实验说明文档

1. **程序改动思路**

在没有对程序进行修改的情况下，头文件Queue.h中会出现报错，原因是未定义boolean，因此首先要将所有涉及Boolean的代码修改为Bool类型（Queue.c中也要做相应的修改）。在运行程序时，直接对exp1.c进行编译，则无法得到正确的结果，此处需要使用针对线程的特定语句gcc -pthread \*.c ，该语句的含义为，同时对该文件夹下所有的.c文件进行编译（需要事先将所有文件放在同一文件夹下否则依旧得不到正确结果）。

解决上述问题后，编译运行程序，考虑对结果进行改进。在不对exp1.c修改的情况下，程序运行的结果中，每当生产者向缓冲区写入数据时，消费者并不能紧随其后读出刚刚写入的数据。这是因为只有当缓冲区满时消费者才能够从缓冲区读出数据，而生产者也只有在缓冲区空时才可以写入。针对此情况，作出如下优化：首先需要使得生产者和消费者线程数达到平衡，而程序中仅给出三个生产者线程，两个消费者线程，因此需要增加一个消费者线程。此外，还需要保证生产者和消费者sleep的时值相同。综合以上改动，基本能够实现生产者写入缓冲区后，消费者立即读出刚刚写入的数据。

1. **代码修改情况**

**Exp1.c**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

#include <errno.h>

#include <sys/ipc.h>

#include <semaphore.h>

#include <fcntl.h>

#include "Queue.h"

#include "const.h"

#define N 5

time\_t end\_time;

sem\_t mutex,full,empty;

int fd;

Queue \* qt;//缓冲区

Elemtype p;

void\* consumer(void \*arg);

void\* productor(void \*arg);

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

{

**pthread\_t id1,id2,id3,id4,id5,id6;**

pthread\_t mon\_th\_id;

int ret;

end\_time = time(NULL)+30;

qt = InitQueue();

p.lNumber = 1000;

ret=sem\_init(&mutex,0,1);

//初使化互斥信号量为１

ret=sem\_init(&empty,0,N);

//初使化empty信号量为N

ret=sem\_init(&full,0,0);

//初使化full信号量为０

if(ret!=0)

perror("sem\_init");

ret=pthread\_create(&id1,NULL,&productor, NULL);//生产者线程

if(ret!=0)

perror("pthread cread1");

ret=pthread\_create(&id3,NULL,&productor, NULL);//生产者线程

if(ret!=0)

perror("pthread cread3");

ret=pthread\_create(&id2,NULL,&consumer, NULL);//消费者线程

if(ret!=0)

perror("pthread cread2");

ret=pthread\_create(&id5,NULL,&productor, NULL);//生产者线程

if(ret!=0)

perror("pthread cread5");

ret=pthread\_create(&id4,NULL,&consumer, NULL);//消费者线程

if(ret!=0)

perror("pthread cread4");

**ret=pthread\_create(&id6,NULL,&consumer, NULL);**

**//增加一个消费者线程，使生产者和消费者线程数量达到平衡**

**if(ret!=0)**

**perror("pthread cread6");**

pthread\_join(id1,NULL);

pthread\_join(id2,NULL);

pthread\_join(id3,NULL);

pthread\_join(id4,NULL);

pthread\_join(id5,NULL);

**pthread\_join(id6,NULL);**

exit(0);

}

void\* productor(void \*arg)

//生产者线程

{

int i,nwrite;

while(time(NULL) < end\_time)

{

sem\_wait(&empty);

// p(empty)

sem\_wait(&mutex);

// p(mutex)

if(TRUE==QueueFull(\*qt))

//队满

{

printf("缓存区已满，无法写入\n");

}

else

{

EnQueue(qt,p);

printf("生产者向缓存区写入 [%d]\n",p.lNumber);

p.lNumber++;

}

**//V操作的顺序不固定，可以随意互换**

**sem\_post(&mutex);//v(mutex)**

**sem\_post(&full);//v(full)**

sleep(1);

}

}//生产者productor

void\* consumer(void \*arg)

//消费者线程

{

int nolock=0;

int ret,nread;

Elemtype p2;

while((time(NULL) < end\_time)||(FALSE==(QueueEmpty(\*qt))))

{

sem\_wait(&full);//p(full)

sem\_wait(&mutex);//p(mutex)

if(TRUE==QueueEmpty(\*qt))

//队空

{

printf("缓存区空，无数据可读\n");

}

else

{

DeQueue(qt,&p2);

printf("消费者从缓存区中读出 [%d] \n",p2.lNumber);

}

sem\_post(&empty);

//v(empty)

sem\_post(&mutex);

//v(mutex)

**sleep(1);//修改sleep时值为1，与生产者保持一致**

}

}

//end of while((time(NULL) < end\_time)||(FALSE==(QueueEmpty(\*qt))))

//消费者consumer

**Queue.h**

#ifndef QUEUE\_H\_LY

#define QUEUE\_H\_LY

#include "Typedefine.h"

Queue \* InitQueue();

//初使化队列信息

int EnQueue(Queue \*Q,Elemtype x);

//进队（因为没有队满的情况,所以没有返回值）

int DeQueue(Queue \*Q,Elemtype \*x);

//出队，x 为出队结点的数据域，返回 1 成功/0 失败

int QueueEmpty(Queue Q);

//判断队列空 1（不空）/0（为空）

int QueueCount(Queue \*HQ);

//统计一个栈链中的元素的个数

**BOOL QueueFull(Queue Q);//使用bool变量，boolean在此处无法运行**

// 判断队列是否为满，满返回 TRUE，非满返回 FALSE

#endif

**Queue.c**

//队列的基本操作函数

#include "stdio.h"

#include "malloc.h"

#include "const.h"

#include "Queue.h"

Queue \* InitQueue( )

//初使化队列

{

Queue \* Q = (Queue \*)malloc(sizeof(Queue));

Q->front = Q->rear = 0;

return Q;

}

int EnQueue(Queue \*Q,Elemtype x)

//进队

{

if((Q->rear + 1)%MaxSize==Q->front)

//队满

return FALSE;

else

{

Q->q[Q->rear] = x;

Q->rear = (Q->rear+1)%MaxSize;

return TRUE;

}

}

int DeQueue(Queue \*Q,Elemtype \*x)

//出队

{

if(Q->rear==Q->front)

//队空

return FALSE;

else

{

\*x = Q->q[Q->front];

Q->front = (Q->front+1)%MaxSize;

return TRUE;

}

}

int QueueEmpty(Queue Q)

//判断队是否为空，空返回0 非空返回 1

{

if(Q.rear==Q.front)

return TRUE;

else

return FALSE;

//队空

}

Elemtype Last(Queue \*Q)

//返回队列中的最后的一个元素（原队列还要存在）

{

Elemtype \*prElem = NULL;

Queue \*prTempQueue;

//这个临时队列，把原队列放到里面，当取完最后一个元素后，就把这个队列中的元素放回原来的队列

prTempQueue = InitQueue();

while(QueueEmpty(\*Q)==1)

{

DeQueue(Q,prElem);

EnQueue(prTempQueue,\*prElem);

}

while(QueueEmpty(\*prTempQueue)==1)

{

DeQueue(prTempQueue,prElem);

EnQueue(Q,\*prElem);

}

return \*prElem;

}

//Last

**BOOL QueueFull(Queue Q) //使用bool变量，boolean在此处无法运行**

//判断队是否为满，满返回TRUE，非满返回FALSE

{

if(((Q.rear+1)%MaxSize)==Q.front)

//队空

return TRUE;

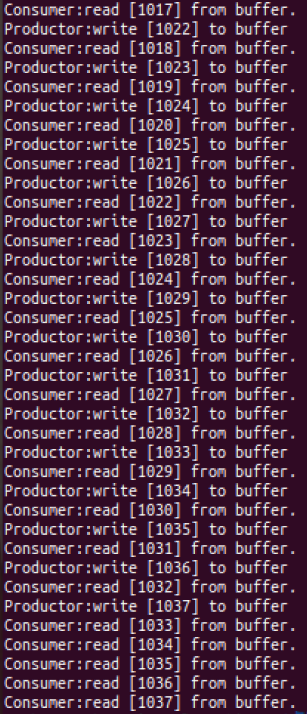
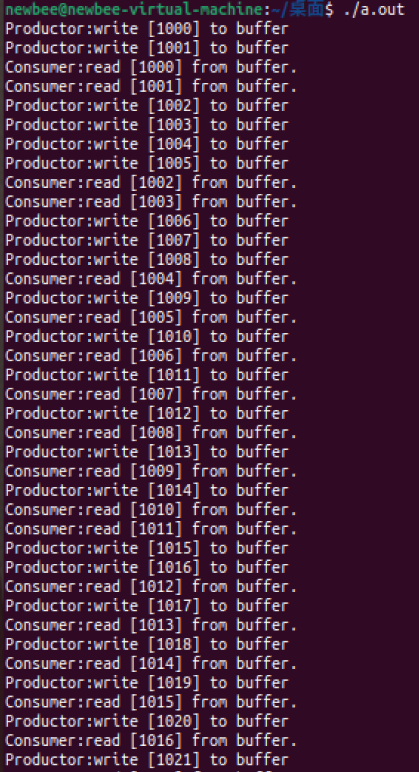
else

return FALSE;

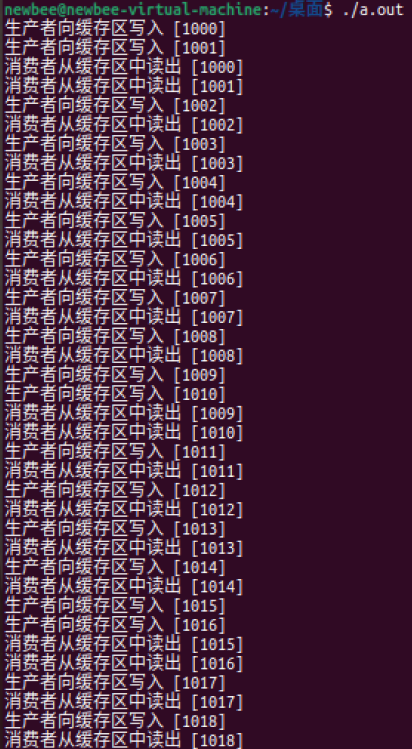
}

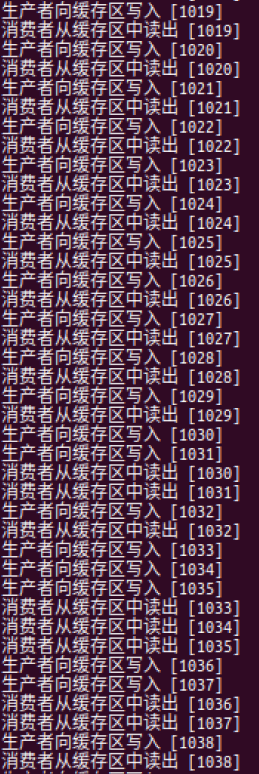
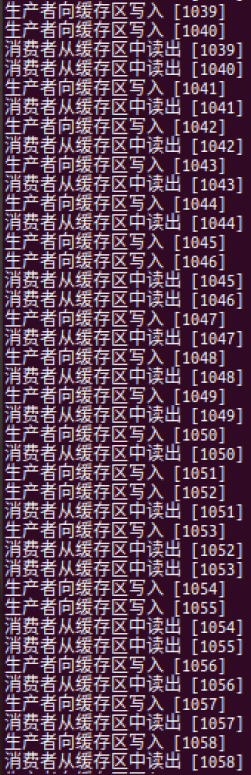
1. **运行截图**

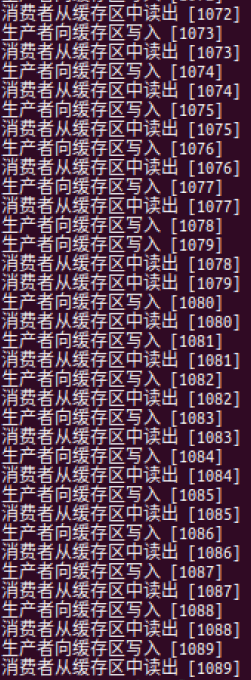
程序在未改进前的运行结果如下：



可以看到，程序中生产者每向缓冲区执行一次写操作，消费者不能立即执行读操作，而是在生产者再执行一次或几次之后才开始读，这样一来使得整个最终结果较为混乱。为此，对程序进行了优化改进，实现了让生产者每写一次，消费者便紧跟着读一次，最终结果如下：



此方法作出的改进并不十分完美，尽管生产者和消费者均为sleep(1)，但在实际运行过程中可能和1秒存在偏差，有时可能需要连续向缓冲区写入2-3个数据才能使缓冲区满，此时消费者才能从中读出对应的数据，也正因此消费者可能需要连续释放资源才会进行下一次资源的申请。所以，合理安排缓冲区大小，可能可以进一步完善该程序。