操作系统

实验报告

实验名称：实验四 同步互斥问题

姓名：陈亚楠

学号：16340041

实验名称：**同步互斥问题**

一、实验目的：

通过两个经典同步问题：生产者 - 消费者问题、读者 - 写者问题掌握采用信号量解决同步问题的方法。

二、实验要求：

1.生产者 - 消费者问题：

①设计一个程序来解决有限缓冲问题，其中的生产者与消费者进程如图 6.10 与图 6.11 所示。

②在 6.6.1 小节中，使用了三个信号量: empty (以记录有多少空位)、full (以记录有多少满位)以及 mutex (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。

③本项目，可采用 Pthread 。

2.读者 - 写者问题：

①在 Linux 环境下，创建一个进程，此进程包含 n 个线程。用这 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。

②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。

③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。

三、实验过程：

1．生产者 - 消费者问题

（1）定义缓冲区：

从内部来说,缓冲区是一个元数据类型为 buffer_item (可通过 typedef 来定义)的固定大小的数组。而从使用上来说，这个数组可按环形队列来处理。buffer_item 的定义及缓冲区大小可保存在头文件中，如下所示：

```c
// 缓冲区元数据 buffer_item 定义
typedef int buffer_item;
// 缓冲区大小
#define BUFFER_SIZE 5
```

缓冲区可通过如下两个函数来实现：inser_item 与 remove_item。这两个函数将为生产者和消费者线程所分别使用，其函数结构如下所示：

```c
int insert_item(buffer_item item)
{
    if (count <= BUFFER_SIZE) {
        buffer[rear] = item;
        rear = (rear + 1) % BUFFER_SIZE;
        count++;
        return 0;
    } else {
        return -1;
    }
}

int remove_item(buffer_item item)
{
    if (count == 0) {
        return -1;
    } else {
        item = buffer[head];
        head = (head + 1) % BUFFER_SIZE;
        count--;
        return 0;
    }
}
```

（2）声明定义测试数据的结构：

```c
// 测试数据结构
typedef struct
{
    pthread_t pthreadId;
    int sleepTime;
    int keepTime;
    buffer_item productId;
}data;
```

(3) 定义生产者与消费者线程：

①生产者线程：

```
    void *producer(void* param)
{
    data* pthread = (data*)param;
    pthread_t pthreadId = pthread->pthreadId;
    int sleepTime = pthread->sleepTime;
    int keepTime = pthread->keepTime;
    buffer_item productId = pthread->productId;
    free(pthread);
    while(true){
        sleep(sleepTime);
        sem_wait(&empty);
        sem_wait(&mutex);
        if (insert_item(productId)) {
            printf("Error, the buffer is full!\n");
            exit(-1);
        } else {
            printf("Producer pthread %ld produced product %d.\n", pthreadId,
productId);
        }
        sleep(keepTime);
        sem_post(&mutex);
        sem_post(&full);
        break;
    }
}
```

②消费者线程：

```
    void *consumer(void *param)
{
    data *pthread = (data *)param;
    pthread_t pthreadId = pthread->pthreadId;
    int sleepTime = pthread->sleepTime;
    int keepTime = pthread->keepTime;
```

```c
    buffer_item bufferItem;
    free(pthread);
    while(true){
        sleep(sleepTime);
        sem_wait(&full);
        sem_wait(&mutex);
        if (remove_item(bufferItem))
        {
            printf("Error, The buffer is empty!\n");
            exit(-1);
        }
        else
        {
            printf("Consumer pthread %ld consumed product.\n", pthreadId);
        }
        sleep(keepTime);
        sem_post(&mutex);
        sem_post(&empty);
        break;
    }
}
```

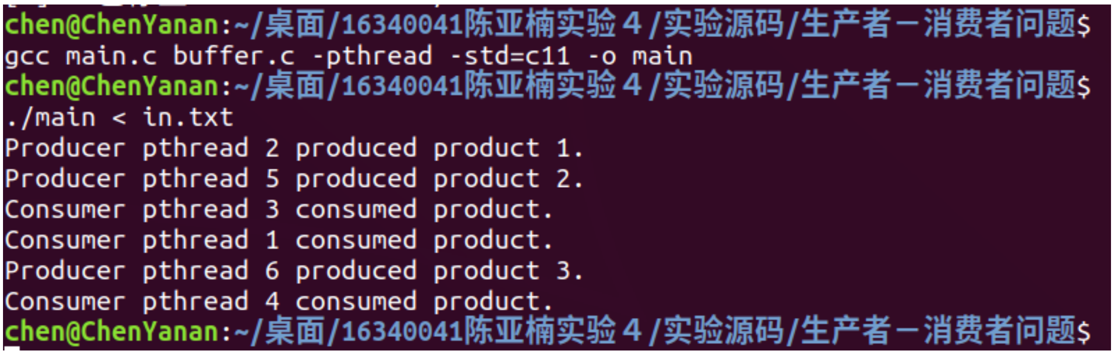
（4）定义创建线程函数：

```c
void createPthread() {
    pthread_t pthreadId;
    char pthreadRole;
    for(int i = 0; i < TESTNUMBER; i++)
    {
        scanf("%ld %c ", &pthreadId, &pthreadRole);
        data* pthread = malloc(sizeof(data));
        pthread->pthreadId = pthreadId;
        if(pthreadRole == 'C') {
            scanf("%d %d", &pthread->sleepTime, &pthread->keepTime);
            pthread_create(&pthreadId, &attr, consumer, pthread);
        }
        else if (pthreadRole == 'P') {
            scanf("%d %d %d", &pthread->sleepTime, &pthread->keepTime,
&pthread->productId);
            pthread_create(&pthreadId, &attr, producer, pthread);
        } else {
            printf("Invalid input!\n");
            exit(-1);
        }
    }
}
```

（5）主函数 main 将初始化缓冲、信号量，创建生产者与消费者线程。在创建完这些线程后，主函数 main 将睡眠一段时间，并在被唤醒的时候终止应用程序。主函数 main 的结构如下：

```
int main(int argc, char const *argv[])
{
    // 初始化信号量
    sem_init(&mutex, 0, 1);
    sem_init(&empty, 0, BUFFER_SIZE);
    sem_init(&full, 0, 0);

    pthread_attr_init(&attr);
    //pthread_t pthreadArray[TESTNUMBER];
    // 创建进程
    createPthread();

    sleep(60);

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

    return 0;
}
```

（6）实验结果：



```
chen@ChenYanan:~/桌面/16340041陈亚楠实验4/实验源码/生产者—消费者问题$
gcc main.c buffer.c -pthread -std=c11 -o main
chen@ChenYanan:~/桌面/16340041陈亚楠实验4/实验源码/生产者—消费者问题$
./main < in.txt
Producer pthread 2 produced product 1.
Producer pthread 5 produced product 2.
Consumer pthread 3 consumed product.
Consumer pthread 1 consumed product.
Producer pthread 6 produced product 3.
Consumer pthread 4 consumed product.
chen@ChenYanan:~/桌面/16340041陈亚楠实验4/实验源码/生产者—消费者问题$
```

2．读者 - 写者问题

（1） 读者优先：

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 write。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。

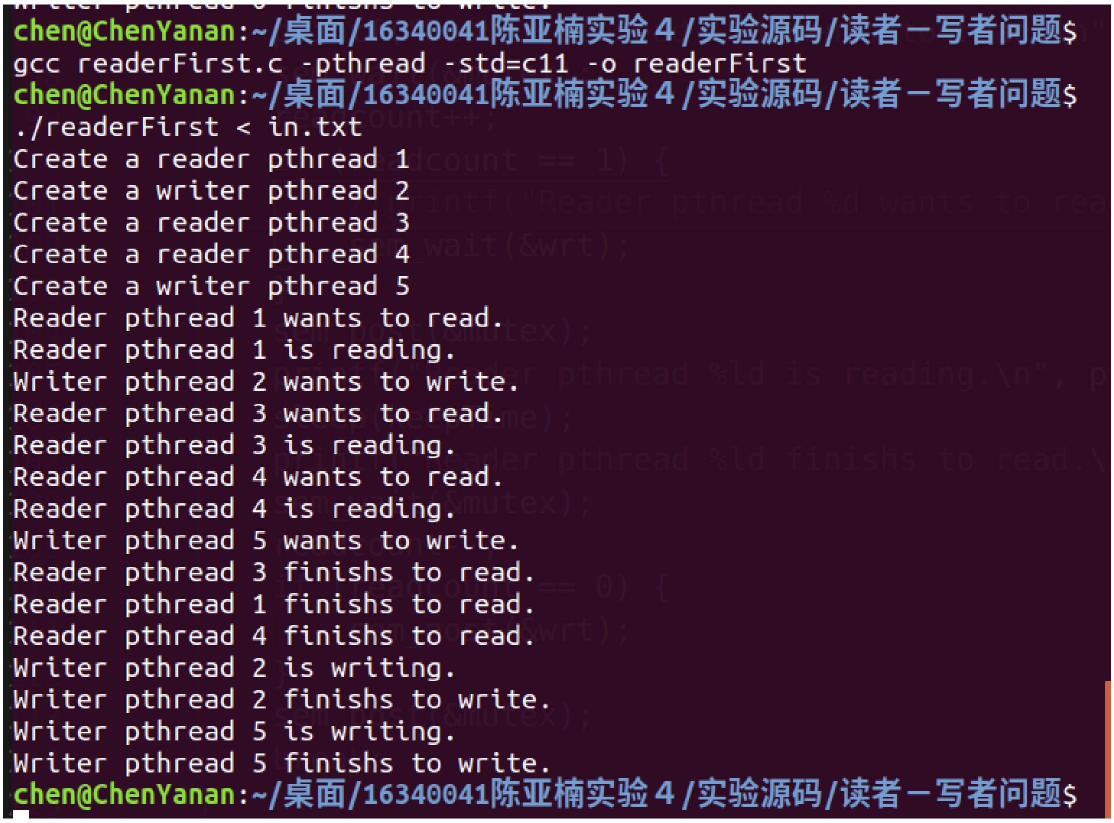
当读者拥有临界区的所有权时，写者阻塞在临界区对象 write 上。当写者拥有临界区的所有权时，第一个读者判断完“read_count==1”后阻塞在 write 上，其余的读者由于等待对 read_count 的判断，阻塞在 mutex 上。

这里仅强调读者、写者线程的定义：

① 读者：

```
void *reader(void *param)
{
    data *pthread = (data *)param;
    pthread_t pthreadId = pthread->pthreadId;
    int sleepTime = pthread->sleepTime;
    int keepTime = pthread->keepTime;
    free(pthread);
    while(true){
        sleep(sleepTime);
        printf("Reader pthread %ld wants to read.\n", pthreadId);
        sem_wait(&mutex);
        readcount++;
        if (readcount == 1) {
            sem_wait(&wrt);
        }
        sem_post(&mutex);
        printf("Reader pthread %ld is reading.\n", pthreadId);
        sleep(keepTime);
        printf("Reader pthread %ld finishs to read.\n", pthreadId);
        sem_wait(&mutex);
        readcount--;
        if (readcount == 0) {
            sem_post(&wrt);
        }
        sem_post(&mutex);
        break;
    }
}
```

实验结果：



```
chen@ChenYanan:~/桌面/16340041陈亚楠实验4/实验源码/读者－写者问题$
gcc readerFirst.c -pthread -std=c11 -o readerFirst
chen@ChenYanan:~/桌面/16340041陈亚楠实验4/实验源码/读者－写者问题$
./readerFirst < in.txt
Create a reader pthread 1
Create a writer pthread 2
Create a reader pthread 3
Create a reader pthread 4
Create a writer pthread 5
Reader pthread 1 wants to read.
Reader pthread 1 is reading.
Writer pthread 2 wants to write.
Reader pthread 3 wants to read.
Reader pthread 3 is reading.
Reader pthread 4 wants to read.
Reader pthread 4 is reading.
Writer pthread 5 wants to write.
Reader pthread 3 finishs to read.
Reader pthread 1 finishs to read.
Reader pthread 4 finishs to read.
Writer pthread 2 is writing.
Writer pthread 2 finishs to write.
Writer pthread 5 is writing.
Writer pthread 5 finishs to write.
chen@ChenYanan:~/桌面/16340041陈亚楠实验4/实验源码/读者－写者问题$
```

② 写者：

```
void *writer(void *param)
{
    data *pthread = (data *)param;
    pthread_t pthreadId = pthread->pthreadId;
    int sleepTime = pthread->sleepTime;
    int keepTime = pthread->keepTime;
    free(pthread);
    while(true){
        sleep(sleepTime);
        printf("Writer pthread %ld wants to write.\n", pthreadId);
        sem_wait(&wrt);
        printf("Writer pthread %ld is writing.\n", pthreadId);
        sleep(keepTime);
        printf("Writer pthread %ld finishs to write.\n", pthreadId);
        sem_post(&wrt);
        break;
    }
}
```

（2） 写者优先：

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。为此应当添加一个整型变量 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 mutex2。

为了实现写者优先，应当添加一个临界区对象 read，当有写者在写文件或等待时，读者必须阻塞在 read 上。同样，有读者读时，写者必须等待。于是，必须有一个互斥对象 RW_mutex 来实现这个互斥。

有写者在写时，写者必须等待。

读者线程要对全局变量 read_count 实现操作上的互斥，必须有一个互斥对象命名为 mutex1。

这里仅强调读者、写者线程的定义：

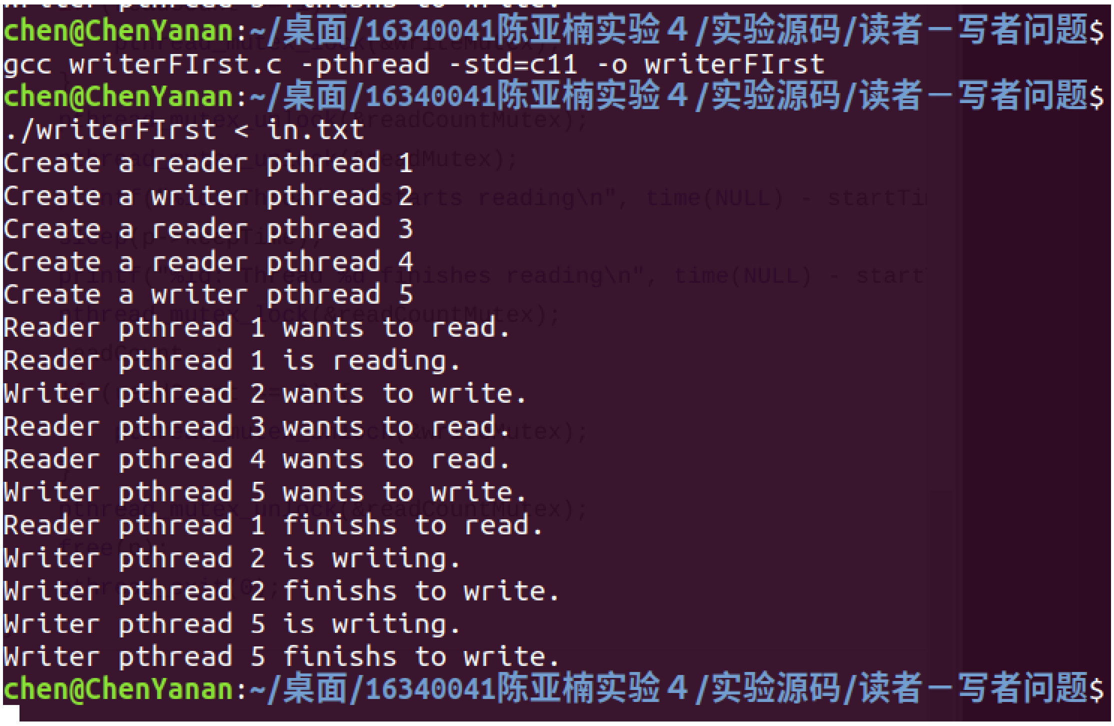
① 读者：

```
void *writer(void *param)
{
    data *pthread = (data *)param;
    pthread_t pthreadId = pthread->pthreadId;
    int sleepTime = pthread->sleepTime;
    int keepTime = pthread->keepTime;
    free(pthread);
    while (true)
    {
        sleep(sleepTime);
        printf("Writer pthread %ld wants to write.\n", pthreadId);
        sem_wait(&writeMutex);
        writecount++;
        if(writecount == 1) {
            sem_wait(&rd);
        }
        sem_post(&writeMutex);

        sem_wait(&wrt);
        printf("Writer pthread %ld is writing.\n", pthreadId);
        sleep(keepTime);
        printf("Writer pthread %ld finishs to write.\n", pthreadId);
        sem_post(&wrt);

        sem_wait(&writeMutex);
        writecount--;
        if (writecount == 0)
        {
            sem_wait(&rd);
        }
        sem_post(&writeMutex);
        break;
    }
}
```

② 写者：

```
void *reader(void *param)
{
    data *pthread = (data *)param;
    pthread_t pthreadId = pthread->pthreadId;
    int sleepTime = pthread->sleepTime;
    int keepTime = pthread->keepTime;
    free(pthread);
    while (true)
    {
        sleep(sleepTime);
        printf("Reader pthread %ld wants to read.\n", pthreadId);
        sem_wait(&rd);
        sem_wait(&readMutex);
        readcount++;
        if (readcount == 1)
        {
            sem_wait(&wrt);
        }
        sem_post(&readMutex);
        sem_post(&rd);
        printf("Reader pthread %ld is reading.\n", pthreadId);
        sleep(keepTime);
        printf("Reader pthread %ld finishs to read.\n", pthreadId);
        sem_wait(&readMutex);
        readcount--;
        if (readcount == 0)
        {
            sem_post(&wrt);
        }
        sem_post(&readMutex);
        break;
    }
}
```

实验结果：



```
Writer pthread 5 finishs to write.
chen@ChenYanan:~/桌面/16340041陈亚楠实验4/实验源码/读者－写者问题$
gcc writerFIrst.c -pthread -std=c11 -o writerFIrst
chen@ChenYanan:~/桌面/16340041陈亚楠实验4/实验源码/读者－写者问题$
./writerFIrst < in.txt
Create a reader pthread 1
Create a writer pthread 2
Create a reader pthread 3
Create a reader pthread 4
Create a writer pthread 5
Reader pthread 1 wants to read.
Reader pthread 1 is reading.
Writer pthread 2 wants to write.
Reader pthread 3 wants to read.
Reader pthread 4 wants to read.
Writer pthread 5 wants to write.
Reader pthread 1 finishs to read.
Writer pthread 2 is writing.
Writer pthread 2 finishs to write.
Writer pthread 5 is writing.
Writer pthread 5 finishs to write.
chen@ChenYanan:~/桌面/16340041陈亚楠实验4/实验源码/读者－写者问题$
```