写一份word文档，里面包括（设计思路、流程（原理）图、基本内容、源代码）章节。  
（2）设计思路：简单介绍生产者进程的功能以及消费者进程的功能。  
（3）流程（原理）图：绘制流程图或原理图。  
（4）基本内容：详细介绍生产者进程与消费者进程之间的同步与互斥关系。  
（5）源代码：列出源代码，也可以仅列出伪代码。

1.新建X个生产者进程和Y个消费者进程，他们共享一个Z大小的缓存区。

(X、Y分别创建进程,而缓存区的大小指的是生产者和消费者能存取的数据个数)

(生产者与生产者或生产者与消费者或消费与消费者之间有三种关系:互斥,同步,互斥与同步)

2.缓冲区的作用：

（1）生产者在缓存区没满的时候可以任意存入信息，他在缓存区满的时候只能等待消费者取出信息才能存入下一条信息。

（2）消费者取出缓冲区中自己需要的信息,取出完成后删除缓冲区信息然后生产者能向缓存区进行存入信息。

# 设计思路

本次设计定义的三个缓冲区是环形循环的,且需要按序访问.

生产者分配的缓冲区都是互斥的防止他们访问同一个内存进行操作,所以这两个生产者之间和两个消费者都是互斥的,,生产者生产完成后把锁的权限交于消费者进行产品消费.

# 流程(原理)图

# 基本内容

创建2个生产者进程和两个消费者进程，生产者1的作用是生成战斗过程消息，生产者2的作用生成战斗结果信息（他们是互斥的，每次只能存放一个信息，但是生产者1生成完战斗信息后将由生产者2生成战斗结果信息，生成者2生成完战斗结果后将由生成者1生成下次战斗过程），消费者1的作用是取出生成者1的战斗过程信息，消费者2的作用是取出消费者1的信息的同时取出生成者2的信息。程序通过linux的信号量实现多线程,通过sem\_init创建信号量判断运行条件,通过pthread\_create创建进程,创建的进程直到结束都会一直运行,创建4个死循环的进程达到4个线程不断运行,他们之间通过三对信号量进行判断阻塞和运行,通过pthread\_mutex\_lock进行锁定线程,一个进程只能有锁定的线程,所以实现线程之间顺序执行,执行完后通过pthread\_mutex\_unlock解锁进程并给信号量加或减值让其他线程停止阻塞并运行.

# 源代码

#include <stdio.h>

#include <pthread.h>

#include<semaphore.h>

sem\_t empty1,full1,empty2,full2,empty3,full3,empty4,full4; pthread\_mutex\_t mutex1 ;

int message1=0;

int message2=0;

int message3=0;

void \*producer1( void \*arg );

void \*producer2( void \*arg );

void \*consumerr1( void \*arg );

void \*consumerr2( void \*arg );

void \*producer1( void \*arg )

{

    while(1)

    {

        sem\_wait(&empty1);

        pthread\_mutex\_lock( &mutex1 );

        printf("producer1:input\_message1:");

        pthread\_mutex\_unlock( &mutex1 );

        sem\_post(&full1);

        sem\_post(&empty2);          1

    }

};

void \*producer2( void \*arg )

{

    while(1)

    {

        sem\_wait(&empty2);

        pthread\_mutex\_lock( &mutex1 );

        printf("producer2:input\_message2:");

        pthread\_mutex\_unlock( &mutex1 );

        sem\_post(&full2);

    }

};

void \*consumerr1( void \*arg )

{

    while(1)

    {

        sem\_wait(&full1);

        pthread\_mutex\_lock( &mutex1 );

        printf("consumerr1:ouput\_message1:%d",&message1);

        pthread\_mutex\_unlock( &mutex1 );

         sem\_post(&empty3);

        sem\_wait(&empty3);

        printf("consumerr1:input\_message2:");

        message3=message1;

        pthread\_mutex\_unlock( &mutex1 );

        sem\_post(&full3);

    }

};

void \*consumerr2( void \*arg )

{

    while(1)

    {

        sem\_wait(&full2);

        pthread\_mutex\_lock( &mutex1 );

       printf("consumerr2:ouput\_message1:%d",&message3);

        pthread\_mutex\_unlock( &mutex1 ); //解锁

        sem\_wait(&full3);  //empty-1

        pthread\_mutex\_lock( &mutex1 );

        printf("consumerr2:ouput\_message1:%d",&message2);

        pthread\_mutex\_unlock( &mutex1 ); //解锁

        //}

        sem\_post(&empty1);

    }

};

int main(char argc,char\* argv[])

{

    pthread\_t producter1,producter2,consumer1,consumer2;//创建四个对象两个生产者两个消费者

    pthread\_mutex\_init(&mutex1,NULL);

    sem\_init (&empty1, 0, 1);            //初始化empty1信号量//(信号对象,0多线程同步1多进程共享,初始值)

    sem\_init (&full1, 0, 0);            //初始化full1信号量

    //pthread\_mutex\_init(&mutex2,NULL);

    sem\_init (&empty2, 0, 0);            //初始化empty2信号量

    sem\_init (&full2, 0, 0);            //初始化full2信号量

    //pthread\_mutex\_init(&mutex3,NULL);

    sem\_init (&empty3, 0, 0);            //初始化empty2信号量

    sem\_init (&full3, 0, 0);            //初始化full2信号量

        //pthread\_mutex\_init(&mutex4,NULL);

    sem\_init (&empty4, 0, 0);            //初始化empty2信号量

    sem\_init (&full4, 0, 0);            //初始化full2信号量

    if( pthread\_create( &producter1 , NULL, producer1 ,  //创建生产者和消费者线程

                NULL ) != 0 )

        printf( "thread create failed." );

    if( pthread\_create( &producter2 , NULL, producer2 ,

                NULL ) != 0 )

        printf( "thread create failed." );

    if( pthread\_create( &consumer1 , NULL, consumerr1 ,

                NULL ) != 0 )

        printf( "thread create failed." );

    if( pthread\_create( &consumer2 , NULL, consumerr2 ,

                NULL ) != 0 )

        printf( "thread create failed." );

//等待线程结束

    if( pthread\_join( producter1 , NULL ) != 0 )

        printf( " wait thread failed.");

    if( pthread\_join( producter2 , NULL ) != 0 )

        printf( " wait thread failed.");

            if( pthread\_join( consumer1 , NULL ) != 0 )

        printf( " wait thread failed.");

    if( pthread\_join( consumer2 , NULL ) != 0 )

        printf( " wait thread failed.");

      sem\_destroy (&full1);               //释放同步量

      sem\_destroy(&empty1);            //释放同步量

      sem\_destroy (&full2);               //释放同步量

      sem\_destroy(&empty2);            //释放同步量

      sem\_destroy(&empty3);

      sem\_destroy(&full3);

    sem\_destroy(&empty4);

      sem\_destroy(&full4);

    pthread\_mutex\_destroy( &mutex1 );        //关闭互斥量

        return 0;

}