操作系统
实验报告

实验名称：<u>实验四 同步互斥问题</u>

姓名：刘亚辉

学号：16340157

实验名称：进程间通信和命令解释器

一、实验目的：

1. 学习利用互斥锁进行临界区互斥访问
2. 学习利用信号量实现同步

二、实验要求：

1. 利用进程同步机制，实现生产者-消费者问题
2. 实现读者-写者问题：读者优先、写者优先

三、实验过程：

1. 实现生产者-消费者问题：

思路：

首先，定义信号量，在该问题中，生产者需要在产品空位不为 0 时才可以生产新的产品，消费者需要在产品数量不为 0 的时候才可以消费产品，并且，生产者消费者对产品队列进行操作时，需要互斥进行。所以总共需要三个信号量，empty，full，mutex：当 empty=0 时，表示当前产品数量为 0；当 full!=0 时，表示当前有剩余产品；当 mutex 为 1 时，表示没有生产者/消费者正在操作产品队列，当 mutex 为 0 时，表示存在一个生产者/消费者正在操作产品队列。

生产者函数如下：

```cpp
void *produce(void *_producer){
    person *producer=(person*)_producer;
    sleep(producer->startTime);
    cout<<"The producer thread "<<producer->tid<<" produces an item "<<producer->itemId<<"."<<endl;

    sem_wait(&shared.empty);
    sem_wait(&shared.mutex);
    cout<<"The producer thread "<<producer->tid<<" adds the item "<<producer->itemId<<" to the buffer."<<endl;

    shared.items.push(producer->itemId);
    sleep(producer->duration);

    sem_post(&shared.mutex);
    sem_post(&shared.full);
    cout<<"The producer thread "<<producer->tid<<" ends producing."<<endl;
}
```

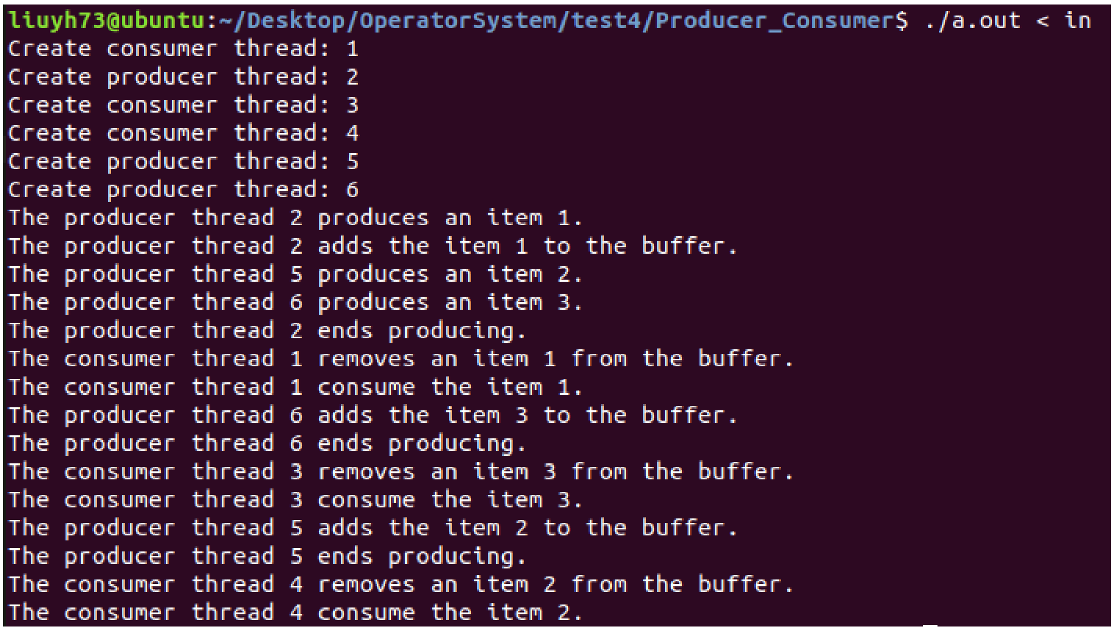
首先，等待一段时间，然后尝试进入临界区将生产的产品加入到产品队列中，此时，需要等待信号量 empty，只有当产品队列还有空位时，才可以进行该操作，然后等待互斥量 mutex，保证同一时刻只有一个线程可以操作产品队列。最后退出临界区是，释放 mutex，full，告知消费者现在有产品。

消费者函数如下：

```
void *consume(void *_consumer){
    person *consumer=(person*)_consumer;
    sleep(consumer->startTime);

    sem_wait(&shared.full);
    sem_wait(&shared.mutex);
    int itemId=shared.items.front();
    cout<<"The consumer thread "<<consumer->tid<<" removes an item "<<itemId<<" from the buffer."<<endl;

    shared.items.pop();
    sleep(consumer->duration);

    sem_post(&shared.mutex);
    sem_post(&shared.empty);
    cout<<"The consumer thread "<<consumer->tid<<" consume the item "<<itemId<<"."<<endl;
}
```

首先，和生产者类似，先等待一段时间，然后尝试获取产品，此时需要等待信号量 full，只有当产品队列不为空时，才可以进行获取产品的操作，然后等待互斥量 mutex，保证同一时刻只有一个线程操作产品队列。最后退出临界区，释放 mutex，empty，告知生产者现在存在产品空位。

运行结果如下：



```
liuyh73@ubuntu:~/Desktop/OperatorSystem/test4/Producer_Consumer$ ./a.out < in
Create consumer thread: 1
Create producer thread: 2
Create consumer thread: 3
Create consumer thread: 4
Create producer thread: 5
Create producer thread: 6
The producer thread 2 produces an item 1.
The producer thread 2 adds the item 1 to the buffer.
The producer thread 5 produces an item 2.
The producer thread 6 produces an item 3.
The producer thread 2 ends producing.
The consumer thread 1 removes an item 1 from the buffer.
The consumer thread 1 consume the item 1.
The producer thread 6 adds the item 3 to the buffer.
The producer thread 6 ends producing.
The consumer thread 3 removes an item 3 from the buffer.
The consumer thread 3 consume the item 3.
The producer thread 5 adds the item 2 to the buffer.
The producer thread 5 ends producing.
The consumer thread 4 removes an item 2 from the buffer.
The consumer thread 4 consume the item 2.
```

2. 实现读者-写者问题：

（1）读者优先：

首先，需要定义互斥量和信号量，reader 在尝试进入临界区时，需要确保当前没有 writer 正在临界区，这是需要一个临界区信号量 write_region；其次，由于 reader 之间进入临界区不互斥，则需要一个 reader 计数器，只需要在第一个 reader 到来时等待 write_region 即可，这是就需要一个互斥信号量来确保对 reader 计数器的操作的互斥性。所以，所需的信号量即为 write_region 和 reader_count_mutex，其中 reader_count_mutex 可以定义为线程锁。

读者函数如下：

```
void *read(void *_reader){
    person* reader=(person*)_reader;
    sleep(reader->startTime);

    cout<<"The reader thread "<<reader->tid<<" trys to read"<<endl;
    pthread_mutex_lock(&reader_count_mutex);
    shared.reader_count++;
    if(shared.reader_count==1)
        sem_wait(&shared.write_region);
    pthread_mutex_unlock(&reader_count_mutex);
    cout<<"The reader thread "<<reader->tid<<" is reading"<<endl;

    sleep(reader->duration);

    pthread_mutex_lock(&reader_count_mutex);
    shared.reader_count--;
    if(shared.reader_count==0)
        sem_post(&shared.write_region);
    pthread_mutex_unlock(&reader_count_mutex);
    cout<<"The reader thread "<<reader->tid<<" ends reading"<<endl;
}
```

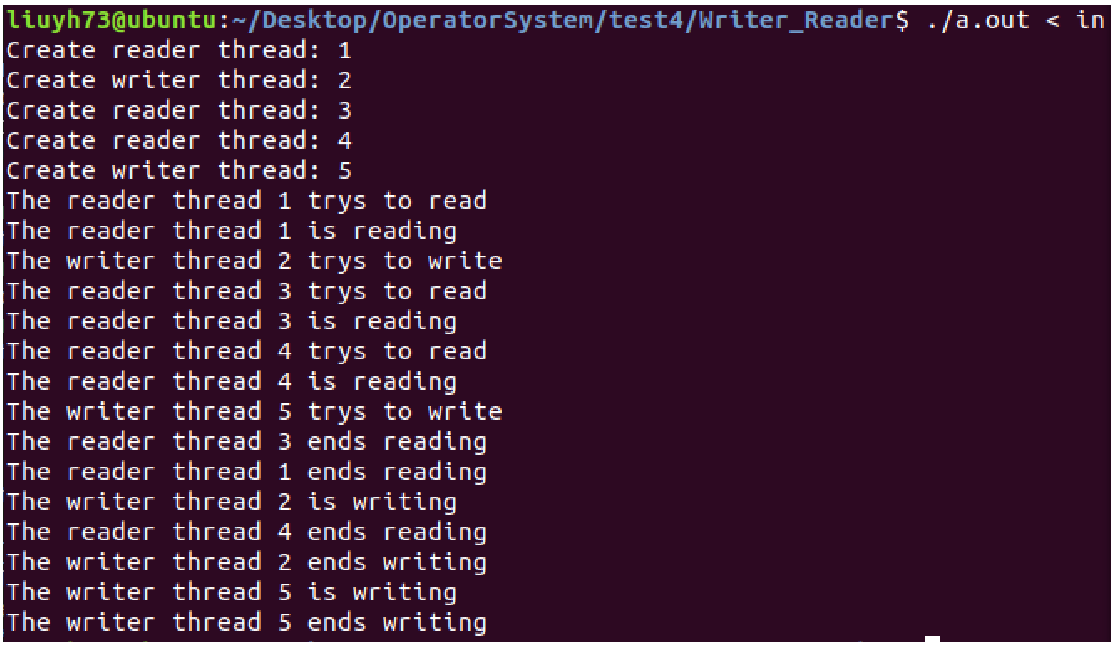
解释如下:每次读者到来时,需要计数器+1,这是互斥锁需要进行加锁操作,然后判断当前读者是否为第一个读者，如果是，则需要等待 write_region，因为读者之间对临界区的访问不互斥，所以后续读者不需要再等待 write_region，访问临界区结束后，对信号量的操作同理。

写者函数如下:

```
void *write(void *_writer){
    person* writer=(person*)_writer;
    sleep(writer->startTime);
    //pthread_mutex_lock(&shared.writer_count_mutex);
    //writer_count++;
    cout<<"The writer thread "<<writer->tid<<" trys to write"<<endl;
    sem_wait(&shared.write_region);
    cout<<"The writer thread "<<writer->tid<<" is writing"<<endl;
    sleep(writer->duration);
    cout<<"The writer thread "<<writer->tid<<" ends writing"<<endl;
    sem_post(&shared.write_region);
}
```

解释如下：写者函数比较简单，只需要在进入临界区前后等待和释放信号量 write_region 确保只有一个线程访问临界区即可。

运行结果如下:



```
liuyh73@ubuntu:~/Desktop/OperatorSystem/test4/Writer_Reader$ ./a.out < in
Create reader thread: 1
Create writer thread: 2
Create reader thread: 3
Create reader thread: 4
Create writer thread: 5
The reader thread 1 trys to read
The reader thread 1 is reading
The writer thread 2 trys to write
The reader thread 3 trys to read
The reader thread 3 is reading
The reader thread 4 trys to read
The reader thread 4 is reading
The writer thread 5 trys to write
The reader thread 3 ends reading
The reader thread 1 ends reading
The writer thread 2 is writing
The reader thread 4 ends reading
The writer thread 2 ends writing
The writer thread 5 is writing
The writer thread 5 ends writing
```

（2）写者优先

写者优先是指，当有写者处于等待进入临界区状态时，读者不可以进入临界区。 所以需要加一个信号量 read_permit 表示当前是否有写者处于临界区或者处于等待状态(是否允许读者进行读操作)，此信号量需要一个写者计数器来辅助使用，当写者计数器为 0 时，表示当前没有写者处于临界区或者处于等待状态，此时释放上述信号量 read_permit；当第一个写者等待进入临界区时，就申请资源 read_permit，使得不可以有新的读者进入临界区。同时，对写着计数器的修改也需要一个写者计数器的锁来进行互斥操作。

读者函数如下：

```
void *read(void *_reader){
    person* reader=(person*)_reader;
    sleep(reader->startTime);

    cout<<"The reader thread "<<reader->tid<<" trys to read until there are no writers waiting"<<endl;
    sem_wait(&shared.read_permit);
    pthread_mutex_lock(&reader_count_mutex);
    shared.reader_count++;
    if(shared.reader_count==1)
        sem_wait(&shared.write_region);
    pthread_mutex_unlock(&reader_count_mutex);
    sem_post(&shared.read_permit);

    cout<<"The reader thread "<<reader->tid<<" is reading"<<endl;
    sleep(reader->duration);

    pthread_mutex_lock(&reader_count_mutex);
    shared.reader_count--;
    if(shared.reader_count==0)
        sem_post(&shared.write_region);
    pthread_mutex_unlock(&reader_count_mutex);
    cout<<"The reader thread "<<reader->tid<<" ends reading"<<endl;
}
```

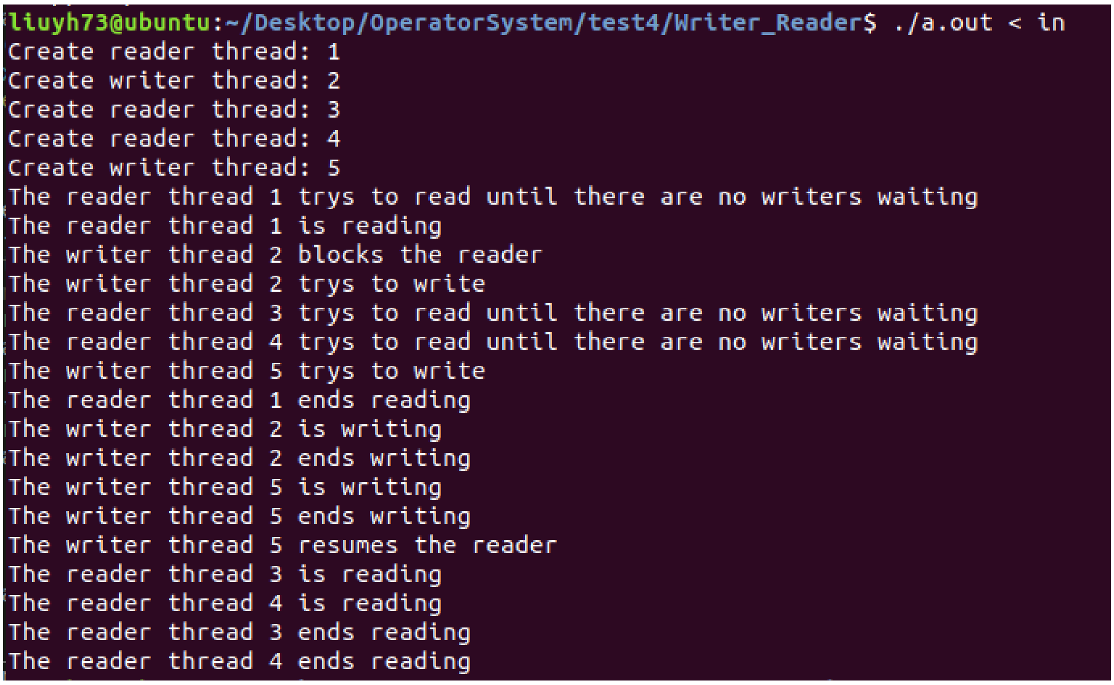
在上述代码中，在读者优先的基础上增加了等待 read_permit 和释放 read_permit 的操作，该操作需要在 reader 尝试进入临界区时执行，并在计数完成后释放，使得在没有 writer 等待的情况下，可以有多个 reader 同时进入临界区。当有 writer 处于等待状态时，除了已经在临界区内的 reader，不可以有新的 reader 进入临界区。

写者函数如下：

```
void *write(void *_writer){
    person* writer=(person*)_writer;
    sleep(writer->startTime);

    pthread_mutex_lock(&writer_count_mutex);
    shared.writer_count++;
    if(shared.writer_count==1){
        cout<<"The writer thread "<<writer->tid<<" blocks the reader"<<endl;
        sem_wait(&shared.read_permit);
    }
    pthread_mutex_unlock(&writer_count_mutex);

    cout<<"The writer thread "<<writer->tid<<" trys to write"<<endl;
    sem_wait(&shared.write_region);
    cout<<"The writer thread "<<writer->tid<<" is writing"<<endl;
    sleep(writer->duration);
    cout<<"The writer thread "<<writer->tid<<" ends writing"<<endl;
    sem_post(&shared.write_region);

    pthread_mutex_lock(&writer_count_mutex);
    shared.writer_count--;
    if(shared.writer_count==0){
       cout<<"The writer thread "<<writer->tid<<" resumes the reader"<<endl;
        sem_post(&shared.read_permit);
    }
    pthread_mutex_unlock(&writer_count_mutex);
}
```

写者函数由于增加了 writer 计数器，使得代码变多，具体操作与 reader 计数器操作一直，需要利用锁来进行互斥；在计数中，需要进行判断，如果是第一个写者，这需要调用 sem_wait(&shared.read_permit)来申请资源，如果是最后一个写者退出临界区，则需要调用 sem_post(&shared,read_permit)。

结果如下：



```
liuyh73@ubuntu:~/Desktop/OperatorSystem/test4/Writer_Reader$ ./a.out < in
Create reader thread: 1
Create writer thread: 2
Create reader thread: 3
Create reader thread: 4
Create writer thread: 5
The reader thread 1 trys to read until there are no writers waiting
The reader thread 1 is reading
The writer thread 2 blocks the reader
The writer thread 2 trys to write
The reader thread 3 trys to read until there are no writers waiting
The reader thread 4 trys to read until there are no writers waiting
The writer thread 5 trys to write
The reader thread 1 ends reading
The writer thread 2 is writing
The writer thread 2 ends writing
The writer thread 5 is writing
The writer thread 5 ends writing
The writer thread 5 resumes the reader
The reader thread 3 is reading
The reader thread 4 is reading
The reader thread 3 ends reading
The reader thread 4 ends reading
```