**生产者——消费者问题**

**实验报告**

姓名： 余泽晨

学号： 71115112

东南大学计算机科学与工程学院、软件学院

School of Computer Science & Engineering

College of Software Engineering

Southeast University

二0 17 年 5 月

一、实验目的

通过实验，掌握 Windows 和 Linux 环境下互斥锁和信号量的实现方法，加深对临界区问题和进程同步机制的理解，同时熟悉利用 Windows API 和 Pthread API 进行多线程编程的方法

二、实验内容

1. 在 Windows 操作系统上，利用 Win32 API 提供的信号量机制，编写应用程序实现生产者——消费者问题。

2. 在 Linux 操作系统上，利用 Pthread API 提供的信号量机制，编写应用程序实现生产者——消费者问题。

3. 两种环境下，生产者和消费者均作为独立线程，并通过 empty、full、mutex 三个信号量实现对缓冲进行插入与删除。

4. 通过打印缓冲区中的内容至屏幕，来验证应用程序的正确性。

三、实验步骤

1. 环境

Win10+VMWare Player+64位Ubuntu14.04虚拟机

2. 在 Win10 操作系统上，利用 Win32 API 提供的信号量机制，编写应用程序实现生产者——消费者问题.在程序中声明并实现output输出缓冲区函数,insert/remove函数,生产者/消费者进程函数,在主程序中让用户输入几个参数,初始化buffer,创建生产者/消费者线程并设置不同id,让主线程睡眠相应时间后退出.

3. 在 Linux 操作系统上，利用 Pthread API 提供的信号量机制，编写应用程序实现生产者——消费者问题. 在程序中声明并实现output输出缓冲区函数,insert/remove函数,生产者/消费者进程函数,在主程序中让用户输入几个参数,初始化buffer,创建生产者/消费者线程并设置不同id,让主线程睡眠相应时间后退出.

4.在生产者/消费者线程函数中打印缓冲区中的内容至屏幕，验证应用程序的正确性。

四、主要数据结构及其说明

1.int buffer[10],使用大小为10的int数组模拟缓冲区

2.int producerPtr, consumerPtr,插入或删除的元素在缓冲区的序号

3. Mutex, 互斥信号量Mutex,实现对缓冲区的互斥访问

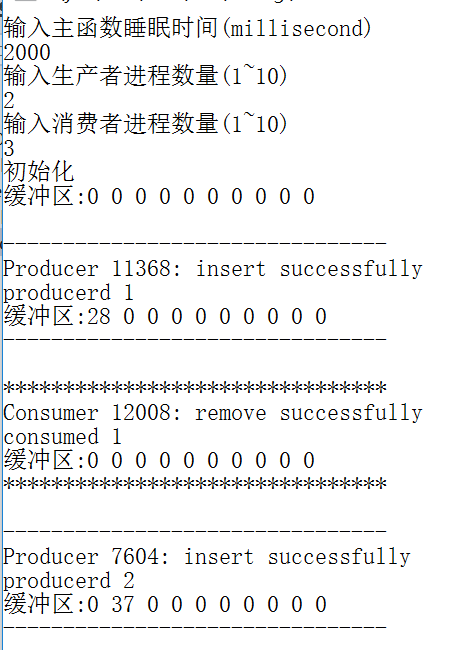
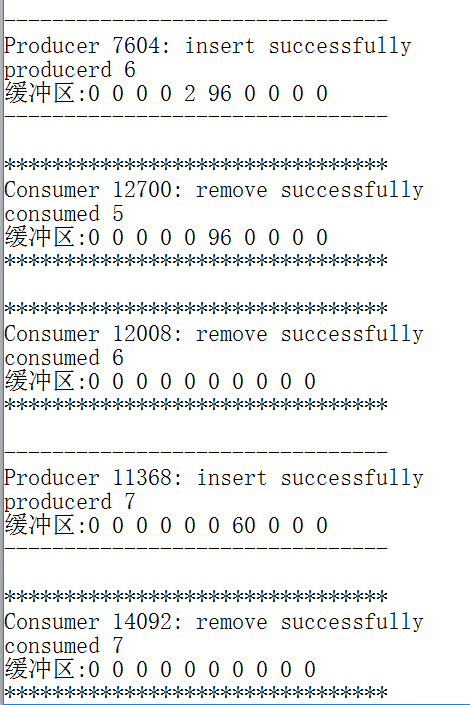
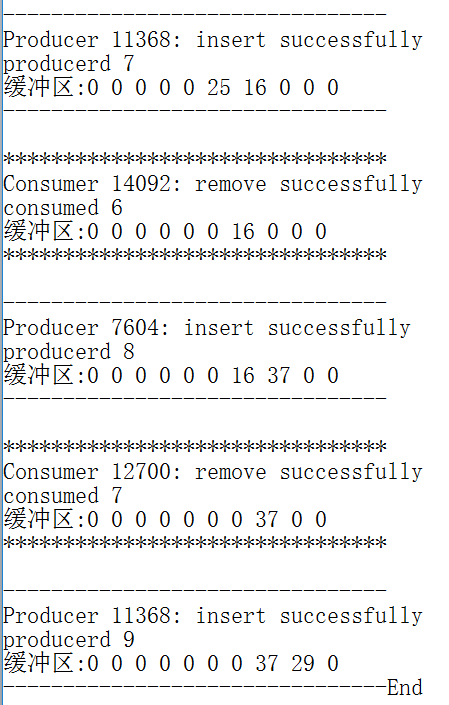
4.Empty, Full, 资源信号量Empty,表示缓冲区中空的数据槽数量,Full,表示缓冲区中满的数据槽的数量

5.PThreadId,CThreadId/pid,cid,生产者消费者线程的id

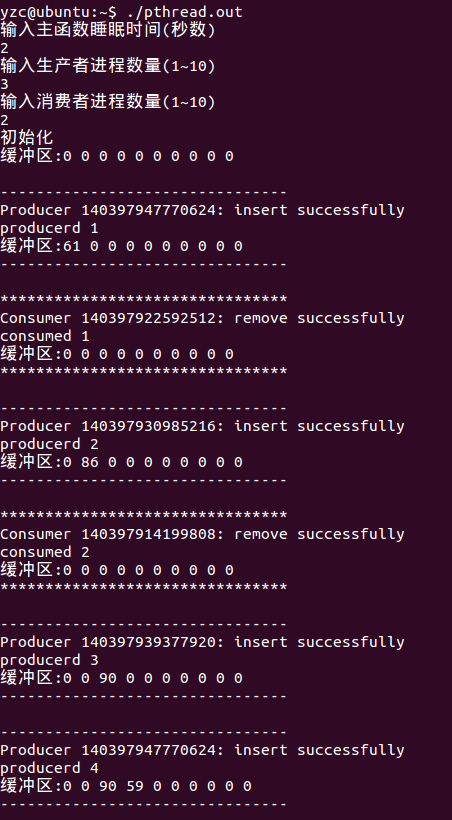
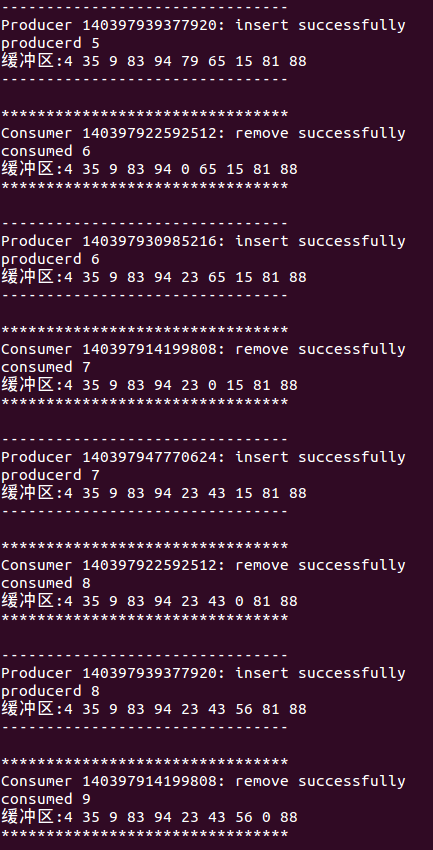
6.PThreadHandle,CThreadHandle/,生产者消费者线程

五、程序运行结果（系统截图）

1. Win32 API

2. Pthread API

六、实验体会

1.在实验中,熟悉了利用 Windows API 和 Pthread API 进行多线程编程,加深了对临界区问题和进程同步机制的理解的认识,对用信号量机制解决一些问题(如生产者消费者问题)也有了更深的体会.

2.在实验中,产生随机数是一个比较头疼的事,最后通过不同线程的不同id来产生不同的随机数种子,实现在短时间内产生不同的随机数.

七、源程序并附上注释

1. Win32 API

#include<iostream>

#include<Windows.h>

#include<stdlib.h>

#include<time.h>

using namespace std;

#define BUFFER\_SIZE 10;//缓冲区大小为10

HANDLE Mutex, Empty, Full;//互斥信号量Mutex,资源信号量Empty,Full

int producerPtr = 0, consumerPtr = 0;//插入或删除的元素在缓冲区的序号

int buffer[10];//表示缓冲区

void output() {//输出缓冲区

cout << "缓冲区:";

for (int i = 0; i < 10; i++)

cout << buffer[i] << ' ';

cout << endl;

}

bool insert(int x) {//插入元素

if (producerPtr >= consumerPtr&&producerPtr < consumerPtr + 10) {//若缓冲区未满,则可成功插入元素

buffer[producerPtr++ % 10] = x;

return true;

}

else

return false;//否则插入失败

}

bool remove() {//移除元素

if (consumerPtr <= producerPtr) {//若缓冲区非空则可成功移除一个元素

buffer[consumerPtr++ % 10] = 0;

return true;

}

else

return false;//否则移除失败

}

DWORD WINAPI producer(LPVOID param)//生产者线程

{

srand(3 \* time(0) + GetCurrentThreadId());

do {

int random = rand() % 100 + 1;

Sleep(random);//随机等待1~100ms

WaitForSingleObject(Empty, INFINITE);//若缓冲区已满,阻塞当前生产者线程

WaitForSingleObject(Mutex, INFINITE);//实现对缓冲区的互斥访问

cout << "--------------------------------" << endl;

cout << "Producer " << GetCurrentThreadId() << ": insert ";

if (insert(random))

cout << "successfully\nproducerd " << (producerPtr - 1) % 10 + 1 << endl;

else

cout << "failed\n" << endl;

output();

cout << "--------------------------------" << endl << endl;

ReleaseMutex(Mutex);

ReleaseSemaphore(Full, 1, NULL);

} while (true);

return 0;

}

DWORD WINAPI consumer(LPVOID param)//消费者线程

{

srand(2 \* time(0) - GetCurrentThreadId());

do {

int random = rand() % 100 + 1;

Sleep(random);//随机等待1~100ms

WaitForSingleObject(Full, INFINITE);//若缓冲区已空,阻塞当前消费者线程

WaitForSingleObject(Mutex, INFINITE);//实现对缓冲区的互斥访问

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "Consumer " << GetCurrentThreadId() << ": remove ";

if (remove())

cout << "successfully\nconsumed " << (consumerPtr - 1) % 10 + 1 << endl;

else

cout << "failed\n" << endl;

output();

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << endl;

ReleaseMutex(Mutex);

ReleaseSemaphore(Empty, 1, NULL);

} while (true);

return 0;

}

int main() {

int SleepTime, NumOfProducer, NumOfConsumer;

cout << "输入主函数睡眠时间(millisecond)" << endl;

cin >> SleepTime;

cout << "输入生产者进程数量(1~10)" << endl;

cin >> NumOfProducer;

cout << "输入消费者进程数量(1~10)" << endl;

cin >> NumOfConsumer;

cout << "初始化" << endl;

memset(buffer, 0, 10);//将缓冲区初始化为0

output();

cout << endl;

Empty = CreateSemaphore(NULL, 10, 10, NULL);

Full = CreateSemaphore(NULL, 0, 10, NULL);

Mutex = CreateMutex(NULL, FALSE, NULL);

DWORD \*PThreadId = new DWORD[NumOfProducer];

DWORD \*CThreadId = new DWORD[NumOfConsumer];

HANDLE \*PThreadHandle = new HANDLE[NumOfProducer];

HANDLE \*CThreadHandle = new HANDLE[NumOfConsumer];

for (unsigned short i = 0; i < NumOfProducer; i++)//创建生产者线程

{

PThreadId[i] = i;

PThreadHandle[i] = CreateThread(NULL, 0, producer, NULL, 0, &PThreadId[i]);

}

for (unsigned short i = 0; i < NumOfConsumer; i++)//创建消费者线程

{

CThreadId[i] = i;

CThreadHandle[i] = CreateThread(NULL, 0, consumer, NULL, 0, &CThreadId[i]);

}

Sleep(SleepTime);//主线程睡眠输入时间

cout << "End" << endl;

return 0;

}

2. Pthread API

#include <iostream>

#include <cstring>

#include <unistd.h>

#include <pthread.h>

#include <semaphore.h>

#include <stdlib.h>

using namespace std;

#define BUFFER\_SIZE 10//缓冲区大小为10

sem\_t Mutex, Empty, Full;//互斥信号量Mutex,资源信号量Empty,Full

int producerPtr = 0, consumerPtr = 0;//插入或删除的元素在缓冲区的序号

int buffer[10];//表示缓冲区

void output() {//输出缓冲区

cout << "缓冲区:";

for (int i = 0; i < 10; i++)

cout << buffer[i] << ' ';

cout << endl;

}

bool insert(int x) {//插入元素

if (producerPtr >= consumerPtr&&producerPtr < consumerPtr + 10) {//若缓冲区未满,则可成功插入元素

buffer[producerPtr++ % 10] = x;

return true;

}

else

return false;//否则插入失败

}

bool remove() {//移除元素

if (consumerPtr <= producerPtr) {//若缓冲区非空则可成功移除一个元素

buffer[consumerPtr++ % 10] = 0;

return true;

}

else

return false;//否则移除失败

}

void \*producer(void \*ptr)//生产者线程

{

srand(2\*time(0)-pthread\_self());

do {

int random = rand() % 100+1;

usleep(random\*1000);//随机等待1~100ms

sem\_wait(&Empty);//若缓冲区已满,阻塞当前生产者线程

sem\_wait(&Mutex);//实现对缓冲区的互斥访问

cout << "--------------------------------" << endl;

cout << "Producer " << pthread\_self() << ": insert ";

if (insert(random))

cout << "successfully\nproducerd " << (producerPtr - 1) % 10 + 1 << endl;

else

cout << "failed\n" << endl;

output();

cout << "--------------------------------" << endl << endl;

sem\_post(&Mutex);

sem\_post(&Full);

} while (true);

return 0;

}

void \*consumer(void \*ptr)//消费者线程

{

srand(3\*time(0)+pthread\_self());

do {

int random = rand() % 100 + 1;

usleep(random\*1000);//随机等待1~100ms

sem\_wait(&Full);//若缓冲区已空,阻塞当前消费者线程

sem\_wait(&Mutex);//实现对缓冲区的互斥访问

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl;

cout << "Consumer " << pthread\_self() << ": remove ";

if (remove())

cout << "successfully\nconsumed " << (consumerPtr - 1) % 10 + 1 << endl;

else

cout << "failed\n" << endl;

output();

cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << endl << endl;

sem\_post(&Mutex);

sem\_post(&Empty);

} while (true);

return 0;

}

int main() {

int SleepTime, NumOfProducer, NumOfConsumer;

cout << "输入主函数睡眠时间(秒数)" << endl;

cin >> SleepTime;

cout << "输入生产者进程数量(1~10)" << endl;

cin >> NumOfProducer;

cout << "输入消费者进程数量(1~10)" << endl;

cin >> NumOfConsumer;

cout << "初始化" << endl;

memset(buffer, false, 10);

output();

cout << endl;

sem\_init(&Empty,0,BUFFER\_SIZE);

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

sem\_init(&Mutex,1,1);

pthread\_t \*pid=new pthread\_t[NumOfProducer];

pthread\_t \*cid=new pthread\_t[NumOfConsumer];

pthread\_attr\_t attr;

pthread\_attr\_init(&attr);

for (unsigned short i = 0; i<NumOfProducer; i++)//创建生产者线程

{

pid[i] = i;

pthread\_create(&pid[i], &attr, producer, NULL);

}

for (unsigned short i = 0; i<NumOfConsumer; i++)//创建消费者线程

{

cid[i] = 10+i;

pthread\_create(&cid[i], &attr, consumer, NULL);

}

sleep(SleepTime);//主线程睡眠输入时间

cout<<"End"<<endl;

return 0;

}