producer threads
  pthread_t producer_threads[num_producers];
  for (int i = 0; i < num_producers; i++)
  {
    srand(time(NULL) + i);
    pthread_create(&producer_threads[i], NULL, producer, NULL);
  }

  // Create consumer threads
  pthread_t consumer_threads[num_consumers];
  for (int i = 0; i < num_consumers; i++)
  {
    srand(time(NULL) + i);
    pthread_create(&consumer_threads[i], NULL, consumer, NULL);
  }

  // Sleep for the specified duration
  sleep(sleep_duration);

  // Terminate the program
  return 0;
}
```

2.4 producer 与 consumer 程序设计

producer 函数采取书上的教程，先生产一个随机数，再等待 empty 信号量获

取一个缓冲区空位，再等待 mutex 互斥将生产出来的随机数放入缓冲区中，最后释放信号量 mutex 与 full，随机睡眠一小段时间。

consumer 函数采取书上的教程，先等待 full 信号量获取一个缓冲区放置了产品的额度，再等待 mutex 互斥获取其位置上的产品，最后释放信号量 mutex 与 mutex，随机睡眠一小段时间。

```
void *producer(void *arg)
{
   while (1)
   {
      // Produce item
      int item = rand() % 100;

      // Wait for an empty slot in the buffer
      sem_wait(&empty);
      sem_wait(&mutex);

      // Add item to the buffer
      buffer[in] = item;
      in = (in + 1) % BUFFER_SIZE;

      // Signal that buffer is no longer empty
      sem_post(&mutex);
      sem_post(&full);

      // Output producer information
      printf("Producer %lu produced: %d\n", pthread_self(), item);

      // Sleep for a random period
      usleep(rand() % 1000000);
   }
}

void *consumer(void *arg)
{
   while (1)
   {
      // Wait for a full slot in the buffer
      sem_wait(&full);
      sem_wait(&mutex);

      // Consume item from the buffer
      int item = buffer[out];
```

```
        out = (out + 1) % BUFFER_SIZE;

        // Signal that buffer is no longer full
        sem_post(&mutex);
        sem_post(&empty);

        // Output consumer information
        printf("Consumer %lu consumed: %d\n", pthread_self(), item);

        // Sleep for a random period
        usleep(rand() % 1000000);
    }
}
```

三、实验结果分析

3.1 执行程序

openEuler 自带一个版本较旧的 gcc，与我所熟悉的 Ubuntu 环境下不同，编译该程序时需要手动链接 pthread 库。与 openEuler 虚拟机不太一样，服务器自带了 vim，而且在 MobaXterm 中支持 ssh 连接服务器后图形化编辑服务器文件，较为便捷。

```
gcc -pthread lab4.c -o lab4
```

```
[root@oe1 ~]# gcc -pthread lab4.c -o lab4
[root@oe1 ~]#
```

命令行执行的三个参数分别是，主程序睡眠时间，生产者线程数与消费者线程数：

```
[root@oe1 ~]# ./lab4
Usage: ./lab4 <sleep duration> <producer threads> <consumer threads>
[root@oe1 ~]#
```

可以看到，输入参数后，生产者与消费者可以正常地运行：

```
[root@oe1 ~]# ./lab4 2 3 1
Producer 140440449304320 produced: 75
Producer 140440440911616 produced: 41
Producer 140440432518912 produced: 58
Consumer 140440424126208 consumed: 75
Consumer 140440424126208 consumed: 41
Producer 140440449304320 produced: 38
Consumer 140440424126208 consumed: 58
Producer 140440432518912 produced: 48
Producer 140440440911616 produced: 24
Consumer 140440424126208 consumed: 38
Producer 140440449304320 produced: 10
Producer 140440449304320 produced: 23
Producer 140440440911616 produced: 49
Consumer 140440424126208 consumed: 48
Producer 140440449304320 produced: 46
Consumer 140440424126208 consumed: 24
Producer 140440432518912 produced: 12
Consumer 140440424126208 consumed: 10
Producer 140440449304320 produced: 70
[root@oe1 ~]#
```

3.2 分析

主线程成功创建了各个生产者与消费者线程，并且都按照各自的申请资源的逻辑阻塞或运行，但是由于创建线程的顺序并不是严格依照线程 ID 的顺序，因此输出和预期有一些误差。总体而言程序的正确性可以保证。

四、总结心得

本次实验总体不是很难，个人认为重点在于理解并实现伪代码转化为实际代码的过程。我在这次实验中熟悉了 openEuler 系统下的编程，对 pthread 等库有了更深的了解。配置服务器环境用了半小时，代码编写用了约 1 小时，调试运行用了约 1 小时。

本次实验中我对于程序开发的思路和过程有了更清晰地认识。通过理论和实践相结合，我更深刻地认识到信号量机制的实现。也让我明白了，只有深入了解理论，才能写出更加高效的代码。