

华中科技大学

实 验 报 告

课 程：操作系统原理

实验序号：第 2 次实验

实验名称：线程实现“生产者-消费者”程序

院 系：软件学院

专业班级：

学 号：

姓 名：

2023 年 03 月 28 日

一、实验目的

1. 理解进程/线程的概念，会对进程/线程进行基本控制；
2. 理解进程/线程的典型同步机制和应用编程；
3. 掌握和推广国产操作系统（推荐银河麒麟或优麒麟）

二、实验内容

在 Linux 下利用线程实现“生产者-消费者”同步控制。使用数组（10 个元素）代替缓冲区。2 个输入线程产生产品（随机数）存到数组中；3 个输出线程从数组中取数输出。

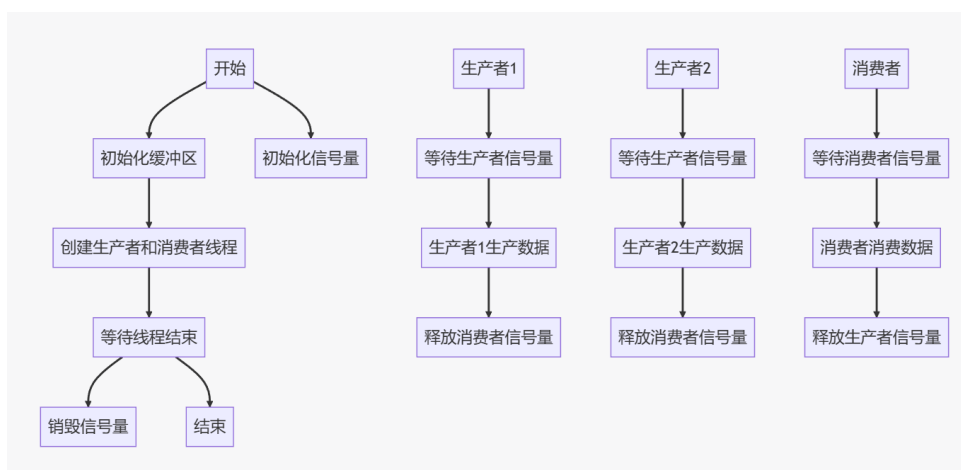
Linux 使用互斥锁对象和轻量级信号量对象，主要函数：sem_wait(), sem_post(), pthread_mutex_lock(), pthread_mutex_unlock()。屏幕打印（或日志文件记录）每个数据的生产和消费记录。

三、实验环境

麒麟 Kylin(V10) + gcc/g++ + vi/vim/vscode/notepad++

四、程序的设计思路

4.1 程序结构



4.2 关键函数或参数或机制

1. 创建线程

```
#include <pthread.h>
int pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *
(*start_routine) (void *), void *arg);
```

参数 pthread_t * thread 指向线程创建成功时线程 id 所在的内存单元; pthread_attr_t * attr 为创建时属性; start_routine 即 run in thread 回调, 这是一个函数指针, 指向函数返回类型为 void*, 参数类型是 void*; void *arg 即 run in thread 回调中的参数。

使用示例:

```
void* count1(void * arg) {};

int main() {
    pthread_t tid1;
    pthread_create(&tid1, NULL, count1, NULL);
    return 0;
}
```

2. 结束线程

```
#include <pthread.h>
void pthread_exit(void *retval);

#include <pthread.h>
int pthread_join(pthread_t thread, void **retval);
```

pthread_join 可以获得 return 和 exit 的状态码, 如果进程没有结束, 则该程序不会结束, 直到进程结束并返回相应的返回值。

使用示例:

```
void* count1(void * arg){
    return NULL;
};

int main(){
    pthread_t tid1;
    pthread_create(&tid1, NULL, count1, NULL);
    pthread_join(tid1, NULL);
    return 0;
}
```

五、关键代码分析

5.1 程序关键片段一：初始化和定义

这部分代码主要是对程序所需的库进行引用，定义了互斥锁、信号量、缓冲区以及生产者和消费者的函数原型。

```
#include<stdio.h>

#include<pthread.h>

#include<unistd.h>

#include<stdlib.h>

#include<semaphore.h>

pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

#define buffer_size 10

sem_t consumer_sem;

sem_t producer_sem;

void *consumer(void*);

void *producer1(void*);

void *producer2(void*);
```

```
int buffer[buffer_size];
```

5.2 程序关键片段二：主函数

主函数中，首先初始化了缓冲区，然后创建了两个生产者线程和三个消费者线程，并等待这些线程执行完毕。最后，销毁了之前创建的信号量。

```
int main(){
    for(int i=0;i<buffer_size;i++)
        buffer[i]=-1;

    pthread_t prod1, prod2, cons1, cons2, cons3;
    sem_init(&consumer_sem,0,0);
    sem_init(&producer_sem,0,buffer_size);

    pthread_create(&prod1,NULL,producer1,NULL);
    pthread_create(&prod2,NULL,producer2,NULL);
    pthread_create(&cons1,NULL,consumer,NULL);
    pthread_create(&cons2,NULL,consumer,NULL);
    pthread_create(&cons3,NULL,consumer,NULL);

    pthread_join(prod1,NULL);
    pthread_join(prod2,NULL);
    pthread_join(cons1,NULL);
    pthread_join(cons2,NULL);
    pthread_join(cons3,NULL);

    sem_destroy(&consumer_sem);
    sem_destroy(&producer_sem);
    return 0;
}
```

5.3 程序关键片段三：消费者函数

消费者函数中，消费者在消费前会先等待信号量，然后锁定互斥锁，消费产品后，释放互斥锁，并通知生产者。

```
void *consumer(void *junk) {
    while(1) {
        sem_wait(&consumer_sem);
        pthread_mutex_lock(&lock);

        int index = rand()%buffer_size;
        while (buffer[index]==-1)
            index = rand()%buffer_size;
        printf("Consumer consumed: %d\n",buffer[index]);
        buffer[index]=-1;

        sem_post(&producer_sem);

        pthread_mutex_unlock(&lock);

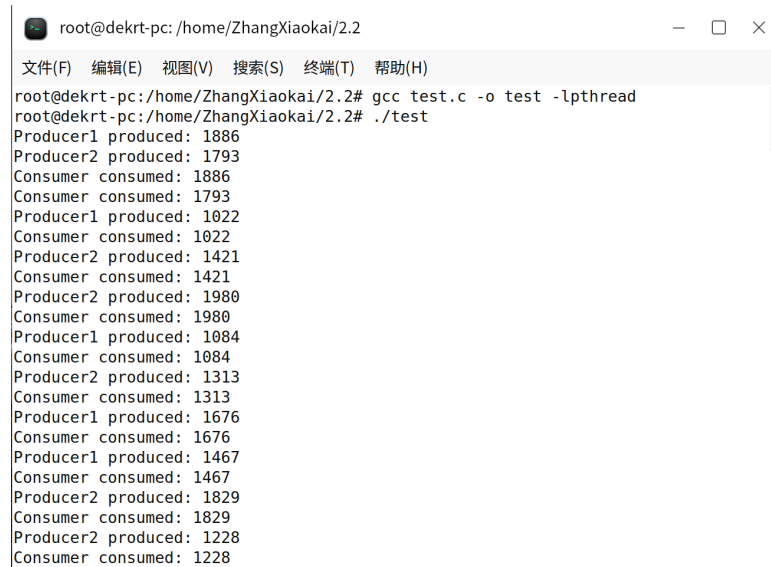
        usleep((rand()%900+100)*1000);
    }
}
```

5.4 程序关键片段四：生产者函数

生产者函数中，生产者在生产前会先等待信号量，然后锁定互斥锁，生产产品后，释放互斥锁，并通知消费者。这里有两个生产者函数，分别是 producer1 和 producer2，它们的功能是一样的，只是在打印输出时标识了是哪个生产者生产的产品。

```
void *producer1(void *junk) {  
    while (1) {  
        sem_wait(&producer_sem);  
        pthread_mutex_lock(&lock);  
  
        int index = rand() % buffer_size;  
        while (buffer[index] != -1)  
            index = rand() % buffer_size;  
        buffer[index] = rand() % 1000 + 1000;  
        printf("Producer1 produced: %d\n", buffer[index]);  
        pthread_mutex_unlock(&lock);  
        sem_post(&consumer_sem);  
        usleep((rand()%900+100)*1000);  
    }  
}  
  
void *producer2(void*junk) {  
    while (1) {  
        sem_wait(&producer_sem);  
        pthread_mutex_lock(&lock);  
  
        int index = rand() % buffer_size;  
        while (buffer[index] != -1)  
            index = rand() % buffer_size;  
        buffer[index] = rand() % 1000 + 1000;  
        printf("Producer2 produced: %d\n", buffer[index]);  
        pthread_mutex_unlock(&lock);  
        sem_post(&consumer_sem);  
        usleep((rand()%900+100)*1000);  
    }  
}
```

六、程序运行结果和分析



```
root@dekart-pc: /home/ZhangXiaokai/2.2
文件(F) 编辑(E) 视图(V) 搜索(S) 终端(T) 帮助(H)
root@dekart-pc:/home/ZhangXiaokai/2.2# gcc test.c -o test -lpthread
root@dekart-pc:/home/ZhangXiaokai/2.2# ./test
Producer1 produced: 1886
Producer2 produced: 1793
Consumer consumed: 1886
Consumer consumed: 1793
Producer1 produced: 1022
Consumer consumed: 1022
Producer2 produced: 1421
Consumer consumed: 1421
Producer2 produced: 1980
Consumer consumed: 1980
Producer1 produced: 1084
Consumer consumed: 1084
Producer2 produced: 1313
Consumer consumed: 1313
Producer1 produced: 1676
Consumer consumed: 1676
Producer1 produced: 1467
Consumer consumed: 1467
Producer2 produced: 1829
Consumer consumed: 1829
Producer2 produced: 1228
Consumer consumed: 1228
```

可以看到，两个线程正确地运行，并且正确地实现了生产者-消费者同步关系。

七、实验错误排查和解决方法

7.1: 编译找不到 thread 相关函数，但是头文件没有问题

gcc 语句的后面需要添加 -lpthread

7.2: gcc 过程显示 wait 函数出问题

没有添加<sys/wait.h>头文件

八、实验参考资料和网址

(1) 教学课件

(2) <https://www.geeksforgeeks.org/producer-consumer-solution-using-threads-in-java/>

(3) <https://github.com/Abdurraheem/Producer-Consumer-Problem>