# 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.

生产者和消费者进程共用一个thread\_info结构体,这个结构体有有四个成员:进程id,进程类型,进程状态,进程存储信息.

生产者1的作用是保存战斗信息并放进缓冲区1的同时开辟缓冲区2,生产者2的作用是将战斗结果放进缓冲区2,消费者1的作用是将缓冲区1的战斗信息读出来,然后开辟缓冲区3写入战斗信息,消费者2的作用是将缓冲区2的消息读出来并将缓冲3的信息读出来,最后将缓冲区恢复为初始化状态进行下一次循环.

# 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，且生产者1生成完战斗信息后将由生产者2生成战斗结果信息，同时生成者2生成完战斗结果后将由生成者1生成下次战斗过程,他们也是按顺序存放的），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(他们是同步且互斥的必须消费者1先拿出生产者1的信息,再将信息保存起来,最后后消费者2拿出生产者2的信息,并拿出消费者1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.

程序首先将初始化并创建好进程,并将三个缓冲区分别置((0,1)(0,0)(0,0)).首先生产者1因为缓冲区1为空所以sem\_wait停止阻塞程序,而缓冲区2还未开辟为0所以sem\_wait先阻塞程序(这样实现排斥1),当生产者1执行完操作后将缓冲区1置为满,同时开辟缓冲区2的空位,生产者2的sem\_wait将不再阻塞程序,同时因为缓冲区1置为1生产者1就阻塞了(排斥2),运行完后将缓冲区2置于1.(排斥1和排斥2共同实现互斥),

当消费者1sem\_wait到缓冲1为1时马上读取生产者1的消息(同步),完成自己的操作后开辟缓冲区3(顺序)同时完成操作将缓冲区1置0、缓冲区3置1.

消费者2sem\_wait读到缓冲区2为1时马上读取生成者2的消息(同步)然后sem\_wait缓冲区3置1的时候马上读取消费者1的消息(顺序)然后输出读取到的消息,最后将缓冲区1置0,让sem信号量变成初始状态程序将进入下一个循环(实现环形循环).

次数 空/满 空/满 空/满

初始 1 0 0 0 0 0

第一次 0 1 1 0 0 0

第二次 0 1 0 1 0 0

第三次 0 0 0 1 1 0

第三次 0 0 0 1 0 1

第四次 0 0 0 0 0 0

第四次 1 0 0 0 0 0

开始下一次循环

# 源代码

#include <stdio.h>

#include <pthread.h>

#include <semaphore.h>

#include <unistd.h>

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

pthread\_mutex\_t mutex1 ;

void \*producer1( void \*arg );

void \*producer2( void \*arg );

void \*consumerr1( void \*arg );

void \*consumerr2( void \*arg );

struct thread\_info

{

int thread\_id;

char Type;

int status;

int message;

};

struct thread\_info produ1,produ2,comsu1,comsu2;

void \*producer1( void \*arg )

{

while(1)

{ sleep(1);

sem\_wait(&empty1);

pthread\_mutex\_lock( &mutex1 );

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

int message=scanf("%d",&produ1.message);

int id=1;

produ1.thread\_id=1;

char Type='p';

produ1.Type='p';

int status=1;

produ1.status=1;

printf("id=%d \n\r Type=%c \n\r status=%d \n\r ",id,Type,status);

pthread\_mutex\_unlock( &mutex1 );

status=0;

produ1.status=0;

printf("id=%d \n\r Type=%c \n\r status=%d \n\r ",id,Type,status);

sem\_post(&full1);

sem\_post(&empty2);

}

};

void \*producer2( void \*arg )

{

while(1)

{

sleep(1);

sem\_wait(&empty2);

pthread\_mutex\_lock( &mutex1 );

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

int message=scanf("%d",&produ2.message);

int id=2;

produ2.thread\_id=2;

char Type='p';

produ2.Type='p';

int status=1;

produ2.status=1;

printf("id=%d \n\r Type=%c \n\r status=%d \n\r",id,Type,status);

pthread\_mutex\_unlock( &mutex1 );

status=0;

produ2.status=0;

printf("id=%d \n\r Type=%c \n\r status=%d \n\r",id,Type,status);

pthread\_mutex\_unlock( &mutex1 );

sem\_post(&full2);

}

};

void \*consumerr1( void \*arg )

{

while(1)

{

sleep(1);

sem\_wait(&full1);

pthread\_mutex\_lock( &mutex1 );

int message=produ1.message;

printf("consumerr1:ouput\_message1:%d\n\r",message);

int id=3;

comsu1.thread\_id=3;

char Type='c';

comsu1.Type='c';

int status=1;

comsu1.status=1;

printf("id=%d \n\r Type=%c \n\r status=%d \n\r",id,Type,status);

pthread\_mutex\_unlock( &mutex1 );

status=0;

comsu1.status=0;

printf("id=%d \n\r Type=%c \n\r status=%d \n\r",id,Type,status);

sem\_post(&empty3);

sem\_wait(&empty3);

printf("consumerr1:input\_message2:%d\n\r",message);

status=1;

comsu1.message=message;

printf("id=%d \n\r Type=%c \n\r status=%d \n\r",id,Type,status);

status=0;

comsu1.status=0;

printf("id=%d \n\r Type=%c \n\r status=%d \n\r",id,Type,status);

pthread\_mutex\_unlock( &mutex1 );

sem\_post(&full3);

}

};

void \*consumerr2( void \*arg )

{

while(1)

{

sleep(1);

sem\_wait(&full2);

pthread\_mutex\_lock( &mutex1 );

int message1=produ2.message;

printf("consumerr2:ouput\_message1:%d\n\r",message1);

int id=4;

comsu2.thread\_id=4;

char Type='c';

comsu2.Type='c';

int status=1;

comsu2.status=1;

printf("id=%d \n\r Type=%c \n\r status=%d \n\r",id,Type,status);

pthread\_mutex\_unlock( &mutex1 );

status=0;

comsu2.status=status;

printf("id=%d \n\r Type=%c \n\r status=%d \n\r",id,Type,status);

sem\_wait(&full3); //empty-1

pthread\_mutex\_lock( &mutex1 );

int message2=comsu1.message;

printf("consumerr2:ouput\_message2:%d \n\r",message2);

status=1;

comsu2.status=status;

printf("id=%d \n\r Type=%c \n\r status=%d \n\r",id,Type,status);

status=0;

comsu2.status=status;

printf("id=%d \n\r Type=%c \n\r status=%d \n\r",id,Type,status);

pthread\_mutex\_unlock( &mutex1 );

sem\_post(&empty1);

}

};

int main(char argc,char\* argv[])

{

pthread\_t producter1,producter2,consumer1,consumer2;

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

sem\_init (&empty1, 0, 1);

sem\_init (&full1, 0, 0);

sem\_init (&empty2, 0, 0);

sem\_init (&full2, 0, 0);

sem\_init (&empty3, 0, 0);

sem\_init (&full3, 0, 0);

sem\_init (&empty4, 0, 0);

sem\_init (&full4, 0, 0);

if( pthread\_create( &producter1 , NULL, producer1 , 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." );

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);

sem\_destroy(&empty1);

sem\_destroy (&full2);

sem\_destroy(&empty2);

sem\_destroy(&empty3);

sem\_destroy(&full3);

sem\_destroy(&empty4);

sem\_destroy(&full4);

pthread\_mutex\_destroy( &mutex1 );

return 0;

}