

廈門大學



软件学院

《实用操作系统》Project2

题 目 基于鸿蒙 Liteos 提供的同步互斥机制

解决生产者消费者问题

姓 名 陈澄

学 号 32420212202930

班 级 软工三班

实验时间 2023/9/26

2023 年 09 月 26 日

1 实验目的

基于鸿蒙 Liteos 提供的同步互斥机制解决一个操作系统实践中遇到的同步互斥经典问题

2 实验环境

主机：Windows 11

虚拟机：Ubuntu 18.04

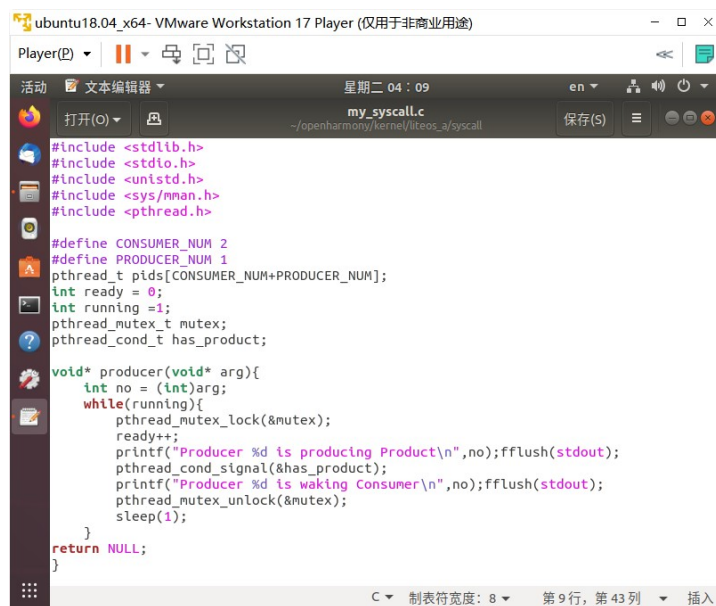
开发板：IMAX6ULL MINI

终端：MobaXterm

3 实验内容

1. 编写测试程序

该实验不涉及对操作系统的改动，因此调用鸿蒙 LiteOs 给出的线程管理方法编写程序即可完成实验。

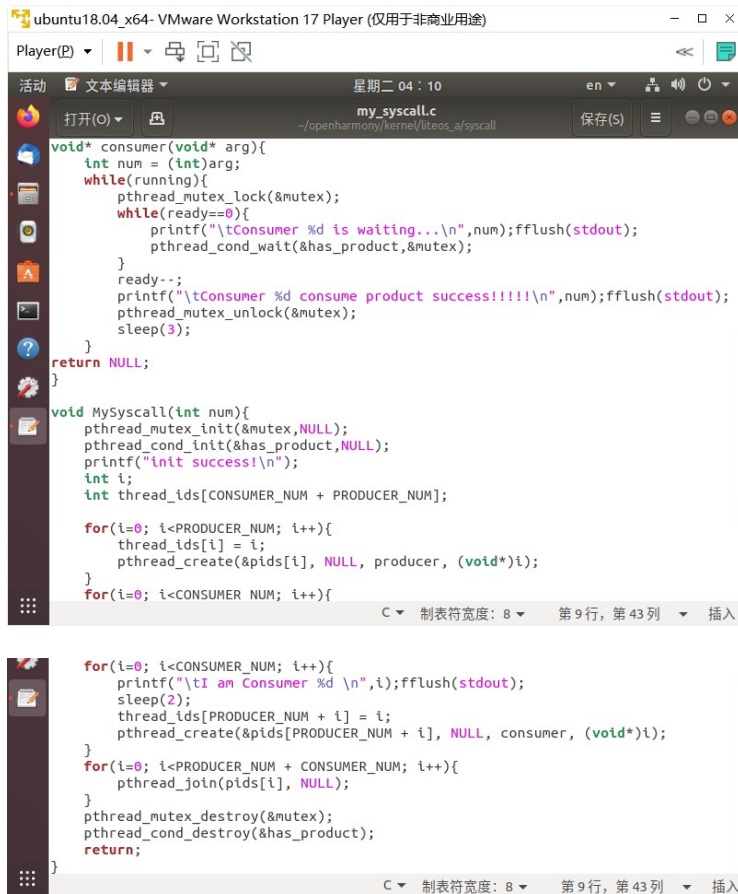


```
ubuntu18.04_x64- VMware Workstation 17 Player (仅用于非商业用途)
Player(P) | | | | |
活动 文本编辑器 星期二 04:09 en 保存(S)
my_syscall.c
~/openharmory/kernel/liteos_a/syscall

#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/mman.h>
#include <pthread.h>

#define CONSUMER_NUM 2
#define PRODUCER_NUM 1
pthread_t pids[CONSUMER_NUM+PRODUCER_NUM];
int ready = 0;
int running = 1;
pthread_mutex_t mutex;
pthread_cond_t has_product;

void* producer(void* arg){
    int no = (int)arg;
    while(running){
        pthread_mutex_lock(&mutex);
        ready++;
        printf("Producer %d is producing Product\n",no);fflush(stdout);
        pthread_cond_signal(&has_product);
        printf("Producer %d is waking Consumer\n",no);fflush(stdout);
        pthread_mutex_unlock(&mutex);
        sleep(1);
    }
    return NULL;
}
```



```
void* consumer(void* arg){
    int num = (int)arg;
    while(running){
        pthread_mutex_lock(&mutex);
        while(ready==0){
            printf("\tConsumer %d is waiting...\n",num);fflush(stdout);
            pthread_cond_wait(&has_product,&mutex);
        }
        ready--;
        printf("\tConsumer %d consume product success!!!!\n",num);fflush(stdout);
        pthread_mutex_unlock(&mutex);
        sleep(3);
    }
    return NULL;
}

void MySyscall(int num){
    pthread_mutex_init(&mutex,NULL);
    pthread_cond_init(&has_product,NULL);
    printf("Init success!\n");
    int i;
    int thread_ids[CONSUMER_NUM + PRODUCER_NUM];

    for(i=0; i<PRODUCER_NUM; i++){
        thread_ids[i] = i;
        pthread_create(&pids[i], NULL, producer, (void*)i);
    }
    for(i=0; i<CONSUMER_NUM; i++){
        for(i=0; i<CONSUMER_NUM; i++){
            printf("\tI am Consumer %d \n",i);fflush(stdout);
            sleep(2);
            thread_ids[PRODUCER_NUM + i] = i;
            pthread_create(&pids[PRODUCER_NUM + i], NULL, consumer, (void*)i);
        }
        for(i=0; i<PRODUCER_NUM + CONSUMER_NUM; i++){
            pthread_join(pids[i], NULL);
        }
        pthread_mutex_destroy(&mutex);
        pthread_cond_destroy(&has_product);
        return;
    }
}
```

2. 代码分析

- 1.在代码中，有两个宏定义 `PRODUCER_NUM` 和 `CONSUMER_NUM`，分别表示生产者的数量和消费者的数量。然后定义了全局变量 `pids` 数组用于存储线程的 ID，`ready` 表示可用产品的数量，`running` 表示程序是否继续运行的标志。
- 2.接下来定义了互斥锁 `mutex` 和条件变量 `has_product`。互斥锁用于保护共享资源的互斥访问，条件变量用于线程间的通信。
- 3.代码中有两个函数，一个是 `producer` 函数，负责生产产品；另一个是 `consumer` 函数，负责消费产品。这两个函数都是无限循环，直到 `running` 为 0 才退出。
- 4.在 `producer` 中，首先使用 `pthread_mutex_lock` 方法对互斥锁进行加锁，将共享资源保护起来，然后生产者进行生产，`ready` 增加，再调用 `pthread_cond_signal` 发送

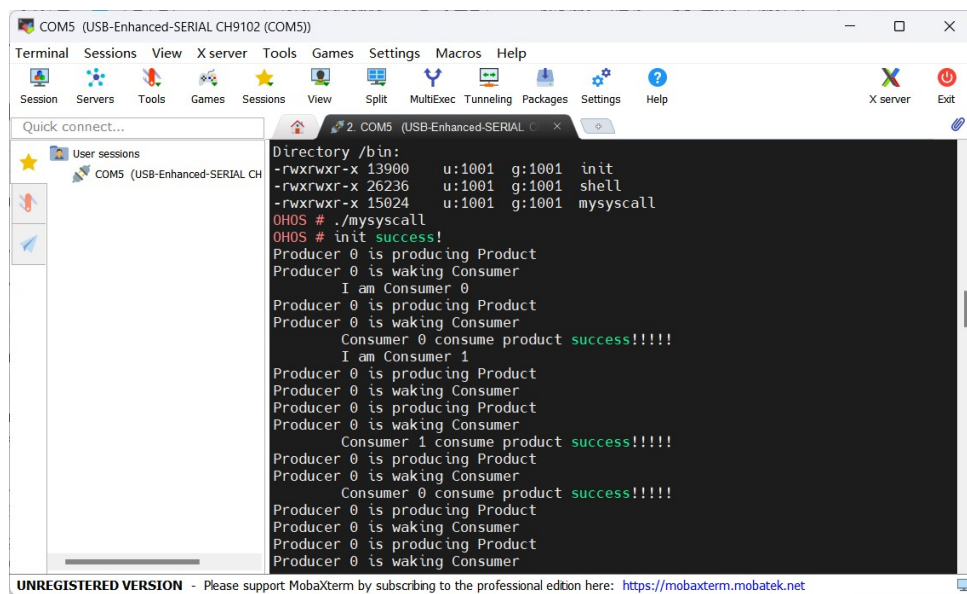
已经有产品资源的 `has_product` 信号，此后调用 `pthread_mutex_unlock` 解锁互斥锁，唤醒消费者线程。

5.在 Consumer 中，大体逻辑与生产者类似，不同的是，消费者一开始就需要调用 `pthread_cond_wait` 等待，直到 `has_product` 信号出现，代表已经有产品可以被使用，然后退出循环，进行消费，最后解锁。

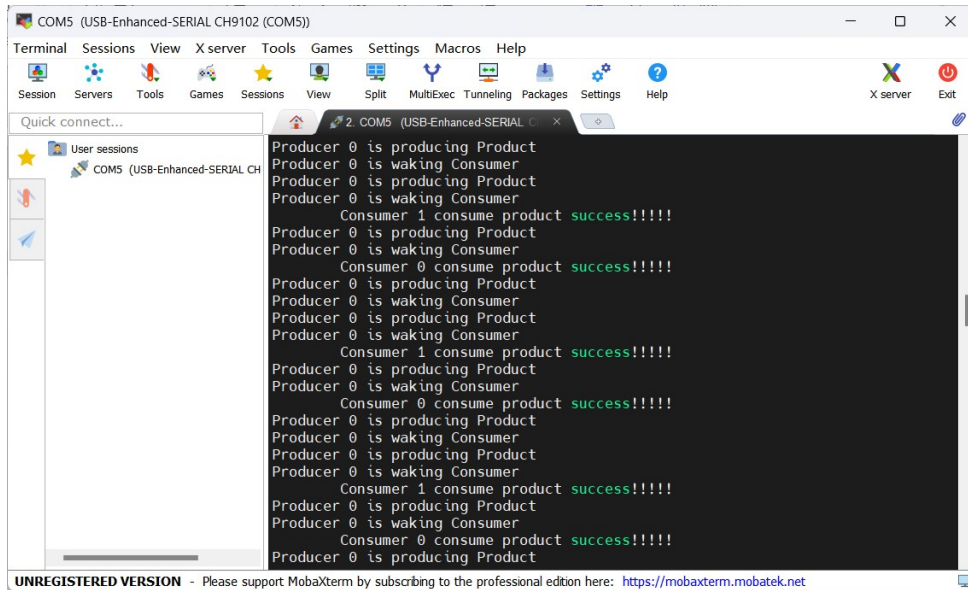
6.在 main 函数中，先初始化互斥锁和条件变量。然后创建指定数量的生产者线程和消费者线程，并传入对应的参数。生产者线程执行 `producer` 函数，消费者线程执行 `consumer` 函数。

7.最后使用 `pthread_join` 函数等待所有线程运行结束，并销毁互斥锁和条件变量。

4 实验结果



```
COM5 (USB-Enhanced-SERIAL CH9102 (COM5))
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help
Quick connect...
User sessions
COM5 (USB-Enhanced-SERIAL CH
Directory /bin:
-rwxrwxr-x 13900 u:1001 g:1001 init
-rwxrwxr-x 26236 u:1001 g:1001 shell
-rwxrwxr-x 15024 u:1001 g:1001 mysyscall
OHOS # ./mysyscall
OHOS # init success!
Producer 0 is producing Product
Producer 0 is waking Consumer
I am Consumer 0
Producer 0 is producing Product
Producer 0 is waking Consumer
Consumer 0 consume product success!!!!
I am Consumer 1
Producer 0 is producing Product
Producer 0 is waking Consumer
Producer 0 is producing Product
Producer 0 is waking Consumer
Consumer 1 consume product success!!!!
Producer 0 is producing Product
Producer 0 is waking Consumer
Consumer 0 consume product success!!!!
Producer 0 is producing Product
Producer 0 is waking Consumer
Producer 0 is producing Product
Producer 0 is waking Consumer
UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net
```



The screenshot shows a MobaXterm terminal window titled 'COM5 (USB-Enhanced-SERIAL CH9102 (COM5))'. The terminal displays the output of a C program implementing a producer-consumer algorithm. The output shows a sequence of messages: 'Producer 0 is producing Product', 'Producer 0 is waking Consumer', 'Consumer 1 consume product success!!!!', and 'Consumer 0 consume product success!!!!'. This cycle repeats several times. The terminal window has a menu bar with 'Terminal', 'Sessions', 'View', 'X server', 'Tools', 'Games', 'Settings', 'Macros', and 'Help'. A sidebar on the left shows 'Quick connect...' and 'User sessions'. At the bottom, a message reads: 'UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: <https://mobaxterm.mobatek.net>'.

```
COM5 (USB-Enhanced-SERIAL CH9102 (COM5))
Terminal Sessions View X server Tools Games Settings Macros Help
Session Servers Tools Games Sessions View Split MultiExec Tunneling Packages Settings Help
Quick connect...
User sessions
COM5 (USB-Enhanced-SERIAL CH
Producer 0 is producing Product
Producer 0 is waking Consumer
Producer 0 is producing Product
Producer 0 is waking Consumer
Consumer 1 consume product success!!!!
Producer 0 is producing Product
Producer 0 is waking Consumer
Consumer 0 consume product success!!!!
Producer 0 is producing Product
Producer 0 is waking Consumer
Producer 0 is producing Product
Producer 0 is waking Consumer
Consumer 1 consume product success!!!!
Producer 0 is producing Product
Producer 0 is waking Consumer
Consumer 0 consume product success!!!!
Producer 0 is producing Product
Producer 0 is waking Consumer
Producer 0 is producing Product
Producer 0 is waking Consumer
Consumer 1 consume product success!!!!
Producer 0 is producing Product
Producer 0 is waking Consumer
Consumer 0 consume product success!!!!
Producer 0 is producing Product
UNREGISTERED VERSION - Please support MobaXterm by subscribing to the professional edition here: https://mobaxterm.mobatek.net
```

5 实验分析

1. 实验结果中第一行 init success 指线程和互斥锁初始化成功。
2. 2-3 行 Product 0 is producing Product 表示生产者进行一次生产，Product 0 is waking Consumer 表示生产者开始唤醒消费者。
3. 第 4 行消费者被唤醒，由于消费者数量设置为两个，因此首先输出自己的消费者编号 I am Consumer 0。
4. 第 7 行 Consumer 0 consume product success 表示消费者成功进行一次消费。然后互斥锁解锁，生产者被唤醒。
5. 此后反复进行以上过程。
6. 整体来说，这段代码成功通过多线程和线程同步机制实现了生产者和消费者之间的协作，确保在有产品时消费者可以消费，没有产品时消费者等待，生产者生产完产品后通知消费者进行消费，无限循环。
7. 实验成功。

6 实验总结

本次实验主要是实现一个生产者-消费者模型，通过多线程和线程同步机制来模拟生产者和消费者之间的协作。在实验中，我们了解了互斥锁和条件变量的基本概念和使用方法，以及如何使用它们来保证线程之间的同步。

在实验过程中，我们发现线程同步是一个非常重要且复杂的问题，需要仔细考虑每个线程的执行顺序和对共享资源的访问方式，否则可能会出现各种意外情况。因此，在编写多线程程序时，需要谨慎设计线程之间的通信和同步方式，以确保程序的正确性和效率。

总的来说，本次实验让我们更深入地理解了多线程编程和线程同步机制的概念和应用，同时也让我们认识到了多线程编程的挑战和复杂性。这对我们今后进一步深入学习计算机系统和操作系统等内容非常有帮助。

7 参考文献

1.[美]William Stallings 著，陈向群，陈渝等译《操作系统——精髓与原理设计（第八版）》

8 附录

1. 生产者消费者代码

```
#include <stdlib.h>
```

```
#include <stdio.h>
```

```
#include <unistd.h>
```

```
#include <sys/mman.h>
```

```
#include <pthread.h>
```

```

#define CONSUMER_NUM 2

#define PRODUCER_NUM 1

pthread_t pids[CONSUMER_NUM+PRODUCER_NUM];

int ready = 0;

int running =1;

pthread_mutex_t mutex;

pthread_cond_t has_product;


void* producer(void* arg){
    int no = (int)arg;
    while(running){
        pthread_mutex_lock(&mutex);

        ready++;

        printf("Producer %d is producing Product\n",no);fflush(stdout);

        pthread_cond_signal(&has_product);

        printf("Producer %d is waking Consumer\n",no);fflush(stdout);

        pthread_mutex_unlock(&mutex);

        sleep(1);
    }
    return NULL;
}


void* consumer(void* arg){

```

```

int num = (int)arg;

while(running){

    pthread_mutex_lock(&mutex);

    while(ready==0){

        printf("\tConsumer %d is waiting...\n",num);fflush(stdout);

        pthread_cond_wait(&has_product,&mutex);

    }

    ready--;

    printf("\tConsumer %d consume product success!!!!\n",num);fflush(stdout);

    pthread_mutex_unlock(&mutex);

    sleep(3);

}

return NULL;

}

```

```

void HxSyscall(int num){

    pthread_mutex_init(&mutex,NULL);

    pthread_cond_init(&has_product,NULL);

    printf("init success!\n");

    int i;

    int thread_ids[CONSUMER_NUM + PRODUCER_NUM];

    for(i=0; i<PRODUCER_NUM; i++){

        thread_ids[i] = i;
    }
}

```



```
    pthread_create(&pids[i], NULL, producer, (void*)i);
}

for(i=0; i<CONSUMER_NUM; i++){
    printf("\tI am Consumer %d \n",i);fflush(stdout);

    sleep(2);

    thread_ids[PRODUCER_NUM + i] = i;

    pthread_create(&pids[PRODUCER_NUM + i], NULL, consumer, (void*)i);
}

for(i=0; i<PRODUCER_NUM + CONSUMER_NUM; i++){
    pthread_join(pids[i], NULL);
}

pthread_mutex_destroy(&mutex);
pthread_cond_destroy(&has_product);

return;
}
```