实验三 同步互斥问题

- 生产者-消费者问题
- 读者-写者问题

一、实验目的

1.用线程同步机制，实现生产者-消费者问题
2.用信号量机制分别实现读者优先和写者优先的读者-写者问题

二、实验运行环境

虚拟机VMware下的Ubuntu16.04系统

三、实验内容及要求

1.生产者-消费者问题
设计一个程序来解决有限缓冲问题。
在课本6.6.1小节中，使用了三个信号量：
①empty（以记录有多少空位）
②full（以记录有多少满位）
③mutex（二进制信号量或互斥信号量，以保护对缓冲插入与删除的操作）
对于本项目，empty与full将采用标准计数信号量，而mutex将采用二进制信号量。生产者与消费者作为独立线程，在empty、full、mutex的同步前提下，对缓冲进行插入与删除。
本项目中采用Pthread。

2.读者-写者问题
在linux环境下，创建一个进程，此进程包含n个线程。用着n个线程来表示n个读者或写者。每个线程按相应测试数据文件（后面有介绍）的要求进行读写操作。用信号量机制分别实现读者优先和写者优先的读者-写者问题。
读者-写者问题的读写操作限制（仅读者优先或写者优先）：
1）写-写互斥，即不能有两个写者同时进行写操作。
2）读-写互斥，即不能同时有一个线程在读，而另一个线程在写。
3）读-读允许，即可以有一个或多个读者在读。
读者优先的附加限制：如果一个读者申请进行读操作时已有另一个读者正在进行读操作，则该读者可直接开始读操作。
写者优先的附加限制：如果一个读者申请进行读操作时已有另一个写者在等待访问共享资源，则该读者必须等到没有写者处于等待状态后才能开始读操作。
运行结果显示要求：要求在每个线程创建、发出读写操作申请、开始读写操作和结束读写操作时分别显示一行提示信息，以确定所有处理都遵守相应的读写操作限制。

2.1读者优先问题
读者优先指的是除非有写者在写文件，否则读者不需要等待。所以可以用一个整型变量read_count记录当前的读者数目，用于确定是否需要释放正在等待的写者线程（当read_count=0时，表明所有的读者读完，需要释放写者等待队列中的一个写者）。每一个读者开始读文件时，必须修改read_count变量。因此需要一个互斥对象mutex来实现对全局变量read_count修改时的互斥。
另外，为了实现写-写互斥，需要增加一个临界区对象wrt。当写者发出写请求时，必须申请临界区对象的所有权。通过这种方法，也可以实现读-写互斥，当read_count=1时（即第一个读者到来时），读者线程也必须申请临界区对象的所有权。
当读者拥有临界区的所有权时，写者阻塞在临界区对象wrt上。当写者拥有临界区的所有权时，第一个读者判断完“read_count==1”后阻塞在wrt上，其余的读者由于等待对read_count的判断，阻塞在mutex上。
2.2写者优先问题
写者优先与读者优先类似。不同之处在于一旦一个写者到来，它应该尽快对文件进行写操作，如果有一个写者在等待，则新到来的读者不允许进行读操作。为此应当增加一个整型变量write_count，用于记录正在等待的写者的数目，当write_count=0时，才可以释放等待的读者线程队列。
为了对全局变量write_count实现互斥，必须增加一个互斥对象writeAccess。
为了实现写者优先，应当增加一个临界区对象mutexW，当有写者在写文件或等待时，读者必须阻塞在mutexW上。
读者线程除了要对全局变量read_count实现操作上的互斥外，还必须有一个互斥对象对阻塞read这一过程实现互斥。这两个互斥对象分别命名为mutexR和readAccess。

四、测试数据

1.生产者-消费者问题
测试数据文件包括n行测试数据，分别描述创建的n个线程是生产者还是消费者，以及生产者或消费者存放或取产品的开始时间和持续时间。
每行测试数据包括四个或五个字段，各个字段间用空格分隔。
第一字段为一个正整数，表示线程序号。
第二字段表示相应线程角色，P表示生产者，C表示消费者。
第三字段为一个正数，表示存放或取出操作的开始时间：线程创建后，延迟相应时间（单位为秒）后发出对共享资源的使用申请。
第四字段为一个正数，表示操作的持续时间。

第五字段为一个正数（仅生产者有），表示生产的产品号。
当线程申请成功后，开始对共享资源的操作，该操作持续相应时间后结束，并释放共享资源。
下面是一个测试数据文件的例子：

```
1 C 3 5
2 P 4 5 1
3 C 5 2
4 C 6 5
5 P 7 3 2
6 P 8 4 3
```

2.读者-写者问题
测试数据文件包括n行测试数据，分别描述创建的n个线程是读者还是写者，以及读写操作的开始时间和持续时间。
每行测试数据包括四个字段，各个字段用空格分隔。
第一字段为一个正整数，表示线程序号。
第二字段表示相应线程角色，R表示读者，W表示写者。
第三字段为一个正数，表示读写操作的开始时间：线程创建后，延迟相应时间（单位为秒）后发出对共享资源的读写申请。
第四字段为一个正数，表示读写操作的持续时间。
当线程读写申请成功后，开始对共享资源的读写操作，该操作持续相应时间后结束，并释放共享资源。
下面是一个测试数据文件的例子：

```
1 R 3 5
2 W 4 5
3 R 5 2
4 R 6 5
5 W 7 3
```

五、实验过程

1.生产者-消费者问题
①声明全局变量total_producer和total_consumer分别用于记录生产者总数和消费者总数，整型数组buffer用于作为缓冲区，存储生产者存放的产品，这里设置最大为4个缓冲区，nextP和nextC分别作为下一个生产者存放的产品和下一个消费者消费的产品。

```
int total_producer = 0, total_consumer = 0;// 生产者总数、消费者总数
int buffer[BUFFER_SIZE];// 生产者存放的产品
int nextP = 0, nextC = 0;// 下一个生产者存放的产品、下一个消费者消费的产品
```

②定义三个信号量：empty（生产者存放时进行阻塞，判断缓冲区是否有空位，若有则可以进行存放操作，若没有空位则必须等待），full（消费者消费时必须进行阻塞，判断缓冲区中是否有产品可以取出，若有则进行消费，若没有则必须等待），mutex（对缓冲区修改的互斥变量，要保证一次只能是生产者或者消费者其中一个缓冲区进行修改，不能两个角色同时修改缓冲区）；

```
sem_t empty, full, mutex;// 记录有多少空位、记录有多少满位、保护对缓冲区插入与成熟的操作
```

③定义一个结构体command，用于存放测试数据。

```
struct command
{
    int pid;// 线程号
    char type;// 线程角色（P：生产者；C：消费者）
    int startTime;// 操作开始的时间
    int lastTime;// 操作的持续时间
    int num;// 生产者存放产品的编号
};
```

④生产者线程：
1）将传入参数param转化为一个command的结构体，包含生产者的pid（线程号），type（即是P，线程角色：生产者），startTime（开始时间），lastTime（持续时间），num（生产者所生成的产品编号）。
2）进入while循环：
    开始阻塞信号empty，等待缓冲区是否有空位；
    睡眠sleep开始时间startTime，阻塞mutex信号，修改缓冲区，把生成的编号为data->num的产品放入缓冲区，打印出已完成生产的信息，修改缓冲区指针。
    睡眠sleep持续时间lastTime，释放信号量mutex和full，允许消费者对缓冲区进行修改和取出产品。
3）退出循环。

```
// 生产者线程
void *producer(void *param) {
    struct command* data = (struct command*)param;

    while (true) {
        sem_wait(&empty);
        sleep(data->startTime);
        sem_wait(&mutex);
```

```
        buffer[nextP] = data->num;
        cout << "Producer No." << data->pid
            << " produces " << "product No." << data->num << endl;
        nextP = (nextP + 1) % BUFFER_SIZE;

        sleep(data->lastTime);
        sem_post(&mutex);
        sem_post(&full);

        pthread_exit(0);
    }
}
```

⑤消费者线程：
1）将传入参数param转化为一个command的结构体（注意此时消费者没有num），包含生产者的pid（线程号），type（即是C，线程角色：消费者），startTime（开始时间），lastTime（持续时间）。
2）进入while循环：
开始阻塞信号full，等待缓冲区是否有产品；
睡眠sleep开始时间startTime，阻塞mutex信号，修改缓冲区，取出消费编号为buffer[nextC]的产品，打印出已完成消费的信息，修改缓冲区指针，缓冲区减少一个产品；
睡眠sleep持续时间lastTime，释放mutex和empty，允许生产者对缓冲区进行修改和向其中存放产品。
3）退出循环。

```
// 消费者线程
void *consumer(void *param) {
    struct command* data = (struct command*)param;

    while (true) {
        sem_wait(&full);
        sleep(data->startTime);
        sem_wait(&mutex);

        cout << "Consumer No." << data->pid
            << " consumes " << "product No." << buffer[nextC] << endl;
        buffer[nextC] = 0;
        nextC = (nextC + 1) % BUFFER_SIZE;

        sleep(data->lastTime);
        sem_post(&mutex);
        sem_post(&empty);

        pthread_exit(0);
    }
}
```

⑥main函数：
1）程序运行时，传入argv[1]表示生产者和消费者总数，也即生产者和消费者线程总数。
2）初始化各个信号量，数组结构体information存放所有的生产者或者消费者信息，包含生产者或者消费者的pid（线程号），type（线程角色：生产者或者消费者），startTime（开始时间），lastTime（持续时间），num（生产者所生成的产品编号，消费者没有）。

```
int total = atoi(argv[1]);
struct command information[total];
pthread_t Pid[total];

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

for (int i = 0; i < BUFFER_SIZE; i++) {
    buffer[i] = 0;
}
```

3）将测试数据文件导入，并将读取到的char类型的信息存放到结构体数组information中。
测试数据如下：

```
1 C 3 5
2 P 4 5 1
3 C 5 2
4 C 6 5
5 P 7 3 2
6 P 8 4 3
```

```
ifstream in("data.txt");
if (!in.is_open()) {
    exit(1);
}
```

```
char buffer[256];
int j = 0;
char ch;
while (!in.eof()) {
    in.read(&ch, 1);
    if (ch != ' ' && ch != '\n' && ch != '\r')
        buffer[j++] = ch;
}
j = 0;

for (int i = 0; i < total; i++) {
    information[i].pid = buffer[j++] - '0';
    information[i].type = buffer[j++];
    information[i].startTime = buffer[j++] - '0';
    information[i].lastTime = buffer[j++] - '0';
    if (information[i].type == 'P')
        information[i].num = buffer[j++] - '0';
}
```

4）生成生产者线程或者生成消费者线程：

```
for (int i = 0; i < total; i++) {
    if (information[i].type == 'P') {
        total_producer++;
        cout << "Create Producer No." << information[i].pid << endl;
        pthread_create(&Pid[i], NULL, producer, &information[i]);
    }

    if (information[i].type == 'C') {
        total_consumer++;
        pthread_create(&Pid[i], NULL, consumer, &information[i]);
        cout << "Create Consumer No." << information[i].pid << endl;
    }
}

for (int i = 0; i < total; i++) {
    pthread_join(Pid[i], NULL);
}
```

5）释放信号量empty、full、mutex。

```
sem_destroy(&full);
sem_destroy(&empty);
sem_destroy(&mutex);
```

⑦编译运行程序，得到结果如下（有多种情况）：
下面给出两种情况：

```
linjiafengyang@ubuntu:~/Desktop$ g++ -g Producer_and_Consumer.cpp -o main -lpthr
ead
linjiafengyang@ubuntu:~/Desktop$ ./main 6
Create Consumer No.1
Create Producer No.2
Create Consumer No.3
Create Consumer No.4
Create Producer No.5
Create Producer No.6
Producer No.2 produces product No.1
Producer No.5 produces product No.2
Producer No.6 produces product No.3
Consumer No.1 consumes product No.1
Consumer No.3 consumes product No.2
Consumer No.4 consumes product No.3
```

```
linjiafengyang@ubuntu:~/Desktop$ ./main 6
Create Consumer No.1
Create Producer No.2
Create Consumer No.3
Create Consumer No.4
Create Producer No.5
Create Producer No.6
Producer No.2 produces product No.1
Producer No.5 produces product No.2
Consumer No.1 consumes product No.1
Consumer No.3 consumes product No.2
Producer No.6 produces product No.3
Consumer No.4 consumes product No.3
```

2.读者-写者问题

2.1读者优先

①声明两个临界区对象writer和mutex，分别用于阻塞读写操作和改变读者数量。

②声明read_count用于记录读者的数量。

③定义一个结构体command用于测试数据的输入。其中pid表示线程号，type表示线程角色（R：读者；W：写者），startTime表示线程开始的时间，lastTime表示线程的持续时间。

```
int data = 0;
int read_count = 0;// 记录读者的数量
sem_t writer, mutex;// 临界区对象writer和mutex分别用于阻塞读写操作和改变读者数量

struct command
{
    int pid;// 线程号
    char type;// 线程角色（R：读者；W：写者）
    int startTime;// 操作开始的时间
    int lastTime;// 操作的持续时间
};
```

④写操作函数：随机写入一个0-MAX_RAND（1000）的操作数并打印出来，全局变量data记录这个随机数，用于读操作函数的打印。

```
// 写操作函数
void write() {
    int rd = rand() % MAX_RAND;
    cout << "Write data " << rd << endl;
    data = rd;
}
```

⑤读操作函数：打印在写操作中写入的data。

```
// 读操作函数
void read() {
    cout << "Read data " << data << endl;
}
```

⑥写者线程：

1）将传入参数param转化为一个command的结构体，包含写者的pid（线程号），type（即是W，线程角色：写者），startTime（开始时间），lastTime（持续时间）。

2）进入while循环：

睡眠sleep开始时间startTime；

打印出线程号为pid的线程正在申请资源信息，并开始阻塞writer；

打印出开始写的信息，并执行写操作函数write()，显示写入的信息；

睡眠sleep持续时间lastTime；

打印出结束写的信息并释放资源，解除阻塞信号writer；

3）退出循环。

```
// 写者线程
void *Writer(void *param) {
    struct command* c = (struct command*)param;
    while (true) {
        sleep(c->startTime);
        cout << "Writer(the " << c->pid << " pthread) requests to write." << endl;
        sem_wait(&writer);

        cout << "Writer(the " << c->pid << " pthread) begins to write." << endl;
        write();

        sleep(c->lastTime);
        cout << "Writer(the " << c->pid << " pthread) stops writing." << endl;
        sem_post(&writer);

        pthread_exit(0);
    }
}
```

⑦读者线程：

1）将传入参数param转化为一个command的结构体，包含读者的pid（线程号），type（即是R，线程角色：读者），startTime（开始时间），lastTime（持续时间）。

2）进入while循环

睡眠sleep开始时间startTime；

打印出线程号为pid的线程正在申请资源信息，并开始阻塞互斥信号mutex；

读者数量read_count加1，检查read_count，如果为1，那么阻塞writer。判断后释放互斥信号mutex。

打印出开始读取的信息并执行读函数read();
睡眠sleep持续时间lastTime;
打印出结束写的信息并释放资源，并阻塞互斥信号mutex；
读者数量read_count减1，检查read_count，如果为0，那么释放信号writer。判断后解除互斥信号mutex。
3）退出循环。

```
// 读者线程
void *reader(void *param) {
    struct command* c = (struct command*)param;
    while (true) {
        sleep(c->startTime);
        cout << "Reader(the " << c->pid << " pthread) requests to read." << endl;
        sem_wait(&mutex);

        read_count++;
        if (read_count == 1)
            sem_wait(&writer);
        sem_post(&mutex);

        cout << "Reader(the " << c->pid << " pthread) begins to read." << endl;
        read();

        sleep(c->lastTime);
        cout << "Reader(the " << c->pid << " pthread) stops reading." << endl;
        sem_wait(&mutex);

        read_count--;
        if (read_count == 0)
            sem_post(&writer);
        sem_post(&mutex);

        pthread_exit(0);
    }
}
```

⑧main函数：
1）程序运行时，传入argv[1]作为线程的总数量number_person，表示读者和写者的总数量。
2）初始化信号量writer、mutex，数组结构体information存放读者或者写者信息，包含读者或者写者的pid（线程号），type（线程角色：读者或者写者），startTime（开始时间），lastTime（持续时间）。

```
int number_person = atoi(argv[1]);
sem_init(&writer, 0, 1);
sem_init(&mutex, 0, 1);
struct command information[number_person];
pthread_t pid[number_person];
```

3）输入测试数据：

```
1 R 3 5
2 W 4 5
3 R 5 2
4 R 6 5
5 W 7 3
```

```
for (int i = 0; i < number_person; i++) {
    cin >> information[i].pid >> information[i].type
            >> information[i].startTime >> information[i].lastTime;
}
```

4）根据测试数据的第二字段type判断是读者线程还是写者线程，然后用pthread_create创建相应type的线程。

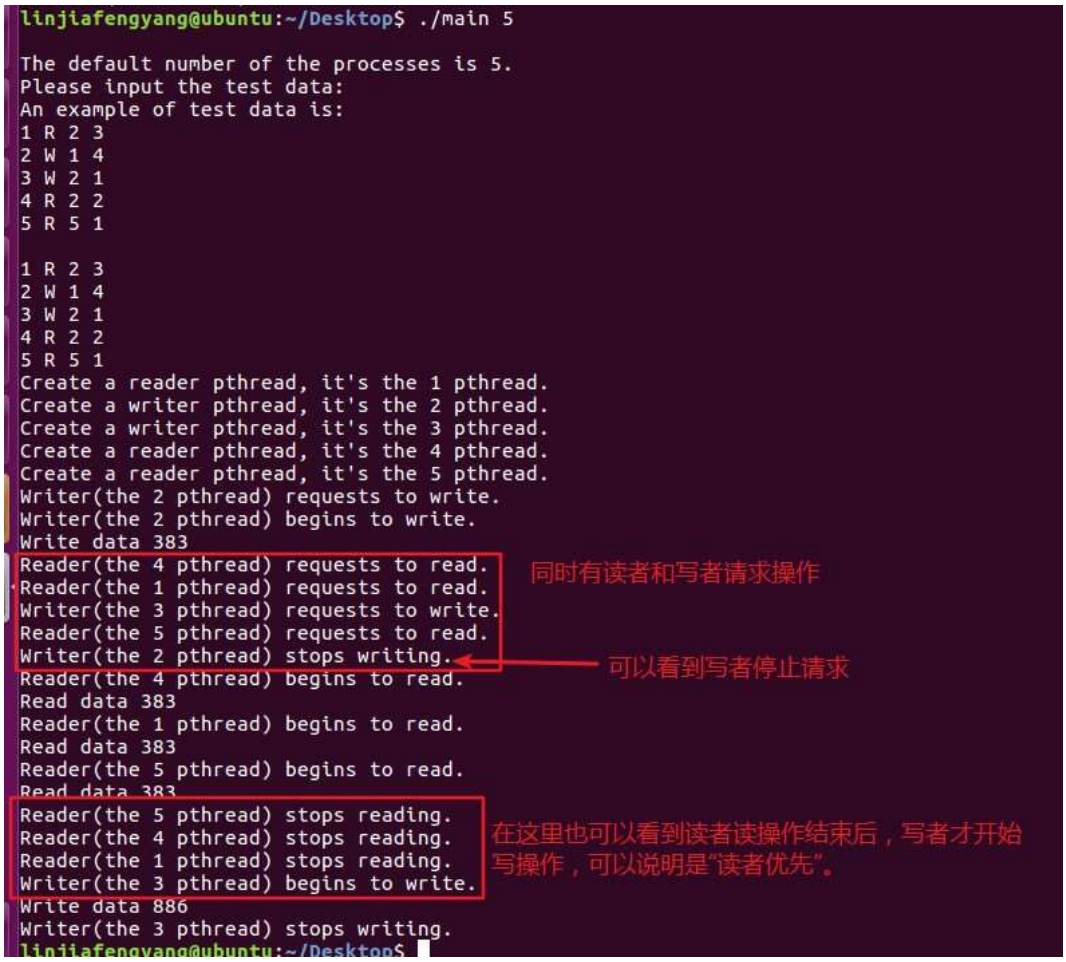
```
for (int i = 0; i < number_person; i++) {
    if (information[i].type == 'R') {
        cout << "Create a reader pthread, it's the " << information[i].pid << " pthread." << endl;
        pthread_create(&pid[i], NULL, reader, &information[i]);
    }

    if (information[i].type == 'W') {
        pthread_create(&pid[i], NULL, Writer, &information[i]);
        cout << "Create a writer pthread, it's the " << information[i].pid << " pthread." << endl;
    }
}

for (int i = 0; i < number_person; i++) {
```

```
    pthread_join(pid[i], NULL);
}
```

5）释放信号量writer和mutex。

```
sem_destroy(&writer);
sem_destroy(&mutex);
```

⑨编译运行，得到的结果如下：说明程序执行正确，实现了“读者优先”的机制。



2.2写者优先
①声明四个临界区对象writeAccess、readAccess、mutexR、mutexW；
②int类型read_count用于记录读者的数量，write_count用于记录写者的数量。
③定义一个结构体command用于测试数据的输入。其中pid表示线程号，type表示线程角色（R：读者；W：写者），startTime表示线程开始的时间，lastTime表示线程的持续时间。

```
int data = 0;
int read_count = 0, write_count = 0;// 记录读者的数量和写者的数量
// writeAccess:对全局变量write_count实现互斥
// readAccess：对全局变量read_count实现互斥
// mutexR：对阻塞read这一过程实现互斥
// mutexW：当有写者在写文件或者等待时，读者阻塞在mutexW上
sem_t writeAccess, readAccess, mutexR, mutexW;

struct command
{
    int pid;// 线程号
    char type;// 线程角色（R：读者；W：写者）
    int startTime;// 操作开始的时间
    int lastTime;// 操作的持续时间
};
```

④写操作函数以及读操作函数与2.1读者优先相同，这里不再赘述。

```
// 写操作函数
void write() {
```

```
    int rd = rand() % MAX_RAND;
    cout << "Write data " << rd << "." << endl;
    data = rd;
}

// 读操作函数
void read() {
    cout << "Read data " << data << "." << endl;
}
```

⑤写者线程：
1）将传入参数param转化为一个command的结构体，包含写者的pid（线程号），type（即是W，线程角色：写者），startTime（开始时间），lastTime（持续时间）。
2）进入while循环：
    睡眠sleep开始时间startTime；
    打印出线程号为pid的线程正在申请资源信息，并开始阻塞writerAccess；
    写者数量write_count加1，检查write_count，如果为1，那么阻塞mutexR。判断后释放互斥信号writeAccess；
    阻塞mutexW信号，打印出开始写的信息并执行写函数write()；
    睡眠sleep持续时间lastTime；
    打印出结束写的信息并释放资源，阻塞互斥信号mutexW；
    阻塞互斥信号writeAccess，读者数量write_count减1，检查write_count，如果为0，那么释放信号mutexR。判断后解除互斥信号writeAccess。
3）退出循环。

```
// 写者线程
void *writer(void *param) {
    struct command* c = (struct command*)param;
    while (true) {
        sleep(c->startTime);
        cout << "Writer(the " << c->pid << " pthread) requests to write." << endl;
        sem_wait(&writeAccess);

        write_count++;
        if (write_count == 1)
            sem_wait(&mutexR);
        sem_post(&writeAccess);

        sem_wait(&mutexW);
        cout << "Writer(the " << c->pid << " pthread) begins to write." << endl;
        write();

        sleep(c->lastTime);
        cout << "Writer(the " << c->pid << " pthread) stops writing." << endl;
        sem_post(&mutexW);

        sem_wait(&writeAccess);
        write_count--;
        if (write_count == 0)
            sem_post(&mutexR);
        sem_post(&writeAccess);

        pthread_exit(0);
    }
}
```

⑥读者线程：
1）将传入参数param转化为一个command的结构体，包含读者的pid（线程号），type（即是W，线程角色：读者），startTime（开始时间），lastTime（持续时间）。
2）进入while循环：
    睡眠sleep开始时间startTime；
    打印出线程号为pid的线程正在申请资源信息，并开始阻塞mutexR检查读者是否允许读取，同时也阻塞信号readAccess；
    读者数量read_count加1，检查read_count，如果为1，那么阻塞mutexW。判断后释放互斥信号readAccess和mutexR；
    打印出开始读取的信息并执行读函数read()；
    睡眠sleep持续时间lastTime；
    打印出结束写的信息；
    阻塞互斥信号readAccess，读者数量read_count减1，检查read_count，如果为0，那么释放信号mutexW。判断后解除互斥信号readAccess。
3）退出循环。

```
// 读者线程
void *reader(void *param) {
    struct command* c = (struct command*)param;
    while (true) {
        sleep(c->startTime);
        cout << "Reader(the " << c->pid << " pthread) requests to read." << endl;
        sem_wait(&mutexR);
        sem_wait(&readAccess);

        read_count++;
```

```
        if (read_count == 1)
            sem_wait(&mutexW);
        sem_post(&readAccess);
        sem_post(&mutexR);

        cout << "Reader(the " << c->pid << " pthread) begins to read." << endl;
        read();

        sleep(c->lastTime);
        cout << "Reader(the " << c->pid << " pthread) stops reading." << endl;

        sem_wait(&readAccess);
        read_count--;
        if (read_count == 0)
            sem_post(&mutexW);
        sem_post(&readAccess);

        pthread_exit(0);
    }
```

⑦main函数：
1）程序运行时，传入argv[1]作为线程的总数量number_person，表示读者和写者的总数量。
2）初始化信号量writeAccess、readAccess、mutexR、mutexW，数组结构体information存放读者或者写者信息，包含读者或者写者的pid（线程号），type（线程角色：读者或者写者），startTime（开始时间），lastTime（持续时间）。

```
int number_person = atoi(argv[1]);
sem_init(&writeAccess, 0, 1);
sem_init(&readAccess, 0, 1);
sem_init(&mutexR, 0, 1);
sem_init(&mutexW, 0, 1);

struct command information[number_person];
pthread_t pid[number_person];
```

3）输入测试数据：

```
1 R 3 5
2 W 4 5
3 R 5 2
4 R 6 5
5 W 7 3
```

```
for (int i = 0; i < number_person; i++) {
    cin >> information[i].pid >> information[i].type
        >> information[i].startTime >> information[i].lastTime;
}
```

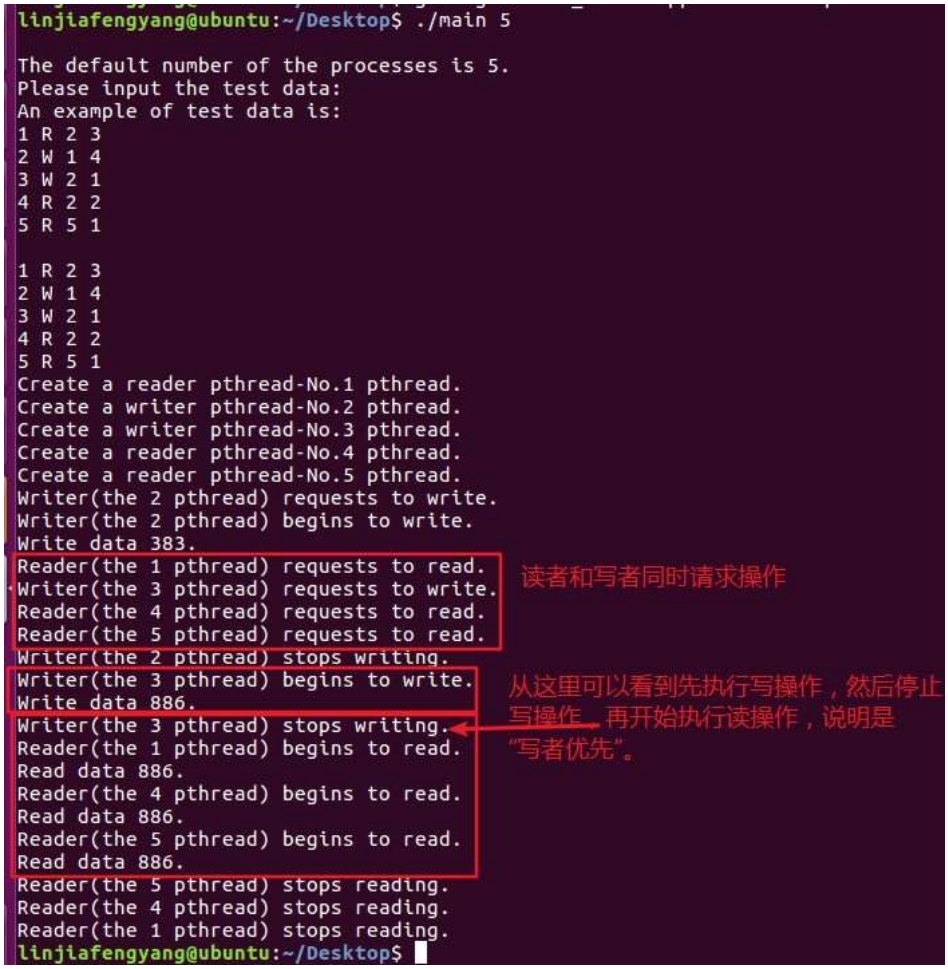
4）根据测试数据的第二字段type判断是读者线程还是写者线程，然后用pthread_create创建相应type的线程。

```
for (int i = 0; i < number_person; i++) {
    if (information[i].type == 'R') {
        cout << "Create a reader pthread-No." << information[i].pid << " pthread." << endl;
        pthread_create(&pid[i], NULL, reader, &information[i]);
    }

    if (information[i].type == 'W') {
        pthread_create(&pid[i], NULL, writer, &information[i]);
        cout << "Create a writer pthread-No." << information[i].pid << " pthread." << endl;
    }
}

for (int i = 0; i < number_person; i++) {
    pthread_join(pid[i], NULL);
}
```

5）释放信号量writeAccess、readAccess、mutexR、mutexW。

```
sem_destroy(&writeAccess);
sem_destroy(&readAccess);
sem_destroy(&mutexW);
sem_destroy(&mutexR);
```

⑧编译运行后，得到的结果如下：说明程序执行正确，实现了“写者优先”的机制。



六、实验结果及结论

根据上述实验过程以及实验结果分析，可以得出实验结果正确，满足了实验目的要求：
1.用线程同步机制，实现生产者-消费者问题
2.用信号量机制分别实现读者优先和写者优先的读者-写者问题

七、实验心得与体会

1.总的来说，这次实验算是这学期接触过的比较难的实验，主要的原因在于不熟悉Linux多线程编程下的线程互斥同步和信号量的使用，不理解pthread.h和semaphore.h这两个库中主要函数的使用，造成无法迅速读懂题意并实行有效编程。
2.好的一点是，老师给了相当多的实验指导材料和有效的指导，生产者-消费者问题在教材中有很大的篇幅描述，而读者-写者问题老师也作了详细的指导，将该问题分成两方面：读者优先和写者优先，为实验提供了很大的帮助和指导意义。
3.通过本次实验可以学习到多线程之间的互斥和同步问题，可以通过线程同步和互斥信号量来解决此类问题，加深了对线程之间工作机理的理解，更深刻地理解支持线程的操作系统的重大意义。