华南师范大学

《操作系统实验》

实验报告

年级：

学号：

姓名：

指导老师：

实验设计时间：

一. 实验目的：

通过采用模拟的方法与编写操作系统内核相关的程序，设计、实现一种操作系统的某几个主要功能，巩固和加深对操作系统的主要功能、基本原理、主要算法和实施技术的理解， 将操作系统理论与实际相结合，提高编程能力。通过操作系统课程设计，学习和掌握操作系 统的主要功能、基本原理、主要算法和实施技术，懂得操作系统在现代计算机系统中的重要 作用，具有初步分析实际操作系统的基本能力。

二. 实验的内容：

通过编程解决生产者-消费者问题、读者-写者问题和多线程文件拷贝，学习多线程同步。

程序的并发执行往往带来与时间有关的错误，甚至引发灾难性的后果。这需要引入同步机制。使用多进程与多线程时，有时需要协同两种或多种动作，此过程就称同步（Synchronization)。引入同步机制的第一个原因是为了控制线程之间的资源同步访问，因为多个线程在共享资源时如果发生访问冲突通常会带来不正确的后果。例如，一个线程正在更新一个结构，同时另一个线程正试图读取同一个结构。结果，我们将无法得知所读取的数据是新的还是旧的，或者是二者的混合。第二个原因是有时要求确保线程之间的动作以指定的次序发生，如一个线程需要等待由另外一个线程所引起的事件。

通过编程实现 Socket 聊天程序，学习网络编程。

所谓 socket 通常也称作"套接字"，应用程序通常通过"套接字"向网络发出请求或者应答网络请求。 以 J2SDK-1.3 为例， Socket 和 ServerSocket 类库位于 java .net 包中。ServerSocket 用于服务器端， Socket 是建立网络连接时使用的。在连接成功时，应用程序两端都会产生一个 Socket 实例，操作这个实例，完成所需的会话。对于一个网络连接来说，套接字是平等的，并没有差别，不因为在服务器端或在客户端而产生不同级别。不管是 Socket 还是 ServerSocket 它们的工作都是通过 SocketImpl 类及其子类完成的。

通过编程现实大文件操作，学习内存映射。

内存映射文件，是由一个文件到一块内存的映射。Win32 提供了允许应用程序把文件映射到一个进程的函数 (CreateFileMapping)。内存映射文件与虚拟内存有些类似，通过内存映射

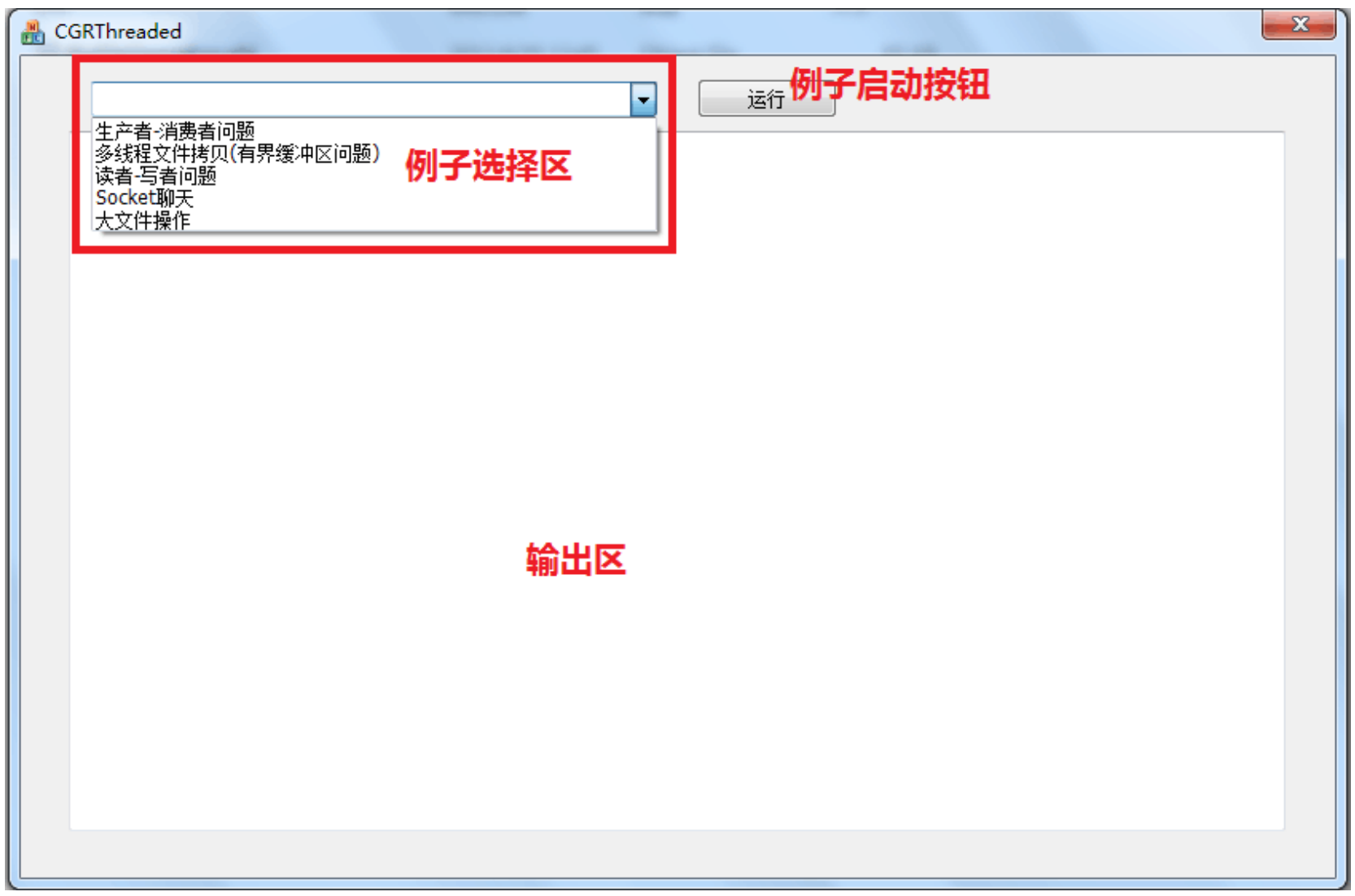
文件可以保留一个地址空间的区域，同时将物理存储器提交给此区域，内存文件映射的物理存储器来自一个已经存在于磁盘上的文件，而且在对该文件进行操作之前必须首先对文件进行映射。使用内存映射文件处理存储于磁盘上的文件时，将不必再对文件执行 I/O 操作，使得内存映射文件在处理大数据量的文件时能起到相当重要的作用。

三. 实验设计的主要思想：

通过编程解决一些经典的问题来学习操作系统编程，如多线程同步中，选择了生产者-消费者问题、读者-写者问题和多线程文件拷贝。网络通信中，选择了Socket聊天程序。文件系统中，选择了大文件操作。很多相关的其实问题都是这些经典问题的变种而已，可以说是一通百通的。本项目所有例子统一在Windows环境下，用MFC进行开发。

四. 实验设计的实现总体框架：

由于所选择的实验例子都是独立的，所以打算让它们实现和运行都互相独立，互不影响。这样方便调试，也容易看出每个例子的各自效果。



如图，这是程序的主界面。在“例子选择区”里，用户可以选择相应的例子。然后点击“运

行”，就能启动相应的例子。在输出区中，将会有相关的实验数据输出。

每个例子都建立至少一个头文件和源文件，如下面大文件操作例子中的部分代码，edit是输出区的控件指针，status字符串反映实验中各种变量或状态，events字符串反映实验中发生的各种事件。UpdateOutput函数将负责输出区的更新。每个例子都有一个Main函数。Main 函数名字和例子相关。

```
static  CEdit* edit;
static  CString status;
static  CString events;
static___int64 qwFileSize;

static  void  UpdateOutput()
{
    status.Format(L"FileSize:%d¥r¥n",qwFileSize);
    CString res;
    res.Append(L"版权所有：华南师范大学08级李海全¥r¥n");
    res.Append(L"**********************************************************************
**********¥r¥n");
    res.Append(L"Status:¥r¥n");
    res.Append(status);
    res.Append(L"**********************************************************************
**********¥r¥n");
    res.Append(L"Events:¥r¥n");
    res.Append(events);
    edit->SetWindowTextW(res);
    Sleep(1000);
}

VOID BigFileOperationMain(CEdit *Edit)
```

实现各个例子后，通过下面的代码将它们各自的Main 函数绑定到选择区的那个控件上。

```
mSamples.push_back(Sample(CString(L"生产者-消费者问题"),PRODUCER_COMSUMER,
(LPTHREAD_START_ROUTINE)ProducerComsumerMain));
    mSamples.push_back(Sample(CString(L"多线程文件拷贝(有界缓冲区问
题 )"),BOUNDED_BUFFER, (LPTHREAD_START_ROUTINE)BoundedBufferMain));
    mSamples.push_back(Sample(CString(L"读者-写者问题"),READERS_WRITERS,
(LPTHREAD_START_ROUTINE)ReadersWritersMain));
    mSamples.push_back(Sample(CString(L"Socket 聊 天 "),SOCKET_CHAT,
(LPTHREAD_START_ROUTINE)SocketChatMain));
mSamples.push_back(Sample(CString(L"大文件操作"),BIG_FILE_OPERATION,
(LPTHREAD_START_ROUTINE)BigFileOperationMain));
```

```
    CComboBox* box=((CComboBox*)GetDlgItem(IDC_SAMPLES));

    for(size_t i=0;i<mSamples.size();i++)
        box->InsertString(mSamples[i].type,mSamples[i].name);
```

当用户点击运行按钮时，通过下面的代码来启动相应的例子。所有例子都是通过多线程的方法来启动的。

```
void CCGRThreadedDlg::OnBnClickedRun()
{
    // TODO: 在此添加控件通知处理程序代码
    DWORD id;
     if(mCurSample)
        TerminateThread(mCurSample,-1);
     CComboBox* box=(CComboBox*)GetDlgItem(IDC_SAMPLES);
     CEdit* edit=(CEdit*)GetDlgItem(IDC_OUTPUT);

mCurSample=CreateThread(NULL,0,mSamples[box->GetCurSel()].func,edit,0,&id);
}
```

五.　源代码的详细分析：

生产者-消费者问题简述

生产者-消费者问题是最著名的进程同步问题。它描述了一组生产者向一组消费者提供产品，它们共享一个有界缓冲区，生产者向其中投入产品，消费者从中取走产品。这个问题是许多相互合作进程的一种抽象。例如，在输入时，输入进程是生产者，计算进程是消费者；在输出时，计算进程是生产者，打印进程是消费者。

为解决这一问题，应当设置两个同步信号量，一个说明空缓冲区数目，用empty表示，初值为有界缓冲区大小n；另一个说明满缓冲区数目（即产品数目），用full 表示，初值为0。此外，还需要设置一个互斥信号量mutex，其初值为1，以保证多个生产者或者多个消费者互斥访问缓冲池。

生产者-消费者问题的同步程序结构描述如下：

```
Semaphore full=0; /*满缓冲单元的数目*/
Semaphore empty=n; /*空缓冲单元的数目*/
```

```
Semaphore  mutex=1;  /*对有界缓冲区进行操作的互斥信号量*/
Buffer array[0,...,n-1]; /*存放产品的缓冲区*/
Integer  in=0,out=0;  /*缓冲池的指针, 指示生产者和消费者进程存取位置*/
Main ()
{
Cobegin
Producer();
Consumer();
Coend;
}
Producer()
{
While(true)
{
Produce an item put in nextp;
P(empty);
P(mutex);
Buffer(in)=nextp;  /*将产品放入缓冲池*/
in=(in+1) mod n; /* 指 针 后 移 */
V(mutex);
V(full);
}
}
Consumer()
{
While()
{
P(full);
P(mutex);
nextc=buffer(out);
out=(out+1) mod n;
V(mutex);
V(empty);
Consumer the item in nextc;
}
}
```

生产者-消费者问题源码

```
#include "stdafx.h"
#include <stdio.h>
```

```
#include <windows.h>
#include <stdlib.h>
#include "BoundedBuffer.h"
#include <process.h>



#define SIZE 10000
#define BUFSIZE 5

typedef struct _Buffer {
    void *Buf[SIZE];
    HANDLE hBufferLock;
    size_t nbyte;
} Buffer;

Buffer *SharedBuffer[BUFSIZE];
int head, tail;
int count;
int TheEnd;

static HANDLE hMutex;
static  HANDLE hNotFullEvent, hNotEmptyEvent;
static  size_t nbyte;




static  CEdit* edit;
static  CString status;
static  CString events;

static  void  UpdateOutput()
{
    status.Format(L"FullBuffersNum:%d¥r¥n",count);
    CString res;
    res.Append(L"版权所有：华南师范大学08级李海全¥r¥n");
    res.Append(L"**********************************************************
**********¥r¥n");
    res.Append(L"Status:¥r¥n");
    res.Append(status);
    res.Append(L"**********************************************************
```

```
**********************************¥r¥n");
    res.Append(L"Events:¥r¥n");
    res.Append(events);
    edit->SetWindowTextW(res);
    Sleep(1000);
}



void  Producer(FILE *infile)
{
    size_t n;
    int index;

    do {
        while(1){
            WaitForSingleObject(hMutex,INFINITE);
            if (count == BUFSIZE) {
                ReleaseMutex(hMutex);
                WaitForSingleObject(hNotFullEvent,INFINITE);
                continue;
            }
            break;
        }
        WaitForSingleObject(SharedBuffer[tail]->hBufferLock,INFINITE);

        index = tail;
        tail = (tail+1) % BUFSIZE;
        count++;
        ReleaseMutex(hMutex);
        n = fread(SharedBuffer[index]->Buf,1,SIZE,infile);
        SharedBuffer[index]->nbyte = n;
        CString str;
        str.Format(L"Produce %d bytes¥r¥n",n);
        events.Insert(0,str);   UpdateOutput();

        ReleaseMutex(SharedBuffer[index]->hBufferLock);
        PulseEvent(hNotEmptyEvent);
    } while(n > 0);
    TheEnd = TRUE;
```

```
    CString str;
    str.Format(L"exit producer thread¥r¥n");
    events.Insert(0,str);
    UpdateOutput();


}

void  Consumer(FILE *outfile)
{
    int index;
    while (1) {
        WaitForSingleObject(hMutex,INFINITE);
    if ((count == 0) && (TheEnd == TRUE))  {


        CString str;
        str.Format(L"End of data, exit consumer thread¥r¥n");
        events.Insert(0,str);
        UpdateOutput();

        ReleaseMutex(hMutex);
        ExitThread(0);
    }
    if (count == 0){
        ReleaseMutex(hMutex);

        WaitForSingleObject(hNotEmptyEvent,INFINITE);
        continue;
    }

    WaitForSingleObject(SharedBuffer[head]->hBufferLock,INFINITE);

    index = head;
    head = (head+1) % BUFSIZE;
    count--;
    ReleaseMutex(hMutex);

    fwrite(SharedBuffer[index]->Buf, SharedBuffer[index]->nbyte,1,
        outfile);


    CString str;
```

```
        str.Format(L"Consumed:wrote %d bytes¥r¥n",SharedBuffer[index]->nbyte);
        events.Insert(0,str);
        UpdateOutput();


        ReleaseMutex(SharedBuffer[index]->hBufferLock);
        PulseEvent(hNotFullEvent);
    }

}

void BoundedBufferMain(CEdit* Edit)
{
    events=L"";
    edit=Edit;


HANDLE hThreadVector[2];
DWORD ThreadID;
FILE *infile, *outfile;
Buffer Buf0,Buf1,Buf2,Buf3,Buf4;
int i;


    CString inPath;
CFileDialog inDlg(TRUE);
if(inDlg.DoModal()==IDOK)
    inPath=inDlg.GetPathName();
char cInPath[MAX_PATH];
wcstombs(cInPath,inPath.GetBuffer(),MAX_PATH);
for(int i=0;i<strlen(cInPath);i++)
    if(cInPath[i]=='¥¥')
        cInPath[i]='/';
infile = fopen(cInPath,"rb");


CString outPath;
CFileDialog  outDlg(FALSE);
if(outDlg.DoModal()==IDOK)
    outPath=outDlg.GetPathName();
char cOutPath[MAX_PATH];
wcstombs(cOutPath,outPath.GetBuffer(),MAX_PATH);
    for(int i=0;i<strlen(cOutPath);i++)
```

```
    if(cOutPath[i]=='¥¥')
        cOutPath[i]='/';
outfile = fopen(cOutPath,"wb");




count = 0;
head = 0;
tail = 0;

SharedBuffer[0] = &Buf0;
SharedBuffer[1] = &Buf1;
SharedBuffer[2] = &Buf2;
SharedBuffer[3] = &Buf3;
SharedBuffer[4] = &Buf4;


for  (i=0;i<BUFSIZE;i++)
    SharedBuffer[i]->hBufferLock = CreateMutex(NULL,FALSE,NULL);


hMutex = CreateMutex(NULL,FALSE,NULL);


hNotFullEvent = CreateEvent(NULL,TRUE,FALSE,NULL);
hNotEmptyEvent = CreateEvent(NULL,TRUE,FALSE,NULL);


hThreadVector[0]                =                CreateThread
(0,0,(LPTHREAD_START_ROUTINE)Producer,infile,0,&ThreadID);
    hThreadVector[1]     =          CreateThread     (0,0,(LPTHREAD_START_ROUTINE)
Consumer,outfile,0,&ThreadID);


    WaitForMultipleObjects(2,hThreadVector,TRUE,INFINITE);

    fclose(infile);
    fclose(outfile);
}
```

读者-写者问题简述

有一个许多进程共享的数据区，这个数据区可以是一个文件或者主存的一块空间；
有一些只读取这个数据区的进程（读者）和一些只往数据区写数据的进程(写者)，此外还需要满足以下条件：

（1） 任意多个读者可以同时读这个文件；

（2） 一次只有一个写者可以往文件中写；

（3） 如果一个写者正在进行操作，禁止任何读进程读文件。

我们需要分两种情况实现该问题：

读优先：一个读者试图进行读操作时，如果这时正有其他读者在进行读操作，他可直接开始读操作，而不需要等待。

写优先：一个读者试图进行读操作时，如果有其他写者在等待进行写操作或正在进行写操作，他要等待该写者完成写操作后才开始读操作。

读者优先算法：课本P49 中给出的算法即为读者优先算法。

写者优先算法：

1） 多个读者可以同时进行读

2） 写者必须互斥（只允许一个写者写，也不能读者写者同时进行）

3） 写者优先于读者（一旦有写者，则后续读者须等待，唤醒时优先考虑写者）

如果读者数是固定的，我们可采用下面的算法： rwmutex：用于写者与其他读者/写者互斥的访问共享数据 rmutex：该信号量初始值设为10，表示最多允许10 个读者同时进行读操作Semaphore rwmutex=1,rmutex=10；

```
procedure reader_i() //第i 个读者进程 (i=1,2,…)
{
P(rwmutex);
P(rmutex);
V(rwmutex); //释放读写互斥信号量，允许其它读、写进程访问
读数据;
V(rmutex);
}
procedure Writer_j //第j 个写者进程 (j=1,2,…)
{
P(rwmutex);
for (i = 1;i <= 10;i++)
P(rmutex); //禁止新读者，并等待已进入的读者退出
写更新;
for (i = 1;i <= 10;i++)
V(rmutex); // 恢复rmutex 值为10
V(rwmutex);
}
```

读者-写者问题源码

```
#include "stdafx.h"
#include <stdio.h>
#include <windows.h>
#include <stdlib.h>
#include "BoundedBuffer.h"
#include <process.h>

#define MAX 9999
static HANDLE  mutex;
static  HANDLE  blockedReaders;
static   HANDLE blockedWriters;
static int activeReaders = 0, waitingReader = 0;
static int activeWriters = 0, waitingWriter = 0;
static int sharedBuffer = 0;




static   CEdit* edit;
static   CString status;
static   CString events;

static   void   UpdateOutput()
{
    status.Format(L"SharedBuffer:%d¥r¥nActiveReaders:%d WaitingReader:%d
    ActiveWriters:%d
    WaitingWriter:%d¥r¥n",sharedBuffer,activeReaders,waitingReader,activeWriter
s,waitingWriter);
    CString res;
    res.Append(L"版权所有：华南师范大学08级李海全¥r¥n");
    res.Append(L"**********************************************************
**********************************¥r¥n");
    res.Append(L"Status:¥r¥n");
```

```
    res.Append(status);
    res.Append(L"**********************************************************************
**********************************¥r¥n");
    res.Append(L"Events:¥r¥n");
    res.Append(events);
    edit->SetWindowTextW(res);
    Sleep(1000);
}




static VOID StartReading()
{
    WaitForSingleObject(mutex, INFINITE);

    if (activeWriters > 0 || waitingWriter > 0) {
        waitingReader++;
        ReleaseMutex(mutex);

    }
    else {
        activeReaders++;
        ReleaseMutex(mutex);
    }
}

static VOID StopReading()
{
    WaitForSingleObject(mutex, INFINITE);
    activeReaders--;

    if (activeReaders == 0 && waitingWriter > 0) {
        activeWriters = 1;
        waitingWriter--;
        ReleaseSemaphore(blockedWriters, 1, NULL);
    }
    ReleaseMutex(mutex);
}

static VOID StartWriting()
{
    WaitForSingleObject(mutex, INFINITE);
```

```
    if (activeReaders == 0 && activeWriters == 0) {

        activeWriters = 1;
        ReleaseMutex(mutex);
    }
    else {

        waitingWriter++;
        ReleaseMutex(mutex);
        WaitForSingleObject(blockedWriters, INFINITE);
    }
}

static VOID StopWriting()
{
    WaitForSingleObject(mutex, INFINITE);
    activeWriters = 0;
    if (waitingReader > 0) {

        while (waitingReader > 0) {
            waitingReader--;
            activeReaders++;
            ReleaseSemaphore(blockedReaders, 1, NULL);
        }
    }
    else if (waitingWriter > 0) {

        waitingWriter--;

        ReleaseSemaphore(blockedWriters, 1, NULL);
    }
    ReleaseMutex(mutex);
}

static VOID Reader(LPVOID num)
{
    int localVar, i;
    for (i = 0; i<10;i++){
        StartReading();
        localVar = sharedBuffer;

        CString str;
        str.Format(L"Reader Thread %d reads %d¥r¥n",num,localVar);
        events.Insert(0,str);   UpdateOutput();
```

```
        StopReading();
    }
}

static VOID Writer(LPVOID num)
{
    int i, j;

    j = (int) num * 10;

    for (i=j; i< j+5; i++){
        StartWriting();
        sharedBuffer = i;

            CString str;
        str.Format(L"Writer Thread %d writes %d¥r¥n",num,sharedBuffer);
        events.Insert(0,str);   UpdateOutput();

        StopWriting();
    }
}

VOID ReadersWritersMain(CEdit *Edit)
{
      status=events="";
        edit=Edit;


    HANDLE hThreadVector[4];
    DWORD ThreadID;
    mutex = CreateMutex(NULL, FALSE, NULL);
    blockedReaders = CreateSemaphore(NULL, 0, MAX, NULL);
    blockedWriters = CreateSemaphore(NULL, 0, MAX, NULL);
    hThreadVector[0] = CreateThread (NULL, 0,
        (LPTHREAD_START_ROUTINE)Writer,(LPVOID) 1, 0,
        (LPDWORD)&ThreadID);

    hThreadVector[1] = CreateThread (NULL, 0,
        (LPTHREAD_START_ROUTINE)Writer,(LPVOID) 2, 0,
        (LPDWORD)&ThreadID);

    hThreadVector[2] = CreateThread (NULL, 0,
```

```
        (LPTHREAD_START_ROUTINE)Reader,(LPVOID) 3, 0,
        (LPDWORD)&ThreadID);

    hThreadVector[3] = CreateThread (NULL, 0,
        (LPTHREAD_START_ROUTINE)Reader,(LPVOID) 4, 0,

        (LPDWORD)&ThreadID);
    WaitForMultipleObjects(4,hThreadVector,TRUE,INFINITE);
}
```

Socket 聊天简述

一：SOCKET 简介

80 年代初，美国政府的高级研究工程机构（ARPA）给加利福尼亚大学Berkeley 分校提供了资金，让他们在UNIX 操作系统下实现TCP/IP 协议。在这个项目中，研究人员为TCP/IP 网络通信开发了一个API（应用程序接口）。这个API 称为Socket 接口（套接字）。今天，SOCKET 接口是TCP/IP 网络最为通用的API，也是在INTERNET 上进行应用开发最为通用的API。

90 年代初，由 Microsoft 联合了其他几家公司共同制定了一套WINDOWS 下的网络编程接口，即 WindowsSockets 规范。它是 BerkeleySockets 的重要扩充，主要是增加了一些异步函数，并增加了符合Windows 消息驱动特性的网络事件异步选择机制。WINDOWSSOCKETS 规范是一套开放的、支持多种协议的Windows 下的网络编程接口。从 1991 年的 1.0 版到 1995 年的 2.0.8 版，经过不断完善并在Intel、Microsoft、Sun、SGI、Informix、Novell 等公司的全力支持下，已成为Windows 网络编程的事实上的标准。目前，在实际应用中的 WINDOWSSOKCETS 规范主要有 1.1 版和 2.0 版。两者的最重要区别是 1.1 版只支持TCP/IP 协议，而 2.0 版可以支持多协议。2.0 版有良好的向后兼容性，任何使用1.1 版的源代码，二进制文件，应用程序都可以不加修改地在2.0 规范下使用。

SOCKET 实际在计算机中提供了一个通信端口，可以通过这个端口与任何一个具有SOCKET 接口的计算机通信。应用程序在网络上传输，接收的信息都通过这个SOCKET 接口来实现。在应用开发中就像使用文件句柄一样，可以对SOCKET 句柄进行读，写操作。

二：基于WINDOWS SOCKET 的应用开发介绍。

在 WINDOWS95/98,WINDOWSNT 进行WINSOCK 开发使用的编程语言有很多,VC++, JAVA,DELPHI, VB 等。其中 VC 时使用最普遍，和 WINSOCK 结合最紧密的。并且 VC++对原来的WindowsSockets 库函数进行了一系列封装，继而产生了CAsynSocket、CSocket、CSocketFile 等类，它们封装着有关Socket 的各种功能，是编程变得更加简单。但如果你是一个WINSOCK 编程的初学