**实验3 流式套接字应用程序运行分析**

【实验要求】

1、服务器运行异常现象分析

模拟服务器端各种运行异常情況，用 Wireshark抓包，分析对应的现象并提出解决建议。

场景一：服务器进程崩溃

场景二：服务器宿主机崩溃或网络紊乱

场景三：服务器宿主机崩溃后重启

2、流式套接字传输效率分析

测试一：对于固定长度的数据，分析发送次数对发送效率的影响

测试二、测试接收绶冲区大小与数据传送时间的关系

【实验过程】

**1.正常客户端和服务器通信**

通过“netstat -p tcp”命令观察客户端和服务器主机中已经建立的TCP连接状态，如图1所示。



图1 netstat命令观察已经建立的tcp连接

启动服务器，服务器程序可以是之前流式套接字实现的服务器。使用“netstat -a ”观察服务器的连接状态，系统中的27015端口已经处于监听状态。



图 2 通过netstat命令观察到服务器启动后tcp的状态

启动wireshark，设置过滤条件“TCP”，开始捕获网络中的TCP数据。此处省略Wireshark抓包截图。

启动客户端，并在连接建立成功后，使用“netstat -a”命令观察当前主机中的TCP连接状态，系统中的27015端口已经增加了一个ESTABLISHED状态，同时还保留LISTENING状态的记录。如图3所示。

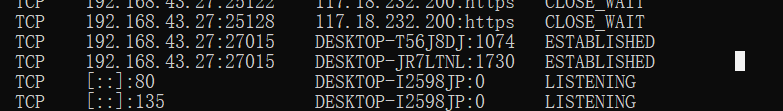


图3 客户端与服务器建立连接后的TCP在7210端口上的连接状态

客户端向服务器端发送信息，运行效果如图4所示。

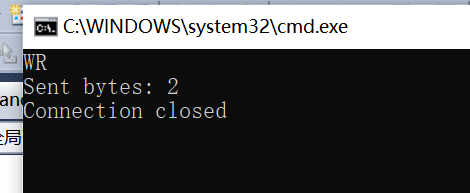


图4a 客户端

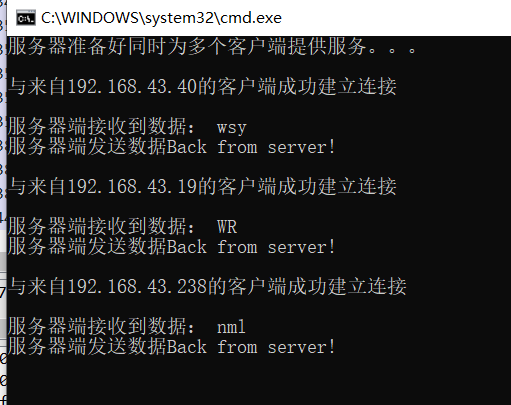


图4b 服务器端

停止Wireshark抓包，观察回射过程的通信过程。如图5所示。

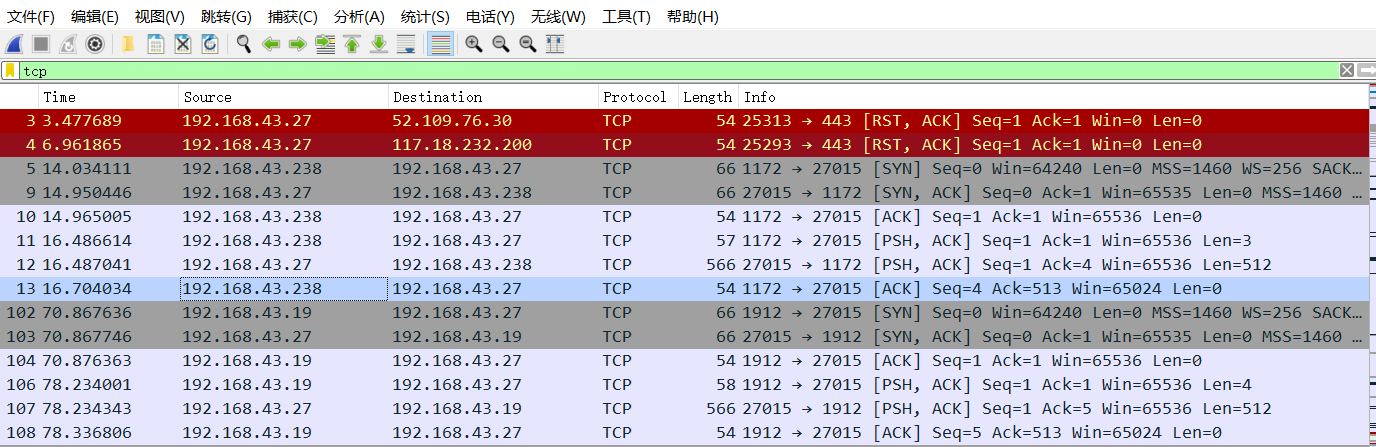
****

图 5 应用程序进程间通信数据包捕获情况

**2.服务器进程崩溃**

2.1正常通信，同实验过程1。

2.2开始wireshark抓包。

服务器进程崩溃，客户端出现错误，如图6所示。

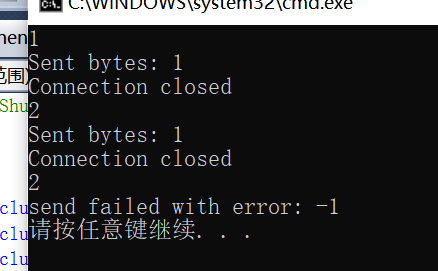


图6 客户端

2.3关闭服务器进程。

在服务器电脑上通过Alt+Ctrl+Del复合键启动任务管理器，在任务管理器中结束服务器进程。如图7所示。



图7 关闭服务器进程

2.4客户端停止抓包，观察到继续回射时，服务器返回RST包。

由于服务器端异常终止，服务器端的TCP协议在收到来自客户端的数据时，返回RST数据包，以通知客户端，服务器出现了异常关闭的情况。客户端可以通过获取出错代码来退出连接。如图8所示。

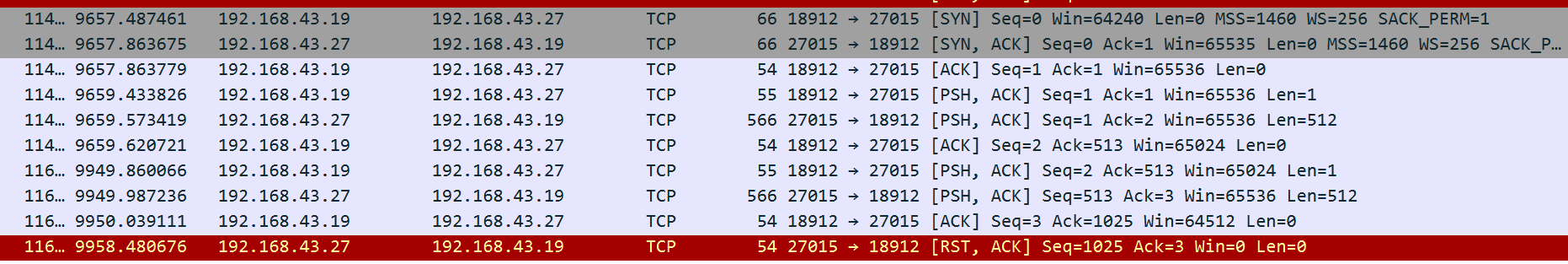
****

图8 服务器崩溃后客户端收到RST包

**3.主机崩溃或网络紊乱**

3.1正常通信，同实验过程1。

3.2断掉服务器网线，或禁用网卡。

通过禁用网卡的方法模拟主机崩溃或网络紊乱的情况。如图9所示。

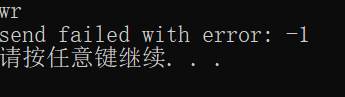


图9a 客户端

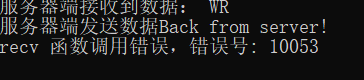


图9b 服务器端

3.3停止抓包，客户端尝试多次重传无果后退出。

如图10所示，在服务器端的协议无法工作时，只能捕获到客户端发往服务器的协议数据包，没有服务器端的返回数据包。客户端在多次尝试发送后，报错并退出连接。

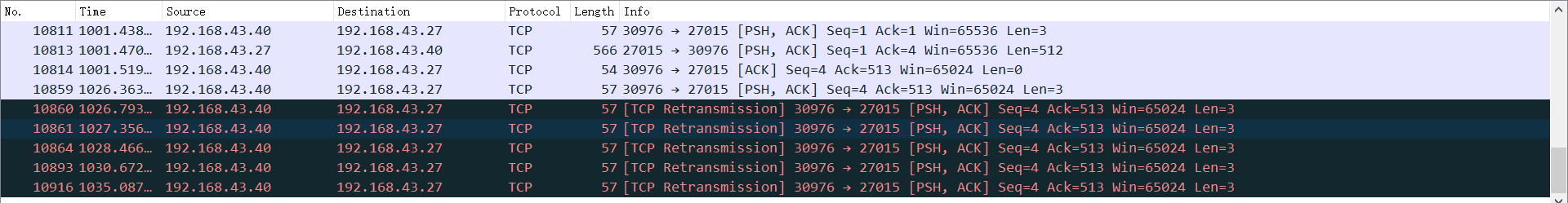


图10 客户端多次重传尝试

**4．主机崩溃后重启**

4.1正常通信，同实验过程1。

4.2服务器断开后重启。

服务器断开后， 如图11所示。

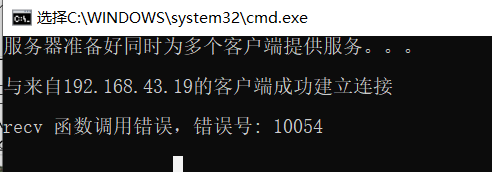


图11 服务器断开

4.3停止抓包，服务器重启，给客户端返回RST包

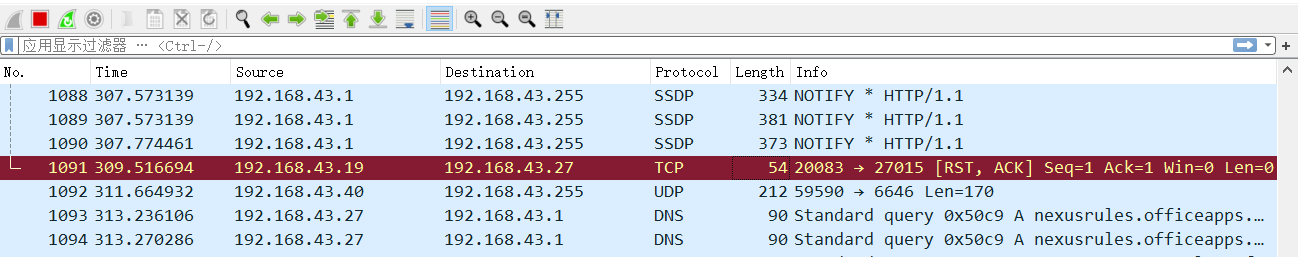


图12a

如图12a所示，服务器宿主机崩溃后，客户端多次尝试重发数据，没有响应，后超时退出。故也可能收不到RST数据包。

服务器主机重启后，客户端需要重新建立与服务器的连接。如图12b所示。

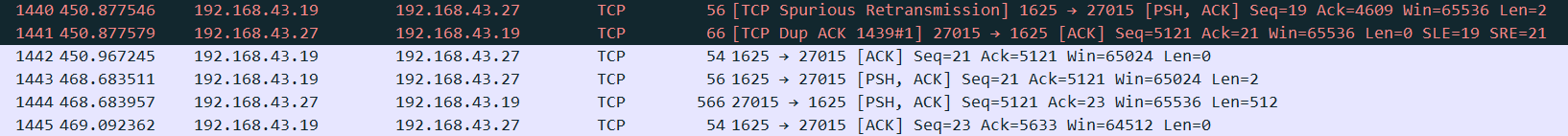


图12b 服务器重启后客户端重新与其建立连接

**5.对于固定长度n,分析发送次数对发送效率的影响**

实验设置总发送长度为10000字节，选择单次发送长度分别为10000字节、1000字节、100字节、10字节和1字节。发送次数越多，系统调用协议的次数越多，产生的系统消耗也就越大。如图13所示。

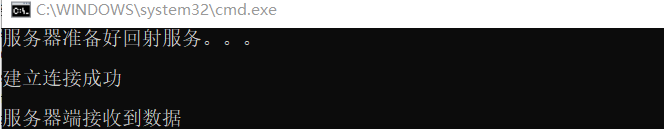
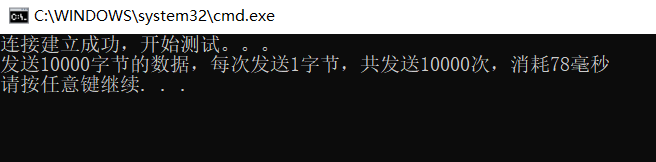
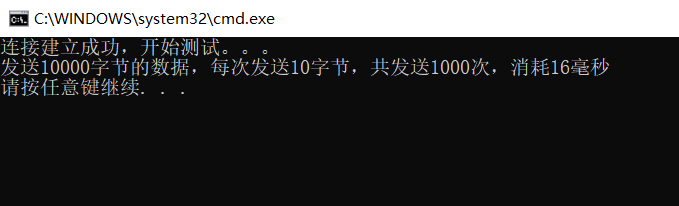
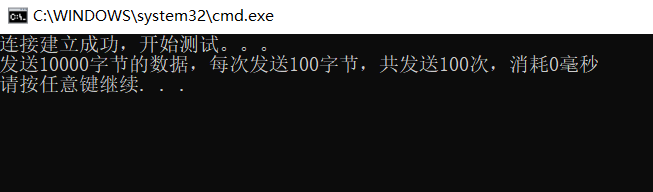
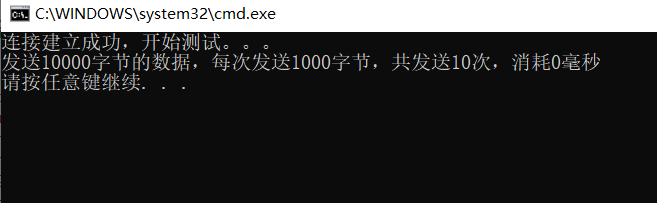


图13a 服务器









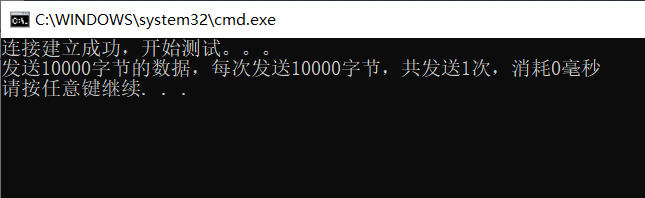


图13b 客户端

**6.测试接收缓冲区与数据传送时间的关系。**

6.1服务器端模拟一个慢速设备的处理。

在应用程序接收缓冲仅设置100字节的空间，对客户端发来的数据按照100字节为单位进行接收，且每次接收间隔100毫秒。

6.2传参数测试

两次测试时，客户端均按照每次发送10000字节，共发送10次。第一次设置服务器接收缓冲区长度为1000000字节，客户端发送时间为62毫秒。第二次设置服务器缓冲区为100字节，客户端发送时间为52625毫秒。如图14所示。

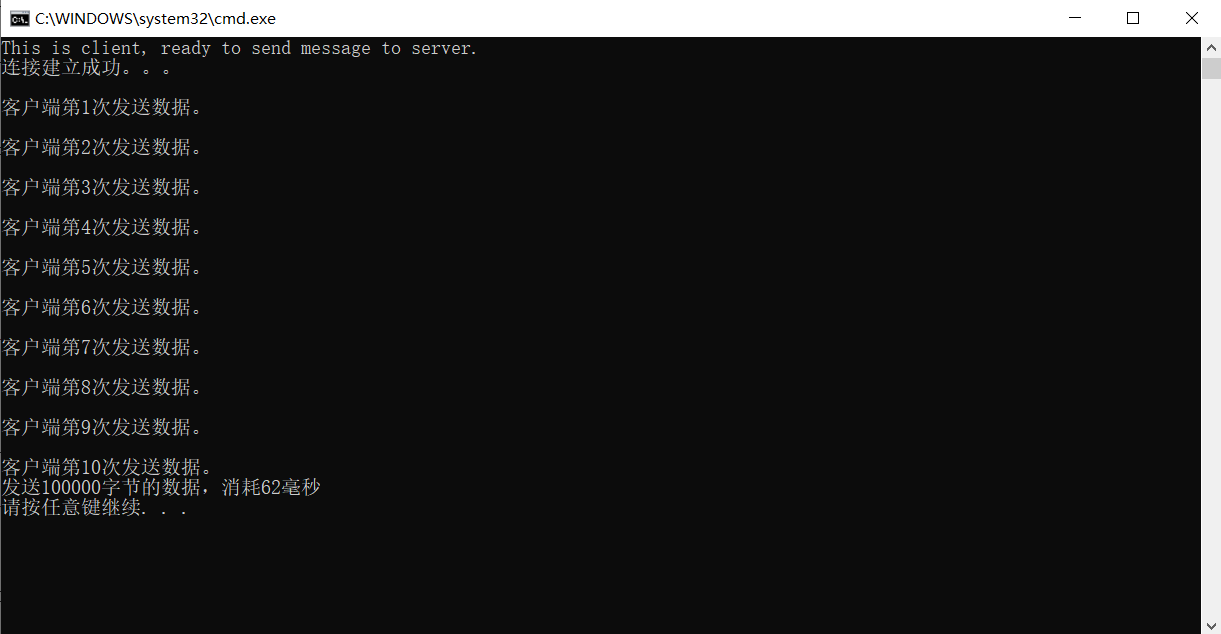
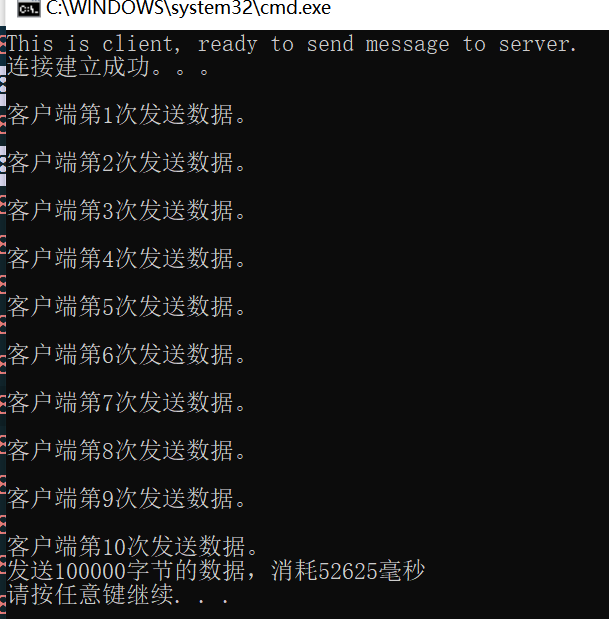
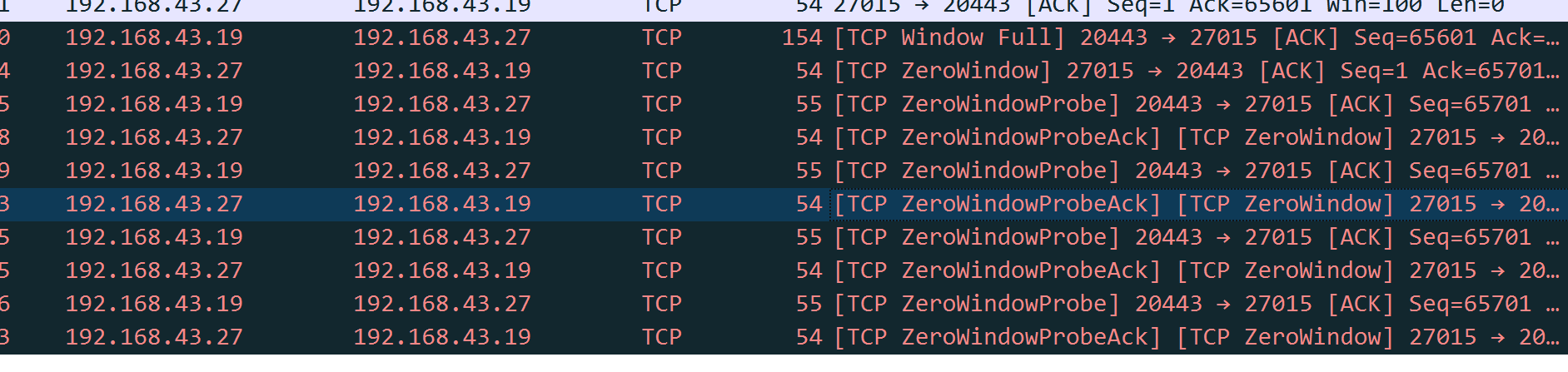
 

图14a服务器缓冲区为1000000 图14b 服务器缓冲区为100

对于客户端发送的总字节数<服务器的系统接收缓存，消耗时间很少；对于客户端发送数据的总字节数>服务器系统接收缓存，消耗的时间很长。

6.3 Wireshark抓包

由于服务器处理速度慢，造成太小的接收缓存不足以存储客户端发来的大量数据，从而限制了发送速度。通过Wireshark抓包观察到，第二次测试网络中出现了大量的接收窗口询问数据包[TCP zerowindowProbe]和[TCP zerowindowProbeACK]。如图15所示。

图15

TCP报文中的window size表示发出这个报文的一端准备多少bytes的数据，当TCP的一端一直接收数据，但是应用层没有及时读取的话，数据一直在TCP模块中缓存，最终受限于接收缓存的大小，window size会变为0，此时我们称呼这个接收窗口为零窗(zero window)，对端也不能再发送更多的数据。

**代码**

**测试一**

**服务器端**

// EchoTCPServer.cpp : 定义控制台应用程序的入口点。

#include "stdafx.h"

#include "CSocketFrame.h"

#include "winsock2.h"

#define ECHOPORT "27015"

int tcp\_server\_fun\_echo( SOCKET s );

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

{

CSocketFrame frame;

int iResult = 0;

SOCKET ListenSocket, ConnectSocket;

//输入参数合法性检查

if (argc != 1)

{

printf("usage: EchoTCPServer-sendn");

return -1;

}

//Windows Sockets Dll初始化

frame.start\_up();

//创建服务器端的流式套接字并在指定端口号上监听

ListenSocket = frame.tcp\_server( NULL, ECHOPORT );

if ( ListenSocket == -1 )

return -1;

printf("服务器准备好回射服务。。。\n");

for ( ; ; ) {

if((ConnectSocket = accept( ListenSocket, NULL, NULL )) != INVALID\_SOCKET ){

//建立连接成功

printf("\r\n建立连接成功\n\n");

//回射

iResult = tcp\_server\_fun\_echo( ConnectSocket );

//如果出错，关闭当前连接套接字，继续接收其它客户端的请求

if(iResult == -1)

printf("当前连接已关闭或出错!\n");

}

else{

printf("accept 函数调用错误，错误号： %d\n", WSAGetLastError());

frame.quit( ListenSocket );

return -1;

}

//关闭连接套接字

if ( closesocket( ConnectSocket ) == SOCKET\_ERROR)

printf("closesocket 函数调用错误，错误号：%d\n", WSAGetLastError());;

}

frame.quit( ListenSocket );

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名：tcp\_server\_fun

输入参数：SOCKET s:服务器的连接套接字

输出参数：0：成功，-1：失败

功能：tcp\_server的具体功能函数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int tcp\_server\_fun\_echo( SOCKET s )

{

int iResult = 0;

char recvline[MAXLINE];

do {

memset( recvline, 0, MAXLINE );

//接收数据

iResult = recv( s, recvline, MAXLINE, 0);

if (iResult > 0){

printf("服务器端接收到数据%s\n", recvline);

//回射发送已收到的数据

iResult = send( s,recvline,iResult, 0 );

if(iResult == SOCKET\_ERROR)

{

printf("send 函数调用错误，错误号: %ld\n", WSAGetLastError());

iResult = -1;

}

else

printf("服务器端发送数据%s\n", recvline);

}

else{

if (iResult == 0)

printf("对方连接关闭，退出\n");

else{

printf("recv 函数调用错误，错误号: %d\n", WSAGetLastError());

iResult = -1;

}

break;

}

} while (iResult > 0);

return iResult;

}

**客户端**

// EchoTCPClient.cpp.cpp : 定义控制台应用程序的入口点。

#include "stdafx.h"

#include <time.h>

#include "CSocketFrame.h"

int tcp\_client\_fun\_sendn(SOCKET s ,int totallen ,int singlelen);

#define ECHOPORT "27015"

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

{

CSocketFrame frame;

int iResult;

SOCKET ClientSocket;

int totallen,singlelen;

//输入参数合法性检查

if (argc != 4)

{

printf("usage: EchoTCPClient-sendn <IPaddress> <TotalLen> <SingleLen>");

return -1;

}

//Windows Sockets Dll初始化

frame.start\_up();

//创建客户端的流式套接字，并与服务器建立连接

printf("连接建立成功，开始测试。。。\n");

ClientSocket = frame.tcp\_client( ( char \*)argv[1], ECHOPORT );

if ( ClientSocket == -1 )

return -1;

totallen = atoi(argv[2]);

singlelen =atoi(argv[3]);

//开始测试

iResult = tcp\_client\_fun\_sendn(ClientSocket ,totallen,singlelen);

frame.quit( ClientSocket );

return iResult;

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名：tcp\_client\_fun\_sendn

输入参数：

SOCKET s:服务器的连接套接字

int totallen:总共发送的字节数

int singlelen:一次send调用发送的字节数

输出参数：0：成功，-1：失败

功能：tcp\_client测试客户端的具体功能函数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int tcp\_client\_fun\_sendn(SOCKET s ,int totallen ,int singlelen)

{

int iResult, leftlen, times=0;

char sendline[MAXLINE],recvline[MAXLINE];

memset(sendline,0,MAXLINE);

memset(recvline,0,MAXLINE);

char \*sendbuf;

DWORD beginticks,endticks;

if(totallen<1 || singlelen<1 || singlelen > totallen)

return -1;

//根据totallen为发送缓存申请空间并为发送缓存赋值

sendbuf = (char \*)malloc(totallen);

memset(sendbuf,0,totallen);

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

sendbuf[i]=i;

char \*ptr = sendbuf;

leftlen=totallen;

//获取当前时间

beginticks = GetTickCount();

//循环发送数据

while(leftlen > 0)

{

iResult = send(s,ptr,singlelen,0);

if(iResult == SOCKET\_ERROR)

{

printf("send 函数调用错误，错误号： %ld\n", WSAGetLastError());

delete(sendbuf);

return -1;

}

ptr +=iResult;

leftlen -=iResult;

times++;

}

endticks =GetTickCount();

printf("发送%d字节的数据，每次发送%d字节，共发送%d次，消耗%d毫秒\r\n",totallen,singlelen,times,endticks-beginticks );

free(sendbuf);

return iResult;

}

**测试二**

**服务器端**

// EchoTCPServer2.cpp : 定义控制台应用程序的入口点。

#include "stdafx.h"

#include "CSocketFrame.h"

#include "winsock2.h"

#define ECHOPORT "27015"

#define BUFLEN 100

int tcp\_server\_fun\_testrecvbuf( SOCKET s );

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

{

CSocketFrame frame;

int iResult = 0;

int buflen;

SOCKET ListenSocket, ConnectSocket;

//输入参数合法性检查

if (argc != 2)

{

printf("usage: EchoTCPServer-testrecvbuf <RecvBufLen>");

return -1;

}

buflen =atoi( argv[1]);

//Windows Sockets Dll初始化

frame.start\_up();

//创建服务器端的流式套接字并在指定端口号上监听

ListenSocket = frame.tcp\_server( NULL, ECHOPORT );

if ( ListenSocket == -1 )

return -1;

//设置服务器接收缓存

iResult = setsockopt( ListenSocket, SOL\_SOCKET, SO\_RCVBUF, ( char \* )&buflen, sizeof( buflen ));

if ( iResult == SOCKET\_ERROR){

printf("setsockopt function failed with error %d\n", WSAGetLastError());

frame.clean\_up();

return -1;

}

printf("服务器准备好测试。。。\n");

for ( ; ; ) {

ConnectSocket = accept( ListenSocket, NULL, NULL );

if(ConnectSocket != INVALID\_SOCKET ){

//建立连接成功

printf("\r\n建立连接成功\n\n");

//回射

iResult = tcp\_server\_fun\_testrecvbuf( ConnectSocket );

//如果出错，关闭当前连接套接字，继续接收其它客户端的请求

if(iResult == -1)

printf("当前连接已关闭或出错!\n");

}

else{

printf("accept 函数调用错误，错误号： %d\n", WSAGetLastError());

frame.quit( ListenSocket );

return -1;

}

//关闭连接套接字

if ( closesocket( ConnectSocket ) == SOCKET\_ERROR)

printf("closesocket 函数调用错误，错误号：%d\n", WSAGetLastError());;

}

frame.quit( ListenSocket );

return 0;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名：tcp\_server\_fun\_testrecvbuf

输入参数：SOCKET s:服务器的连接套接字

输出参数：0：成功，-1：失败

功能：tcp\_server服务器的具体功能函数

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int tcp\_server\_fun\_testrecvbuf( SOCKET s )

{

int iResult = 0;

char recvline[BUFLEN];

do {

memset( recvline, 0, BUFLEN );

//接收数据

iResult = recv( s, recvline, BUFLEN, 0);

if (iResult > 0)

printf("服务器端接收到数据%d字节\n", iResult);

else{

if (iResult == 0)

printf("对方连接关闭，退出\n");

else{

printf("recv 函数调用错误，错误号: %d\n", WSAGetLastError());

iResult = -1;

}

break;

}

Sleep(100);

} while (iResult > 0);

return iResult;

}

**客户端**

// EchoTCPClient2.cpp : 定义控制台应用程序的入口点。

#include "stdafx.h"

#include<string>

using namespace std;

#include "CSocketFrame.h"

int tcp\_client\_fun\_echo(FILE \*fp,SOCKET s);

#define ECHOPORT "27015"

#define BUFLEN 10000

int tcp\_client\_fun\_testrecvbuf(SOCKET s, int times);

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

{

CSocketFrame frame;

int iResult;

int times;

SOCKET ClientSocket;

//输入参数合法性检查

if (argc != 3)

{

printf("usage: EchoTCPClient-testrecvbuf <IPaddress> <Times>");

return -1;

}

times = atoi(argv[2]);

//Windows Sockets Dll初始化

frame.start\_up();

//创建客户端的流式套接字，并与服务器建立连接

printf("连接建立成功。。。\n");

ClientSocket = frame.tcp\_client( ( char \*)argv[1], ECHOPORT );

if ( ClientSocket == -1 )

return -1;

//开始回射请求的发送与接收

iResult = tcp\_client\_fun\_testrecvbuf(ClientSocket,times);

frame.quit( ClientSocket );

return iResult;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

函数名：tcp\_client\_fun\_testrecvbuf

输入参数：

SOCKET s:服务器的连接套接字

int times:发送次数，每次10000字节

输出参数：0：成功，-1：失败

功能：tcp\_client：根据用户指定的发送次数循环发送数据给服务器

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int tcp\_client\_fun\_testrecvbuf(SOCKET s, int times)

{

int iResult;

char sendline[BUFLEN],recvline[BUFLEN];

DWORD beginticks,endticks;

memset(sendline,0,BUFLEN);

memset(recvline,0,BUFLEN);

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

sendline[i]=i;

//获取当前时间

beginticks = GetTickCount();

//循环发送

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

{

iResult = send(s,sendline,BUFLEN,0);

if(iResult == SOCKET\_ERROR)

{

printf("send 函数调用错误，错误号： %ld\n", WSAGetLastError());

return -1;

}

printf("\r\n客户端第%d次发送数据。\r\n", i+1);

}

endticks =GetTickCount();

printf("发送%d字节的数据，消耗%d毫秒 \r\n",times\*BUFLEN,endticks-beginticks );

return iResult;

}

【实验总结】

添加CSocketFrame框架时，出现了很多错误。一些错误通过修改、添加头文件可以解决。需要注意的是，在CSocketFrame.h文件中，需要注释掉构造函数和析构函数。此外，还可以尝试不使用类框架完成。