《信息安全技术课程设计》实验指导书

网络空间安全学院
信息安全与信息对抗实验教学中心

《信息安全技术课程设计》实验指导书

目录

一、课程设计目的与任务

1．目的：

（1）加深对计算机网络的基本概念和原理，以及网络编程接口及 Winsock 概念、编程原理的理解；

（2）提高学生网络应用与编程的能力。学生在七个设计题目中选择一个，并可选择任意开发工具来设计完成，提高其分析与解决问题的能力，为大型网络编程打下坚实基础；

（3）通过撰写课程设计报告，锻炼学生的逻辑组织和语言表达能力；

（4）培养学生理论运用于实践的综合应用和设计创新能力。

2．任务：

（1）设计完成与网络相关题目的基本功能要求；

（2）调试运行之后，要求边演示边解释设计的思想、过程及采用的方法。

（3）按一定的规范要求，撰写课程设计报告。

3. 实验时间

时间：3月31日-4月10日

二、课程设计的基本要求

1．熟练掌握网络的基本概念和原理；

2．掌握网络编程接口及 Winsock 概念及编程原理；

3．掌握基于 TCP/IP 的 Internet 编程技术；

4．掌握各种软件开发工具的使用过程及方法。

三、考核方式：

要求学生每班按学号顺序依次每 2 人组成一组，以组号为顺序，依次选择下面七个题目之一完成，若多完成题目可酌情加分。最后通过调试运行，并以组为单位进行检查。组员必须有明确的分工，每人所撰写的课程设计报告内容应为本人实际完成的实习内容。

1、平时成绩占 30%，主要是平时纪律情况，分数为 30 分。

2、 综合考核占 70%，主要是实验结果的检查和实验报告（网安院课程设计模板）情况，分数为 70 分。总成绩由百分制转换成五级分制。即优秀、良、中、及格、不及格。

3、实验报告提交事项：

- 时间：4 月 25 日（周五）。
- 内容：提交电子版实验报告和实验代码，班长收集
- 电子版命名格式：班级-学号-姓名

四、7 个题目表：

题目一：暴力破解密码

要求：1、设计一个信息系统，该系统可为学籍管理系统、订餐系统、票务管理系统不限，系统必须通过客户端录入账号口令远程登录；

2、系统内至少包含三个以上账号，密码为 6 位以上数字组成；

3、设计程序在该系统所在服务器端进行暴力密码破解，破解后将账号密码通过套接字发送出去；

4、在客户端用破解得到的账号口令进行登录，验证破解成功。

题目二：字典破解密码

要求：1、设计一个信息系统，该系统可为学籍管理系统、订餐系统、票务管理系统不限，系统必须通过客户端录入账号口令远程登录；

2、系统内至少包含三个以上账号，密码为 6 位以上任意字符组成；

3、设计程序设计程序在该系统所在服务器端进行字典密码破解，破解后将账号密码通过套接字发送出去；

字典举例：

当前账号信息：

user：1234567

字典：

00000000

111111111

1234567

6666666666

Aaaaaaaa

1234567

为了破解用户名 user 的密码，将字典文件和密码进行依次比对，比对成功即破解。

4、在客户端用破解得到的账号口令进行登录，验证破解成功。

题目三：认证审计系统

1、 设计一个信息系统，系统必须通过客户端录入账号口令远程登录；

2、 系统内至少包含三个以上账号；

3、 某账号登录后服务器端可实时显示该账号登录的时间及 IP 信息；

4、 服务器端可查询账号的历史登录信息。

题目四：数据嗅探系统

1、 设计一个信息系统，系统必须通过客户端录入账号口令远程登录；

2、 登录后客户端可通过键盘输入向服务器发送数据；

3、 服务器端设置嗅探关键字，如果客户端发送的数据包含该关键字，即将该数据显示出来。

举例：

服务器设置关键字：密码

客户端从键盘输入的数据 1：你好，我是张三

客户端从键盘输入的数据 2：我的密码是 password

服务器端显示：我的密码是 password

题目五：防火墙系统

1、 设计一个信息系统，系统必须通过客户端录入账号口令远程登录；

2、 系统内至少包含三个以上账号；

3、 系统服务器端可设定禁止登录的 IP 地址和账号信息；

4、 如果客户端从禁止的 IP 地址登录或使用禁止的账号登录则显示不允许登录，并断开连接。

题目六：加密传输系统

1、 设计客户端程序向服务器端发送数据；

2、 客户端从键盘输入的数据在发送之前进行加密，加密方法可选择仿射、移位密码；

3、 服务器端接收到数据后进行显示，然后在解密后再次显示。

题目七：图形验证模拟

要求：1、开发一个手机锁屏的图形验证程序，以字符命令行形式来实现要求完成的功能有：

2、 登录时输入用户名；

3、 输入 4*4 坐标下的图形点位置，用字符方式输入；

4、 输入完成后实现在服务器端对图形进行验证；

5、 至少有三个以上的用户验证。

举例：

显示：请输入用户名

输入：user

显示：请输入验证图形

O	O	O	O
O	O	O	O
O	O	O	O
O	O	O	O

输入：set（1,1）

显示：

X	O	O	O

O	O	O	O
O	O	O	O
O	O	O	O

输入：set（1,2）

显示：

X	X	O	O
O	O	O	O
O	O	O	O
O	O	O	O

中间过程省略

显示：

X	X	O	O
O	X	O	O
O	O	X	O

O	O	O	X

输入：set（1,2）
显示：

X	O	O	O
O	X	O	O
O	O	X	O
O	O	O	X

输入：verify
显示：登录成功

五、前期知识：

1）基于 MFC 的程序设计

通过编写简单的 Windows 窗口、基于 MFC 的计算器以及车站售票程序（多线程实现），深入了解 VC++的开发环境，掌握常用的控件使用方法和程序编写过程。

1. 利用 Windows API 函数编写 Windows 应用程序必须首先了解以下内容：

(1)窗口的概念

(2)事件驱动的概念

(3)句柄

(4)消息

2. Windows 应用程序常用消息

（1）WM_LBUTTONDOWN：产生单击鼠标左键的消息

（2）WM_KEYDOWN：按下一个非系统键时产生的消息

（3）WM_ CHAR：按下一个非系统键时产生的消息

（4）WM_CREATE：由 CreateWindow 函数发出的消息

（5）WM_CLOSE：关闭窗口时产生的消息

（6）WM_DESTROY:由 DestroyWiodow 函数发出的消息

（7）WM_QUIT：由 PostQuitMessage 函数发出的消息

（8）WM_PAINT

3. Windows 应用程序组成及编程步骤

（1）应用程序的组成

- C 语言源程序文件
- 头文件
- 模块定义文件
- 资源描述文件
- 项目文件

（2）源程序组成结构

入口函数 WinMain

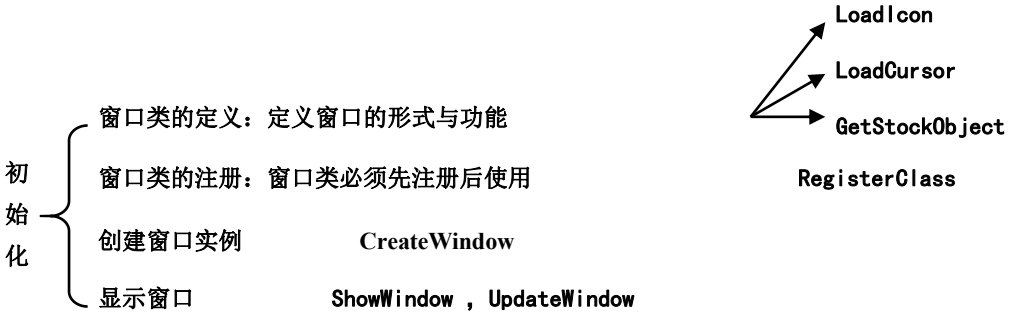
窗口函数 WndProc

①入口函数 WinMain

- WinMain 函数的说明如下：

```
int WINAPI WinMain
( HINSTANCE hThisInst,   //应用程序当前实例句柄
  HINSTANCE hPrevInst,   //应用程序其他实例句柄
  LPSTR lpszCmdLine,     //指向程序命令行参数的指针
  Int nCmdShow        //应用程序开始执行时窗口显示方式的整数值标识
)
```

- 初始化



- 消息循环

消息循环的常见格式如下：

```
MSG Msg;
…
```

```
while (GetMessage (&Msg,NULL,0,0))
{ TranslateMessage(&Msg);
         DispatchMessage(&Msg); }
```

②窗口函数的一般形式如下：

```
LRESULT CALLBACK WndProc(   HWND hwnd,   UINT messgae,
                  WPARAM wParam,LPARAM lParam )
{ …
  switch(message)    //  message 为标识的消息
  { case …
        …
      break;
     …

    case WM_DESTROY:
      PostQuitMessage(0);
    default:
      return DefWindowProc(hwnd,message,wParam,lParam);
  }
return(0);
```

2.多线程

（1）**HANDLE** CreateThread(

LPSECURITY_ATTRIBUTES lpThreadAttributes,

SIZE_T dwStackSize,

LPTHREAD_START_ROUTINE lpStartAddress,

LPVOID lpParameter,

DWORD dwCreationFlags,

LPDWORD lpThreadId

);

A: 第三个参数是个函数指针,指向某种特定的函数,调用约定是 WINAPI//#define WINAPI __stdcall, 参数 LPVOID 保障了函数的合法性.

B: 次函数两个得到两个值:第一个值是 HANDLE,大部分和线程有关的 API 函数都要使用它;第二个是参数 ThreadID 带回来的值,它是独一无二的表示一个进程中的某个线程.

说明:我们不可以从一个线程的 ID 从而得到其 HANDLE.

（2） BOOL CloseHandle(

HANDLE hObject

);

用来释放核心对象。

（3）创建一个新的线程

要创建一个线程，必须得有一个主进程，然后由这个主进程来创建一个线程，在一般的 VC 程序中，主函数所在的进程就是程序的主进程。

让我们从主函数来开始编写我们这个简单的小程序。我们知道 CreateThread 函数可以用来创建一个线程，在 MSDN 中查找这个函数得到如下信息："The CreateThread function creates a thread to execute within the address space of the calling process."和"If the function succeeds, the return value is a handle to the new thread."所以我们得定义一个句柄用来存放它的返回值。还定义一个指向线程 ID 的 DWORD 值 dwThreadId。然后我们就可以用 CreateThread 函数来创建我们的线程了，CreateThread 函数有六个参数分别是

LPSECURITY_ATTRIBUTES lpThreadAttributes, // pointer to security attributes

DWORD dwStackSize, // initial thread stack size

LPTHREAD_START_ROUTINE lpStartAddress, // pointer to thread function

LPVOID lpParameter, // argument for new thread

DWORD dwCreationFlags, // creation flags

LPDWORD lpThreadId // pointer to receive thread ID

其中第一个参数我们设置为 NULL，使这个句柄不能被继承；第二个参数设置为 0，使用默认的堆栈大小；第三个参数为线程函数的起始地址，也就是线程函数的函数名；第四个参数为 NULL，没有值要传递给线程函数；第五个参数为 0，创建好之后马上让线程运行；第六个参数设置为指向线程 ID 的地址。创建好线程之后，线程函数进行初始化之类的操作，主函数继续执行，此时可以输出被创建线程的 ID。我们在主函数中用 WaitForSingleObject 函数来等待线程函数变成受信(signaled)状态，它的两个参数分别是

HANDLE hHandle, // handle to object to wait for

DWORD dwMilliseconds // time-out interval in milliseconds

第一参数为线程函数的句柄，第二个参数设置为 INFINITE，等待线程一直执行完。在程序的最后还要记得用 CloseHandle 函数关闭线程，这样主函数就写完了。

在线程函数里面我们可以简单地做一些工作，比如设置一个循环，让它输出一定的信息等。源程序如下：

```
#include <windows.h>
#include <iostream.h>
```

```
DWORD WINAPI ThreadFunc(HANDLE Thread)
{
 int i;
 for(i=0;i<10;i++)
 {
  cout<<"A new thread has created!"<<endl;
 }
 return 0;
}
      int main(int argc,char* argv[])
{
 HANDLE Thread;
 DWORD dwThreadId;
 Thread=::CreateThread
  (NULL,0,ThreadFunc,NULL,0,&dwThreadId);
 cout<<"The new thread ID is :"<<dwThreadId<<endl;
 ::WaitForSingleObject(Thread,INFINITE);
 ::CloseHandle(Thread);
 return 0;
}
```

2）套接字编程练习

通过在 SDK 模式下完成数据通信的过程，掌握 Windows Socket 的常用函数的形式和使用方法，理解数据通信的过程。主要包括：Winsock 的启动与终止、Winsock 的创建及绑定和关闭、建立通信连接 listen 及 accept 和 connect、数据的传输、简单的客户机/服务器之间的通信。

在 VC 中进行 WINSOCK 的 API 编程开发，需要使用到下面三个文件：

1.WINSOCK.H: 这是 WINSOCK API 的头文件。

2.WSOCK32.LIB: WINSOCK API 连接库文件。在使用中，一点要把它作为项目的非缺省的连接库包含到项目文件中去。

3.WINSOCK.DLL: WINSOCK 的动态连接库，位于 WINDOWS 的安装目录下。

WINSOCK.DLL 位于 TCP/IP 协议栈和应用程序之间。也就是说，WINSOCK 管理与 TCP/IP 协议的接口。在一开始 WINSOCK 的应有开发时，你不必对 TCP/IP 协议有很深刻的了解。但是，如果想成为一个为网络编程的高手，就一定要对下层了解得十分清楚。

总的来说，使用 SOCKET 接口（面向连接或无连接）进行网络通信时，必须按下面简单

的四步进行处理：

1.程序必须建立一个 SOCKET。

2.程序必须按要求配置此 SOCKET。也就是说，程序要么将此 SOCKET 连接到远方的主机上，要么给此 SOCKET 指定一个本地协议端口。

3.程序必须按要求通过此 SOCKET 发送和接收数据。

4.程序必须关闭此 SOCKET。

（一）WinSock 原理

1.客户机/服务器模式

在 TCP/IP 网络中两个进程间的相互作用的主机模式是客户机/服务器模式(Client/Server model)。该模式的建立基于以下两点：（1）非对等作用；（2）通信完全是异步的。客户机/服务器模式在操作过程中采取的是主动请示方式：

首先服务器方要先启动，并根据请示提供相应服务：（过程如下）

（1）打开一通信通道并告知本地主机，它愿意在某一个公认地址上接收客户请求。

（2）等待客户请求到达该端口。

（3）接收到重复服务请求，处理该请求并发送应答信号。

（4）返回第二步，等待另一客户请求

（5）关闭服务器。

客户方：

（1）打开一通信通道，并连接到服务器所在主机的特定端口。

（2）向服务器发送服务请求报文，等待并接收应答；继续提出请求……

（3）请求结束后关闭通信通道并终止。

2.基本套接字

为了更好说明套接字编程原理，给出几个基本的套接字，在以后的篇幅中会给出更详细的使用说明。

创建套接字——socket()

应用程序在使用套接字前，首先必须拥有一个套接字，系统调用 socket()

向应用程序提供创建套接字的手段，其调用格式如下：

SOCKET socket(int af, int type, int protocol);

该调用要接收三个参数：af、type、protocol。参数 af 指定通信发生的区域，UNIX 系统支持的地址族有：AF_UNIX、AF_INET、AF_NS 等，而 DOS、WINDOWS 中仅支持 AF_INET，它是网际网区域。因此，地址族与协议族相同。参数 type 描述要建立的套接字的类型。参数 protocol 说明该套接字使用的特定协议，如果调用者不希望特别指定使用的协议，则置为 0，使用默认的连接模式。根据这三个参数建立一个套接字，并将相应的资源分配给它， 同时返回一个整型套接字号。因此，socket()系统调用实际上指定了相关五元组中的“协议”

这一元。

指定本地地址——bind()

当一个套接字用 socket()创建后，存在一个名字空间(地址族),但它没有被命名。bind()将套接字地址（包括本地主机地址和本地端口地址）与所创建的套接字号联系起来，即将名字赋予套接字，以指定本地半相关。其调用格式如下：

int bind(SOCKET s, const struct sockaddr FAR * name, int namelen);

参数 s 是由 socket() 调用返回的并且未作连接的套接字描述符(套接字号)。参数 name 是赋给套接字 s 的本地地址（名字），其长度可变，结构随通信域的不同而不同。namelen 表明了 name 的长度。

如果没有错误发生，bind()返回 0。否则返回值 SOCKET_ERROR。

地址在建立套接字通信过程中起着重要作用,作为一个网络应用程序设计者对套接字地址结构必须有明确认识。

3)建立套接字连接——connect()与 accept()

这两个系统调用用于完成一个完整相关的建立，其中 connect()用于建立连接。无连接的套接字进程也可以调用 connect()，但这时在进程之间没有实际的报文交换，调用将从本地操作系统直接返回。这样做的优点是程序员不必为每一数据指定目的地址，而且如果收到的一个数据报，其目的端口未与任何套接字建立“连接”，便能判断该端口不可操作。而 accept()用于使服务器等待来自某客户进程的实际连接。

connect()的调用格式如下：

int connect(SOCKET s,const struct sockaddr FAR * name,int namelen);

参数 s 是欲建立连接的本地套接字描述符。参数 name 指出说明对方套接字地址结构的指针。对方套接字地址长度由 namelen 说明。

如果没有错误发生，connect()返回 0。否则返回值 SOCKET_ERROR。在面向连接的协议中，该调用导致本地系统和外部系统之间连接实际建立。

由于地址族总被包含在套接字地址结构的前两个字节中，并通过 socket()调用与某个协议族相关。因此 bind()和 connect()无须协议作为参数。

accept()的调用格式如下：

SOCKET accept(SOCKET s,struct sockaddr FAR* addr,int FAR* addrlen);

参数 s 为本地套接字描述符，在用做 accept() 调用的参数前应该先调用过 listen()。addr 指向客户方套接字地址结构的指针， 用来接收连接实体的地址。addr 的确切格式由套接字创建时建立的地址族决定。addrlen 为客户方套接字地址的长度（字节数）。如果没有错误发生，accept()返回一个 SOCKET 类型的值，表示接收到的套接字的描述符。否则返回值 INVALID_SOCKET。

accept()用于面向连接服务器。参数 addr 和 addrlen 存放客户方的地址信息。调用前，参数 addr 指向一个初始值为空的地址结构,而 addrlen 的初始值为 0；调用 accept() 后，

服务器等待从编号为 s 的套接字上接受客户连接请求，而连接请求是由客户方的 connect()调用发出的。当有连接请求到达时， accept()调用将请求连接队列上的第一个客户方套接字地址及长度放入 addr 和 addrlen，并创建一个与 s 有相同特性的新套接字号。新的套接字可用于处理服务器并发请求。

四个套接字系统调用，socket()、bind()、connect()、accept()，可以完成一个完全五元相关的建立。socket()指定五元组中的协议元，它的用法与是否为客户或服务器、是否面向连接无关。bind()指定五元组中的本地二元，即本地主机地址和端口号，其用法与是否面向连接有关：在服务器方，无论是否面向连接，均要调用 bind() ；在客户方，若采用面向连接，则可以不调用 bind()，而通过 connect()自动完成。若采用无连接，客户方必须使用 bind()以获得一个唯一的地址。

以上讨论仅对客户/服务器模式而言，实际上套接字的使用是非常灵活的，唯一需遵循的原则是进程通信之前，必须建立完整的相关。

监听连接——listen()

此调用用于面向连接服务器，表明它愿意接收连接。listen()需在 accept ()之前调用，其调用格式如下：

```
int listen(SOCKET s, int backlog);
```

参数 s 标识一个本地已建立、尚未连接的套接字号， 服务器愿意从它上面接收请求。backlog 表示请求连接队列的最大长度， 用于限制排队请求的个数，目前允许的最大值为 5。如果没有错误发生，listen()返回 0。否则它返回 SOCKET_ERROR。

listen()在执行调用过程中可为没有调用过 bind() 的套接字 s 完成所必须的连接，并建立长度为 backlog 的请求连接队列。

调用 listen()是服务器接收一个连接请求的四个步骤中的第三步。它在调用 socket()分配一个流套接字，且调用 bind()给 s 赋于一个名字之后调用，而且一定要在 accept()之前调用。

数据传输——send()与 recv()

当一个连接建立以后，就可以传输数据了。常用的系统调用有 send() 和 recv()。

send() 调用用于在参数 s 指定的已连接的数据报或流套接字上发送输出数据，格式如下：

```
int send(SOCKET s, const char FAR *buf, int len, int flags);
```

参数 s 为已连接的本地套接字描述符。buf 指向存有发送数据的缓冲区的指针，其长度由 len 指定。flags 指定传输控制方式,如是否发送带外数据等。如果没有错误发生,send()返回总共发送的字节数。否则它返回 SOCKET_ERROR。

recv()调用用于在参数 s 指定的已连接的数据报或流套接字上接收输入数据，格式如下：

```
int recv(SOCKET s, char FAR *buf, int len, int flags);
```

参数 s 为已连接的套接字描述符。buf 指向接收输入数据缓冲区的指针，其长度由 len 指定。flags 指定传输控制方式，如是否接收带外数据等。如果没有错误发生，recv()返回总共接收的字节数。如果连接被关闭，返回 0。否则它返回 SOCKET_ERROR。

输入/输出多路复用——select()

select()调用用来检测一个或多个套接字的状态。对每一个套接字来说，这个调用可以请求读、写或错误状态方面的信息。请求给定状态的套接字集合由一个 fd_set 结构指示。在返回时，此结构被更新，以反映那些满足特定条件的套接字的子集，同时， select()调用返回满足条件的套接字的数目，其调用格式如下：

int select(int nfds, fd_set FAR * readfds, fd_set FAR * writefds, fd_set FAR * exceptfds, const struct timeval FAR * timeout);

参数 nfds 指明被检查的套接字描述符的值域，此变量一般被忽略。参数 readfds 指向要做读检测的套接字描述符集合的指针，调用者希望从中读取数据。 参数 writefds 指向要做写检测的套接字描述符集合的指针。exceptfds 指向要检测是否出错的套接字描述符集合的指针。timeout 指向 select()函数等待的最大时间，如果设为 NULL 则为阻塞操作。select()返回包含在 fd_set 结构中已准备好的套接字描述符的总数目，或者是发生错误则返回 SOCKET_ERROR。

关闭套接字——closesocket()

closesocket()关闭套接字 s，并释放分配给该套接字的资源；如果 s 涉及一个打开的 TCP 连接，则该连接被释放。closesocket()的调用格式如下：

BOOL closesocket(SOCKET s);

参数 s 待关闭的套接字描述符。如果没有错误发生，closesocket()返回 0。否则返回值 SOCKET_ERROR。

(7)多路复用——select()

功能：用来检测一个或多个套接字状态。

格式：int PASCAL FAR select(int nfds,fd_set FAR * readfds,fd_set FAR * writefds,

fd_set FAR * exceptfds,const struct timeval FAR * timeout);

参数：readfds:指向要做读检测的指针

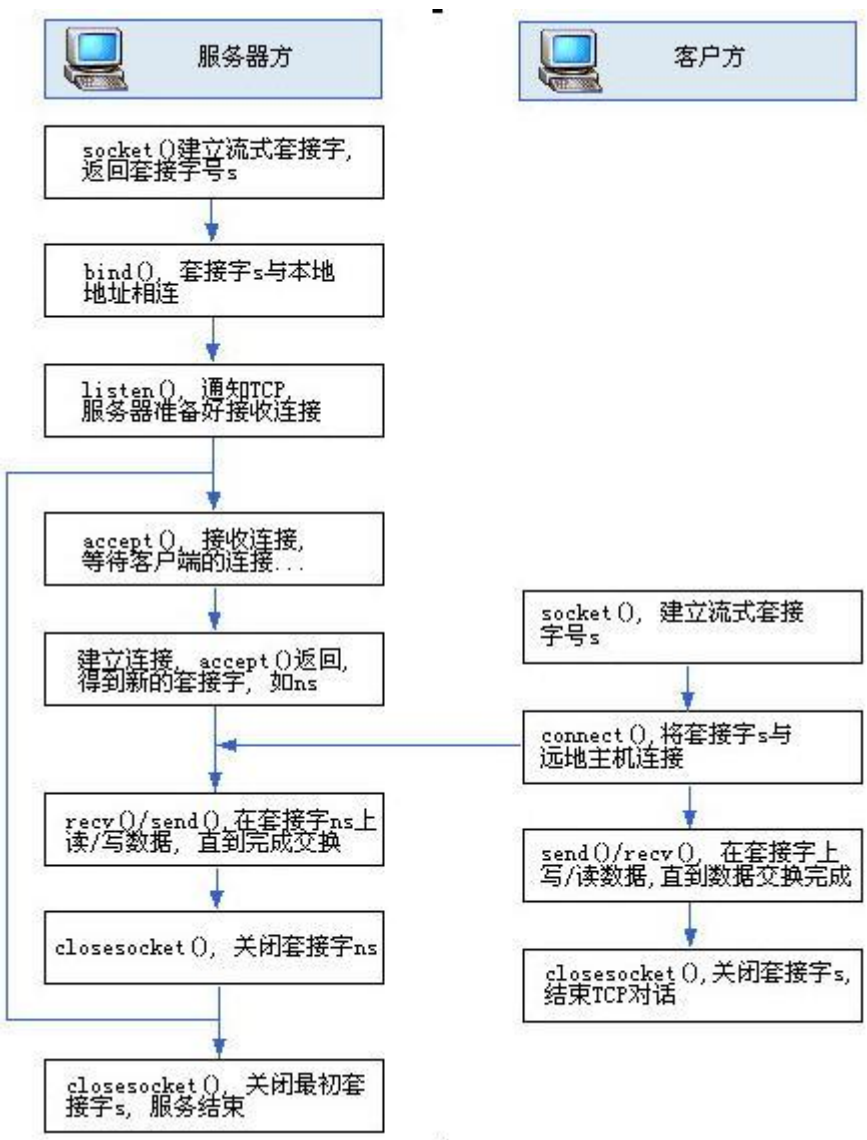
writefds:指向要做写检测的指针

exceptfds:指向要检测是否出错的指针

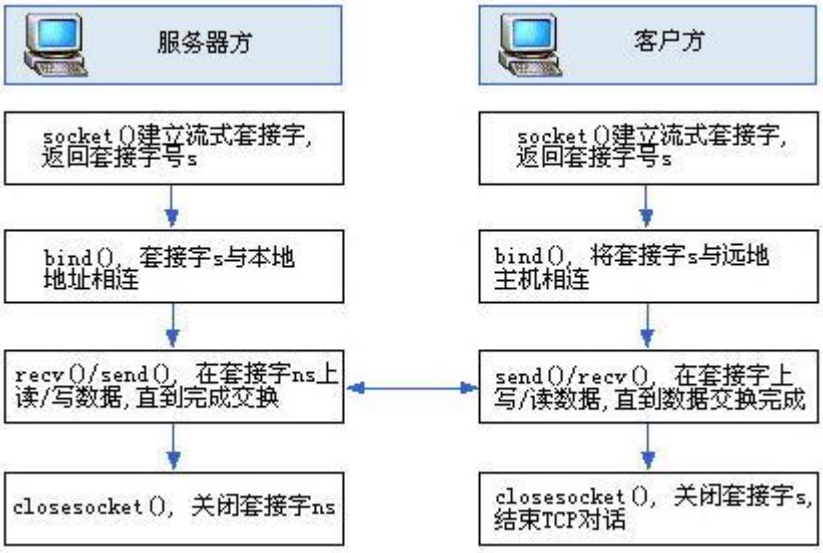
timeout:最大等待时间

3.典型过程图

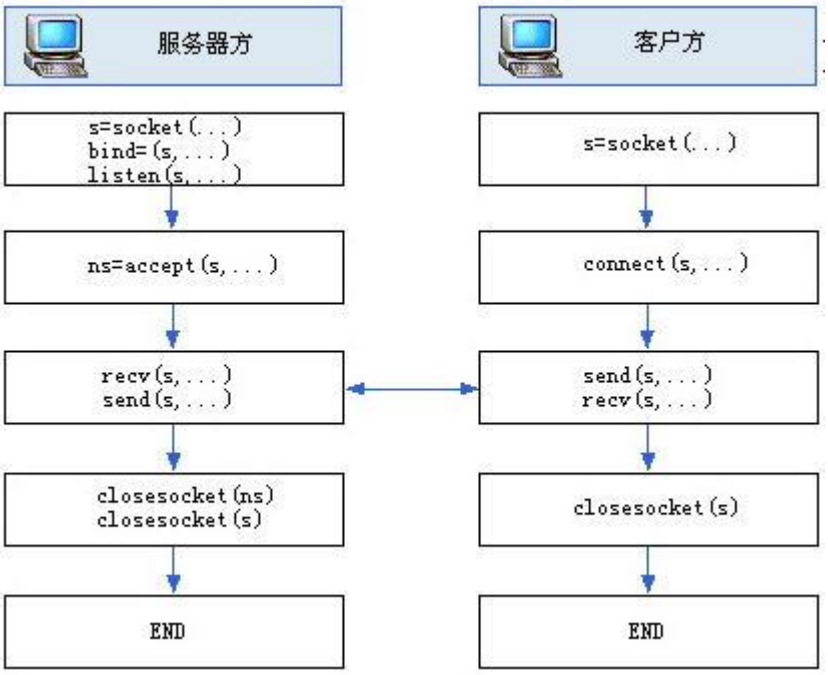
面向连接的套接字的系统调用时序图



无连接协议的套接字调用时序图



面向连接的应用程序流程图



4.WinSock 简介

Windows Sockets 是从 Berkeley Sockets 扩展而来的，其在继承 Berkeley Sockets 的基础上，又进行了新的扩充。这些扩充主要是提供了一些异步函数，并增加了符合 WINDOWS 消息驱动特性的网络事件异步选择机制。

Windows Sockets 由两部分组成：开发组件和运行组件。

开发组件：Windows Sockets 实现文档、应用程序接口(API)引入库和一些头文件。

运行组件：Windows Sockets 应用程序接口的动态链接库(WINSOCK.DLL)。

5.WinSock 主要扩充说明

（1）异步选择机制：

Windows Sockets 的异步选择函数提供了消息机制的网络事件选择，当使用它登记网络事件发生时，应用程序相应窗口函数将收到一个消息，消息中指示了发生的网络事件，以及与事件相关的一些信息。

Windows Sockets 提供了一个异步选择函数 WSAAsyncSelect()，用它来注册应用程序感兴趣的网络事件，当这些事件发生时，应用程序相应的窗口函数将收到一个消息。

函数结构如下：

```
int PASCAL FAR WSAAsyncSelect(SOCKET s,HWND hWnd,unsigned int wMsg,
    long lEvent);
```

参数说明：

hWnd：窗口句柄

wMsg：需要发送的消息

lEvent：事件（以下为事件的内容）

值：	含义：
FD_READ	期望在套接字上收到数据（即读准备好）时接到通知
FD_WRITE	期望在套接字上可发送数据（即写准备好）时接到通知
FD_OOB	期望在套接字上有带外数据到达时接到通知
FD_ACCEPT	期望在套接字上有外来连接时接到通知
FD_CONNECT	期望在套接字连接建立完成时接到通知
FD_CLOSE	期望在套接字关闭时接到通知

例如：我们要在套接字读准备好或写准备好时接到通知，语句如下：

rc=WSAAsyncSelect(s,hWnd,wMsg,FD_READ|FD_WRITE);

如果我们需要注销对套接字网络事件的消息发送，只要将 lEvent 设置为 0

（2）异步请求函数

在 Berkeley Sockets 中请求服务是阻塞的,WINDOWS SICKETS 除了支持这一类函数外，还增加了相应的异步请求函数(WSAAsyncGetXByY();)。

（3）阻塞处理方法

Windows Sockets 为了实现当一个应用程序的套接字调用处于阻塞时，能够放弃 CPU 让其它应用程序运行，它在调用处于阻塞时便进入一个叫“HOOK”的例程，此例程负责接收和分配 WINDOWS 消息，使得其它应用程序仍然能够接收到自己的消息并取得控制权。

WINDOWS 是非抢先的多任务环境，即若一个程序不主动放弃其控制权，别的程序就不能执行。因此在设计 Windows Sockets 程序时，尽管系统支持阻塞操作，但还是反对程序员使用该操作。但由于 SUN 公司下的 Berkeley Sockets 的套接字默认操作是阻塞的，WINDOWS 作为移植的 SOCKETS 也不可避免对这个操作支持。

在 Windows Sockets 实现中，对于不能立即完成的阻塞操作做如下处理：DLL 初始化→循环操作。在循环中，它发送任何 WINDOWS 消息，并检查这个 Windows Sockets 调用是否完成，在必要时，它可以放弃 CPU 让其它应用程序执行。我们可以调用 WSACancelBlockingCall() 函数取消此阻塞操作。

在 Windows Sockets 中，有一个默认的阻塞处理例程 BlockingHook() 简单地获取并发送 WINDOWS 消息。如果要对复杂程序进行处理，Windows Sockets 中还有 WSASetBlockingHook() 提供用户安装自己的阻塞处理例程能力；与该函数相对应的则是 WSAUnhookBlockingHook()，它用于删除先前安装的任何阻塞处理例程，并重新安装默认的处理例程。请注意，设计自己的阻塞处理例程时，除了函数 WSACancelBlockingHook() 之外，它不能使用其它的 Windows Sockets API 函数。在处理例程中调用 WSACancelBlockingHook()函数将取消处于阻塞的操作，它将结束阻塞循环。

（4）出错处理

Windows Sockets 为了和以后多线程环境（WINDOWS/UNIX）兼容，它提供了两个出错处理函数来获取和设置当前线程的最近错误号。（WSAGetLastEror()和 WSASetLastError()）

（5）启动与终止

使用函数 WSAStartup() 和 WSACleanup() 启动和终止套接字。

6.Windows Sockets 网络程序设计核心

我们终于可以开始真正的 Windows Sockets 网络程序设计了。不过我们还是先看一看每个 Windows Sockets 网络程序都要涉及的内容。让我们一步步慢慢走。

（1）启动与终止

在所有 Windows Sockets 函数中，只有启动函数 WSAStartup()和终止函数 WSACleanup()是必须使用的。

启动函数必须是第一个使用的函数，而且它允许指定 Windows Sockets API 的版本，并获得 SOCKETS 的特定的一些技术细节。本结构如下：

int PASCAL FAR WSAStartup(WORD wVersionRequested, LPWSADATA lpWSAData);

其中 wVersionRequested 保证 SOCKETS 可正常运行的 DLL 版本，如果不支持，则返回错误信息。

我们看一下下面这段代码，看一下如何进行 WSAStartup() 的调用

```
WORD wVersionRequested; // 定义版本信息变量
WSADATA wsaData; //定义数据信息变量
int err; //定义错误号变量
wVersionRequested = MAKEWORD(1,1);//给版本信息赋值
err = WSAStartup(wVersionRequested, &wsaData);//给错误信息赋值
if(err!=0)
{
    return;//告诉用户找不到合适的版本
}
//确认 Windows Sockets DLL 支持 1.1 版本
//DLL 版本可以高于 1.1
//系统返回的版本号始终是最低要求的 1.1，即应用程序与 DLL 中可支持的最低版本号
if(LOBYTE(wsaData.wVersion)!= 1|| HIBYTE(wsaData.wVersion)!=1)
{
    WSACleanup();//告诉用户找不到合适的版本
    return;
}
//Windows Sockets DLL 被进程接受，可以进入下一步操作
```

关闭函数使用时，任何打开并已连接的 SOCK_STREAM 套接字被复位，但那些已由

closesocket() 函数关闭的但仍有未发送数据的套接字不受影响，未发送的数据仍将被发送。程序运行时可能会多次调用 WSAStartuo() 函数，但必须保证每次调用时的 wVersionRequested 的值是相同的。

（2）异步请求服务

Windows Sockets 除支持 Berkeley Sockets 中同步请求，还增加了了一类异步请求服务函数 WSAAsyncGerXByY()。该函数是阻塞请求函数的异步版本。应用程序调用它时，由 Windows Sockets DLL 初始化这一操作并返回调用者，此函数返回一个异步句柄，用来标识这个操作。当结果存储在调用者提供的缓冲区，并且发送一个消息到应用程序相应窗口。常用结构如下：

```
HANDLE taskHnd;
char hostname="rs6000";
taskHnd = WSAAsyncBetHostByName(hWnd,wMsg,hostname,buf,buflen);
```

需要注意的是，由于 Windows 的内存对像可以设置为可移动和可丢弃，因此在操作内存对象是，必须保证 WIindows Sockets DLL 对象是可用的。

（3）异步数据传输

使用 send() 或 sendto() 函数来发送数据，使用 recv() 或 recvfrom() 来接收数据。Windows Sockets 不鼓励用户使用阻塞方式传输数据，因为那样可能会阻塞整个 Windows 环境。下面我们看一个异步数据传输实例：

假设套接字 s 在连接建立后，已经使用了函数 WSAAsyncSelect() 在其上注册了网络事件 FD_READ 和 FD_WRITE，并且 wMsg 值为 UM_SOCK，那么我们可以在 Windows 消息循环中增加如下的分支语句：

```
case UM_SOCK:
    switch(lParam)
    {
    case FD_READ:
        len = recv(wParam,lpBuffer,length,0);
        break;
    case FD_WRITE:
        while(send(wParam,lpBuffer,len,0)!=SOCKET_ERROR)
        break;
    }
break;
```

（4）出错处理

Windows 提供了一个函数来获取最近的错误码 WSAGetLastError(),推荐的编写方式如下：

```
len = send (s,lpBuffer,len,0);
if((len==SOCKET_ERROR)&&(WSAGetLastError()==WSAWOULDBLOCK)){...}
```

（二）关于使用套接字编程的一些基本概念

半相关与全相关

半相关

在网络中用一个三元组可以在全局唯一标志一个进程：（协议，本地地址，本地端口号）这样一个三元组，叫做一个半相关（half-association），它指定连接的每半部分。

全相关

一个完整的网间进程通信需要由两个进程组成，并且只能使用同一种高层协议。也就是说，不可能通信的一端用 TCP 协议，而另一端用 UDP 协议。因此一个完整的网间通信需要一个五元组来标识：

（协议，本地地址，本地端口号，远地地址，远地端口号）

这样一个五元组，叫做一个相关（association），即两个协议相同的半相关才能组合成一个合适的相关，或完全指定组成一连接。

TCP/IP 协议的地址结构为：

```
struct sockaddr_in{
    short sin_family;                   /*AF_INET*/
    u_short sin_port;                   /*16 位端口号，网络字节顺序*/
    struct in_addr sin_addr;            /*32 位 IP 地址，网络字节顺序*/
    char sin_zero[8];                   /*保留*/
}
```

(c)套接字类型

TCP/IP 的 socket 提供下列三种类型套接字。

流式套接字（SOCK_STREAM）

提供了一个面向连接、可靠的数据传输服务，数据无差错、无重复地发送，且发送顺序接收。内设流量控制，避免数据流超限；数据被看作是字节流，无长度限制。文件传送协议（FTP）即使用流式套接字。

数据报式套接字（SOCK_DGRAM）

提供了一个无连接服务。数据包以独立包形式被发送，不提供无错保证，数据可能丢

失或重复，并且接收顺序混乱。网络文件系统（NFS）使用数据报式套接字。

原始式套接字（SOCK_RAW）

该接口允许对较低层协议，如 IP、ICMP 直接访问。常用于检验新的协议实现或访问现有服务中配置的新设备。

3）TCP 套接字编程

通过编写简单的 Windows 窗口、基于 MFC 的计算器以及车站售票程序（多线程实现），深入了解 VC++的开发环境，掌握常用的控件使用方法和程序编写过程。主要内容包括：1.主机间 TCP 的性能测试之一：回程时延。2.服务器端能从客户端接收数据并立即将接收到的数据原样返回给客户方。3.客户端能往服务器端发送数据，然后立即接受从服务器端原样返回的数据。

(一)WinSock 的扩充

异步方式指的是发送方不等接收方响应，便接着发下个数据包的通信方式；而同步指发送方发出数据后，等收到接收方发回的响应，才发下一个数据包的通信方式。

阻塞套接字是指执行此套接字的网络调用时，直到成功才返回，否则一直阻塞在此网络调用上，比如调用 recv()函数读取网络缓冲区中的数据，如果没有数据到达，将一直挂在 recv()这个函数调用上，直到读到一些数据，此函数调用才返回；而非阻塞套接字是指执行此套接字的网络调用时，不管是否执行成功，都立即返回。比如调用 recv()函数读取网络缓冲区中数据，不管是否读到数据都立即返回，而不会一直挂在此函数调用上。在实际 Windows 网络通信软件开发中，异步非阻塞套接字是用的最多的。

（同步阻塞、异步非阻塞）

1.默认用作同步阻塞方式，那就是当你从不调用 WSAIoctl()和 ioctlsocket()来改变 Socket IO 模式，也从不调用 WSAAsyncSelect()和 WSAEventSelect()来选择需要处理的 Socket 事件。正是由于函数 accept()，WSAAccept()，connect()，WSAConnect()，send()，WSASend()，recv()，WSARecv()等函数被用作阻塞方式，所以可能你需要放在专门的线程里，这样以不影响主程序的运行和主窗口的刷新。

2.如果作为异步非阻塞方式用，那么程序主要就是要处理事件。它有两种处理事件的办法：

第一种，它常关联一个窗口，也就是异步 Socket 的事件将作为消息发往该窗口，这是由 WinSock 扩展规范里的一个函数 WSAAsyncSelect()来实现和窗口关联。最终你只需要处理窗口消息，来收发数据。

第二种，用到了扩展规范里另一个关于事件的函数 WSAEventSelect()，它是用事件对象的方式来处理 Socket 事件，也就是，你必须首先用 WSACreateEvent()来创建一个事件对象，然后调用 WSAEventSelect()来使得 Socket 的事件和这个事件对象关联。最终你将要在一个线程里用 WSAWaitForMultipleEvents()来等待这个事件对象被触发。这个过程也稍显复杂。

windows socket api

WINSOCK 对 BSD Socket 的扩充主要是在基于消息、对网络事件的异步存取接口上。下表列出了 WINSOCK 扩充的函数功能。

函　数　名	功　　能
WSAAsyncGetHostByAddr()	标准 Berkeley 函数 getXbyY 的异步版本，例
WSAAsyncGetHostByName()	如:函数 WSAAsyncGetHostByName()就是提
WSAAsyncGetProtoByName()	供了标准 Berkeley 函数 gethostbyname 的一
WSAAsyncGetProtoByNumber()	种基于消息的异步实现。
WSAAsyncGetServByName()	
WSAAsyncGetServByPort()	
WSAAsyncSelect()	函数 select()的异步版本
WSACancelAsyncRequest()	取消函数 WSAAsyncGetXByY 执行中的实例
WSACancelBlockingCall()	取消一个执行中的"阻塞"API 调用
WSACleanup()	终止使用隐含的 Windows Sockets DLL
WSAGetLastError()	获取 Windows Sockets API 的最近错误号
WSAIsBlocking()	检测隐含的 Windows Sockets DLL 是否阻塞了一个当前线索的调用
WSASetBlockingHook()	设置应用程序自己的"阻塞"处理函数
WSASetLastError()	设置 Windows Sockets API 的最近错误号
WSAStartup()	初始化隐含的 Windows Sockets DLL
WSAUnhookBlockingHook()	恢复原来的"阻塞"处理函数

从表 1 可以看出，WINSOCK 的扩充功能可以分为如下几类：

（1）异步选择机制：

异步选择函数 WSAAsyncSelect()允许应用程序提名一个或多个感兴趣的网络事件，所有阻塞的网络 I/O 例程（如 send()和 resv()），不管它是已经使用还是即将使用，都

可作为WSAAsyncSelect()函数选择的候选。当被提名的网络事件发生时，Windows 应用程序的窗口函数将收到一个消息，消息附带的参数指示被提名过的某一网络事件。

（２）异步请求例程：

异步请求例程允许应用程序用异步方式获取请求的信息，如 WSAAsyncGetXByY()类函数允许用户请求异步服务，这些功能在使用标准 Berkeley 函数时是阻塞的。函数 WSACancelAsyncRequest()允许用户终止一个正在执行的异步请求。

（３）阻塞处理方法：

WINSOCK 在调用处于阻塞时进入一个叫“Hook”的例程,它负责处理 Windows 消息，使得 Windows 的消息循环能够继续。WINSOCK 还提供了两个函数（WSASetBlockingHook()和 WSAUnhookBlockingHook()）让用户能够设置和取消自己的阻塞处理例程。另外，函数 WSAIsBlocking()可以检测调用是否阻塞，函数 WSACancelBlockingCall()可以取消一个阻塞的调用。

（４）出错处理：

为了和以后的多线索环境（如 Windows/NT）兼容，WINSOCK 提供了两个出错处理函数 WSAGetLastError()和 WSASetLastError()来获取和设置本线索的最近错误号。

（５）启动与终止：

WINSOCK 的应用程序在使用上述 WINSOCK 函数前,必须先调用 WSAStartup()函数对 Windows Sockets DLL 进行初始化，以协商 WINSOCK 的版本支持，并分配必要的资源。在应用程序退出之前，应该先调用函数 WSAClearnup()终止对 Windows Sockets DLL 的使用，并释放资源，以利下一次使用。

在这些函数中，实现 Windows 网络实时通信的关键是异步选择函数 WSAAsyncSelect()的使用，其原型如下：

int PASCAL FAR WSAAsyncSelect(SOCTET s,HWND hWnd,unsigned int wMsg,long lEvent);

它请求 Windows Sockets DLL 在检测到在套接字 s 上发生的 lEvent 事件时,向窗口 hWnd 发送一个消息 wMsg。它自动地设置套接字 s 处于非阻塞工作方式。参数 lEvent 由下列事件的一个或多个组成：

值	含　　义
FD_READ	希望在套接字 s 收到数据（即读准备好）时接到通知
FD_WRITE	希望在套接字 s 可发送数据（即写准备好）时接到通知
FD_OOB	希望在套接字 s 上有带外数据到达时接到通知
FD_ACCEPT	希望在套接字 s 上有外部连接到来时接到通知
FD_CONNECT	希望在套接字 s 连接建立完成时接到通知
FD_CLOSE	希望在套接字 s 关闭时接到通知

表２．　异步选择网络事件表

例如，我们要在套接字 s 读准备好或写准备好时接到通知，可以使用下面的语句：

rc = WSAAsyncSelect(s, hWnd, wMsg, FD_READ | FD_WRITE);

当套接字 s 上被提名的一个网络事件发生时，窗口 hWnd 将收到消息 wMsg，变量 lParam 的低字指示网络发生的事件，高字指示错误码。应用程序就可以通过这些信息来决定自己的下一步动作。

（二）一个简单的 TCP 客户-服务员程序的服务员程序

1.面向连接的服务器程序：

```
#include <PROCESS.H>
#include <windows.h>
#include <winsock.h>
#include <sys/types.h>
#include <fcntl.h>
#include <wsipx.h>
#include <wsnwlink.h>
#include <stdio.h>
#define SERV_TCP_PORT 6000                    /*服务员进程端口号，视具体情况而定*/
#define SERV_HOST_ADDR "10.60.46.40"     /*服务员 IP，视具体情况而定*/
int sockfd;
   ////////////////////////////////////////////////////////////////////////
////    线    程    用    来    处    理    客    户    端    的    请求
            //
 //服务员主进程每与某客户端建立一个连接之后，便启动一个新的线程来处理接下//
 // 客 户 端 的 请 求 ， 参 数 为 服 务 员 与 该 客 户 端 的 连 接 点 ：socket                    //
   ////////////////////////////////////////////////////////////////////////
//
DWORD ClientThread(void *pVoid)
{
int nRet;
char szBuf[1024];
memset(szBuf, 0, sizeof(szBuf));
 /*接收来自客户端的数据信息*/
nRet = recv((SOCKET )pVoid,                  // 与客户端连接的 socket
```

```
            szBuf,                                                  /
/ 接收缓冲区
            sizeof(szBuf),                                          // 缓冲区
长度
            0);
   // Flags
    /*错误处理*/
    if (nRet == INVALID_SOCKET)
    {
    printf("recv()");
     closesocket(sockfd);
     closesocket((SOCKET)pVoid);
     return 0;
     }
    /*显示接收到的数据*/
                        printf("\nData received: %s\n", szBuf);
                        /* 发送数据给客户端*/
                        strcpy(szBuf, "From the Server");                   // 发送
内容
                                                                    nRet     =
send((SOCKET)pVoid,                                  // 与客户端连接的 socket
                                                    szBuf,
                      // 数据缓冲区
                                                    strlen(szBuf),
            // 数据长度
                                                     0);
                          // Flags
                        /*结束连接，释放 socket*/
                        closesocket((SOCKET)pVoid);
                        return 0;
                  }
                  ////////////////////////////////////////////////////////
                                                    //   服   务   员   主   程
序：
          //
```

```
                    //在一个众所周知的端口上等待客户的连接请求                    //
                //有请求到来时建立与客户端的连接,并启动一个线程处理该请求    //
                ////////////////////////////////////////////////////////////
                int main()
                {
                  int clilen;
                  int pHandle=-1;
                  struct sockaddr_in serv_addr;
                  SOCKET           socketClient;
                  DWORD         ThreadAddr;
                  HANDLE           dwClientThread;
                  SOCKADDR_IN   SockAddr;
                  /*初始化 Winsock API，即连接 Winsock 库*/
                  WORD wVersionRequested = MAKEWORD(1, 1);
                  WSADATA wsaData;
                  if (WSAStartup(wVersionRequested, &wsaData)) {
                  printf("WSAStartup failed %s\n", WSAGetLastError());
                  return -1;
                  }
                  /*打开一个 TCP SOCKET */
                  if((sockfd=socket(AF_INET, SOCK_STREAM,0))<0)
                          printf("server:can't open stream socker\n");
                  /*绑定本地地址，以便客户端连接*/
                                  memset((char  *)&serv_addr,0,sizeof(struct
sockaddr_in));
                          serv_addr.sin_family=AF_INET;
                          serv_addr.sin_addr.s_addr=htonl(INADDR_ANY);
                          serv_addr.sin_port=htons(SERV_TCP_PORT);
                               if(bind(sockfd,(struct  sockaddr  *)&serv_addr,
sizeof(serv_addr))<0)
                                  printf("server: can't bind local address");
                  /*设置服务员的最大连接数为 15*/
                  listen(sockfd,5);
                  /*循环等待来自客户端的连接请求*/
```

```
                    while(1)
                    {
                            /*阻塞等待一个请求的到来*/
                            clilen = sizeof(SOCKADDR_IN);
                            socketClient = accept(sockfd,
                                                                    (LPSOCKADD
R)&SockAddr,
                                                                      &clilen);
<
                            /*出错处理*/
                            if (socketClient == INVALID_SOCKET)
                            {
                                    printf("accept failed!\n");
                                    break;
                              }
                            /*打印已建立的连接信息*/
                            printf("Connection accepted on socket:%d from:%s\n",
                                            socketClient,
                                            inet_ntoa(SockAddr.sin_addr));
                          /*启动一个新线程处理该请求*/
                          dwClientThread =CreateThread(NULL,

      0,

      (LPTHREAD_START_ROUTINE)&ClientThread,

      (void *)socketClient,

      0,

      &ThreadAddr);
                          /*错误处理*/
                          if(!dwClientThread)
                                  printf("Cannot start client thread...");
                  /*线程建立以后，主程序里不再使用线程 handle,将其关闭，但线程继
```

续运行*/

```
            CloseHandle((HANDLE)dwClientThread);
        }
        /*结束 windows sockets API*/
        WSACleanup();
        return 0;
        }
```

上述服务员程序已经在服务器端运行，请学生认真阅读，然后根据实验原理二中介绍的内容，设计面向连接的客户端程序，实现客户与服务员间的数据传输。

在仿真机一端运行客户端进程，在监控机端捕获数据并分析。

4）基于 MFC Socket 类的网络编程

以设计实现一个简单的聊天系统（包括客户端和服务器端）为例，理解 MFC 的 Socket 类同 Socket API 之间的区别以及 MFC 的两种类之间的联系与区别。下面。

1、聊天室程序的设计

（1）实现思想

在 Internet 上的聊天室程序一般都是以服务器提供服务端连接响应，使用者通过客户端程序登录到服务器，就可以与登录在同一服务器上的用户交谈，这是一个面向连接的通信过程。因此，程序要在 TCP/IP 环境下，实现服务器端和客户端两部分程序。

（2）服务器端工作流程

服务器端通过 socket()系统调用创建一个 Socket 数组后(即设定了接受连接客户的最大数目)，与指定的本地端口绑定 bind()，就可以在端口进行侦听 listen()。如果有客户端连接请求，则在数组中选择一个空 Socket，将客户端地址赋给这个 Socket。然后登录成功的客户就可以在服务器上聊天了。

（3）客户端工作流程

客户端程序相对简单,只需要建立一个 Socket 与服务器端连接,成功后通过这个 Socket 来发送和接收数据就可以了。

2、实现步骤

1.启动 Visual C++6.0，创建一个控制台项目工程 chat。在此项目工程中添加 Client 和 Server 两个项目。

2.设计服务器端界面并添加相应代码

3.设计客户端并添加相应代码。

4.分别打开服务器和客户端进行验证。

服务器端关键代码如下：

开启服务器功能：

```
    void OnServerOpen() //开启服务器功能
{
  WSADATA wsaData;
  int iErrorCode;
  char chInfo[64];
  if (WSAStartup(WINSOCK_VERSION, &wsaData)) //调用 Windows Sockets DLL
    { MessageBeep(MB_ICONSTOP);
      MessageBox("Winsock 无法初始化!", AfxGetAppName(), MB_OK|MB_ICONSTOP);
      WSACleanup();
      return; }
  else
    WSACleanup();
    if (gethostname(chInfo, sizeof(chInfo)))
    { ReportWinsockErr("\n 无法获取主机!\n ");
      return; }
    CString csWinsockID = "\n==>>服务器功能开启在端口：No. ";
    csWinsockID += itoa(m_pDoc->m_nServerPort, chInfo, 10);
    csWinsockID += "\n";
    PrintString(csWinsockID); //在程序视图显示提示信息的函数，读者可自行创建
    m_pDoc->m_hServerSocket=socket(PF_INET, SOCK_STREAM, DEFAULT_PROTOCOL);
    //创建服务器端 Socket，类型为 SOCK_STREAM，面向连接的通信
    if (m_pDoc->m_hServerSocket == INVALID_SOCKET)
    { ReportWinsockErr("无法创建服务器 socket!");
      return;}
    m_pDoc->m_sockServerAddr.sin_family = AF_INET;
    m_pDoc->m_sockServerAddr.sin_addr.s_addr = INADDR_ANY;
    m_pDoc->m_sockServerAddr.sin_port = htons(m_pDoc->m_nServerPort);
    if (bind(m_pDoc->m_hServerSocket, (LPSOCKADDR)&m_pDoc->m_sockServerAddr,
         sizeof(m_pDoc->m_sockServerAddr)) == SOCKET_ERROR) //与选定的端口绑定
      {ReportWinsockErr("无法绑定服务器 socket!");
        return;}
      iErrorCode=WSAAsyncSelect(m_pDoc->m_hServerSocket,m_hWnd,
      WM_SERVER_ACCEPT, FD_ACCEPT);
      //设定服务器相应的网络事件为 FD_ACCEPT，即连接请求，
      // 产生相应传递给窗口的消息为 WM_SERVER_ACCEPT
```

```
    if (iErrorCode == SOCKET_ERROR)
      { ReportWinsockErr("WSAAsyncSelect 设定失败!");
        return;}
    if (listen(m_pDoc->m_hServerSocket, QUEUE_SIZE) == SOCKET_ERROR) //开始监听
客户连接请求
      {ReportWinsockErr("服务器 socket 监听失败!");
        m_pParentMenu->EnableMenuItem(ID_SERVER_OPEN, MF_ENABLED);
        return;}
    m_bServerIsOpen = TRUE; //监视服务器是否打开的变量
  return;
}
    响应客户发送聊天文字到服务器：ON_MESSAGE(WM_CLIENT_READ, OnClientRead)
    LRESULT OnClientRead(WPARAM wParam, LPARAM lParam)
{
  int iRead;
  int iBufferLength;
  int iEnd;
  int iRemainSpace;
  char chInBuffer[1024];
  int i;
  for(i=0;(i<MAXCLIENT)&&(M_ACLIENTSOCKET[I]!=WPARAM);I++)
    //MAXClient 是服务器可响应连接的最大数目
    {}
  if(i==MAXClient) return 0L;
    iBufferLength = iRemainSpace = sizeof(chInBuffer);
    iEnd = 0;
    iRemainSpace -= iEnd;
        iBytesRead   =   recv(m_aClientSocket[i],   (LPSTR)(chInBuffer+iEnd),
iSpaceRemaining, NO_FLAGS);     //用可控缓冲接收函数 recv()来接收字符
    iEnd+=iRead;
  if (iBytesRead == SOCKET_ERROR)
    ReportWinsockErr("recv 出错!");
    chInBuffer[iEnd] = '\0';
  if (lstrlen(chInBuffer) != 0)
    {PrintString(chInBuffer); //服务器端文字显示
```

```
      OnServerBroadcast(chInBuffer); //自己编写的函数，向所有连接的客户广播这个客户的聊天文字
    }
  return(0L);
}
```

对于客户断开连接，会产生一个 FD_CLOSE 消息，只须相应地用 closesocket()关闭相应的 Socket 即可，这个处理比较简单。

客户端关键代码：

连接到服务器：

```
    void OnSocketConnect()
{ WSADATA wsaData;
  DWORD dwIPAddr;
  SOCKADDR_IN sockAddr;
  if(WSAStartup(WINSOCK_VERSION,&wsaData)) //调用 Windows Sockets DLL
  {MessageBox("Winsock 无法初始化!",NULL,MB_OK);
    return;
  }
  m_hSocket=socket(PF_INET,SOCK_STREAM,0); //创建面向连接的 socket
  sockAddr.sin_family=AF_INET; //使用 TCP/IP 协议
  sockAddr.sin_port=m_iPort; //客户端指定的 IP 地址
  sockAddr.sin_addr.S_un.S_addr=dwIPAddr;
  int nConnect=connect(m_hSocket,(LPSOCKADDR)&sockAddr,sizeof(sockAddr)); //请求连接
  if(nConnect)
    ReportWinsockErr("连接失败!");
  else
    MessageBox("连接成功!",NULL,MB_OK);
    int iErrorCode=WSAAsyncSelect(m_hSocket,m_hWnd,WM_SOCKET_READ,FD_READ);
    //指定响应的事件，为服务器发送来字符
  if(iErrorCode==SOCKET_ERROR)
  MessageBox("WSAAsyncSelect 设定失败!");
}
```

接收服务器端发送的字符也使用可控缓冲接收函数 recv()，客户端聊天的字符发送使用数据可控缓冲发送函数 send()，这两个过程比较简单，在此就不加赘述了。