实 验 十三 RawSock 编程

13.1 实验目的

本实验内容是利用 windows 平台的 WinpCap 开发包开发 RawSock 通信应用，它能基于 IP 等网络设计较具难度的网络层通信应用，程序通过封装 Ping 功能网络包说明原始套接字的使用。

13.2 实验内容

13.2.1 原始套接字介绍

创建 Socket 对象时使用到两个枚举，ProtocolType 枚举指示在创建 Socket 对象时的协议，包括：IP、Icmp、Tcp、Udp、Ipx 等。SocketType 枚举指示在创建 Socket 对象时的类型，包括：Stream、Dgram、Raw、Seqpacket、Unknown。流式套接字 (Stream) 对应于网络协议中的 TCP 通信，数据报套接字 (Dgram) 对应网络协议中的 UDP 通信，TCP 和 UDP 是传输层的网络协议。原始套接字 (Raw) 工作在网络层，它能构造网络层的多种协议，如 IP 协议，ICMP 协议，IGMP 协议等可以使用原始套接字来实现很多功能，原始套接字主要用于一些协议的开发，可以进行比较底层的操作，比如最基本的数据包分析，主机嗅探等，也可以使用原始套接字作一个自定义的传输层协议。原始套接字的功能强大，编写的难度也比流式套接字和数据报套接字大。原始套接字的创建方法与流式或数据报的套接字创建的方法是一致的，下面即是创建用于 ICMP 协议的套接字对象。

Socket socket =new Socket(AddressFamily.InterNetwork, SocketType.Raw, ProtocolType.Icmp);

其中 SocketType.Raw 参数指明要创建是原始套接字对象，ProtocolType.Icmp 参数指明 Socket 支持的协议是网际组管理协议 (ICMP)。

13.2.2 Ping 类的使用

应用程序可以使用 Ping 类检测是否可访问远程计算机。Ping 类尝试将 Internet 控制消息协议 (ICMP) 回送请求发送到远程计算机并接收该计算机通过 ICMP 回送答复消息返回的信息。Ping 类使用 PingReply 类的实例返回有关操作的信息，如操作状态以及发送请求和接收答复所用的时间。Pint 类的 Send 和 SendAsync 方法将 Internet 控制消息协议 (ICMP) 回送请求消息发送到远程计算机并等待来自该计算机的 ICMP 回送答复消息。Send 是同步方法直接返回 PingReply 类的实例，SendAsync 是异步方法，当异步操作完成时，它会引发 PingCompleted 事件，在 PingCompletedEventHandler 方法的 PingCompletedEventArgs 参数中返回 PingReply，

通过 Reply 属性访问 PingReply。

下面的代码示例演示如何异步使用 Ping 类。

```
public static void Main(string[] args)
{
    string who = "192.168.1.100";
    AutoResetEvent waiter = new AutoResetEvent(false);
    Ping pingSender = new Ping();
    // 设置 Ping 类的异步回调事件函数 PingCompletedCallback,
    pingSender.PingCompleted += new PingCompletedEventHandler(PingCompletedCallback);
    //填充 32 bytes 的传输数据.
    string data = "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa";
    byte[] buffer = Encoding.ASCII.GetBytes(data);
    // Wait 12 seconds for a reply.
    int timeout = 12000;
    //设置 Ping 命令属性，数据在销毁前可通过 64 个网关
    PingOptions options = new PingOptions(64, true);
    Console.WriteLine("Time to live: {0}", options.Ttl);
    Console.WriteLine("Don't fragment: {0}", options.DontFragment);
    //启动异步 Ping 命令
    pingSender.SendAsync(who, timeout, buffer, options, waiter);
    //等待 Ping 响应返回事件
    waiter.WaitOne();
    Console.WriteLine("Ping example completed.");
}
//Ping 类的回调事件函数
private static void PingCompletedCallback(object sender, PingCompletedEventArgs e)
{
    // If the operation was canceled, display a message to the user.
    if (e.Cancelled)
    {
        Console.WriteLine("Ping canceled.");
        ((AutoResetEvent)e.UserState).Set();
    }
    if (e.Error != null)
    {
        Console.WriteLine("Ping failed:");
        Console.WriteLine(e.Error.ToString());
        ((AutoResetEvent)e.UserState).Set();
    }
    PingReply reply = e.Reply;
    DisplayReply(reply);
    ((AutoResetEvent)e.UserState).Set();
}
public static void DisplayReply(PingReply reply)
{
    if (reply == null)
        return;
    Console.WriteLine("ping status: {0}", reply.Status);
    if (reply.Status == IPStatus.Success)
    {
        Console.WriteLine("Address: {0}", reply.Address.ToString());
        Console.WriteLine("RoundTrip time: {0}", reply.RoundtripTime);
        Console.WriteLine("Time to live: {0}", reply.Options.Ttl);
        Console.WriteLine("Don't fragment: {0}", reply.Options.DontFragment);
        Console.WriteLine("Buffer size: {0}", reply.Buffer.Length);
    }
}
```

13.2.3 构造 ICMP 包进行 Ping 功能

本小节构造 ICMP 数据实现 Ping 功能，演示原始套接字的使用方法。新建一个用于原始套接字的对象方法如下：

```
Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Raw, ProtocolType.Icmp);
```

创建一个控制台应用程序，类名为 RawPing，添加 ICMP 数据包类：

```
public class IcmpPacket
{
public Byte Type; // type of message
public Byte SubCode; // type of sub code
public UInt16 CheckSum; // ones complement checksum of struct
public UInt16 Identifier; // identifier
public UInt16 SequenceNumber; // sequence number
public Byte [] Data;
} // class IcmpPacket
```

主函数读入目标机地址，调用 Ping 命令。

```
//声明常量
const int SOCKET_ERROR = -1;
const int ICMP_ECHO = 8;
static void Main(string[] args)
{
RawPing p = new RawPing();
Console.WriteLine(" 请输入要 Ping 的 IP 或者主机名字：");
string MyUrl = Console.ReadLine();
Console.WriteLine(" 正在 Ping " + MyUrl + " ……");
Console.Write(p.PingHost(MyUrl));
Console.WriteLine();
Console.ReadLine();
}
```

用于计算数据包的较验和函数：

```
public static UInt16 checksum( UInt16[] buffer, int size )
{
Int32 cksum = 0;
int counter;
counter = 0;
while ( size > 0 )
{
  UInt16 val = buffer[counter];
  cksum += Convert.ToInt32( buffer[counter] );
  counter += 1;
  size -= 1;
}
cksum = (cksum >> 16) + (cksum & 0xffff);
cksum += (cksum >> 16);
return (UInt16)(~cksum);
}
```

将数据包头重新拼接，为计算较验和作准备：

```
public static Int32 Serialize(IcmpPacket packet, Byte[] Buffer,Int32 PacketSize, Int32 PingData
)
{
Int32 cbReturn = 0;
// serialize the struct into the array
int Index=0;
Byte [] b_type = new Byte[1];
b_type[0] = (packet.Type);
Byte [] b_code = new Byte[1];
b_code[0] = (packet.SubCode);
Byte [] b_cksum = BitConverter.GetBytes(packet.CheckSum);
Byte [] b_id = BitConverter.GetBytes(packet.Identifier);
Byte [] b_seq = BitConverter.GetBytes(packet.SequenceNumber);
Array.Copy( b_type, 0, Buffer, Index, b_type.Length );
Index += b_type.Length;
Array.Copy( b_code, 0, Buffer, Index, b_code.Length );
Index += b_code.Length;
Array.Copy( b_cksum, 0, Buffer, Index, b_cksum.Length );
Index += b_cksum.Length;
Array.Copy( b_id, 0, Buffer, Index, b_id.Length );
Index += b_id.Length;
Array.Copy( b_seq, 0, Buffer, Index, b_seq.Length );
Index += b_seq.Length;
// copy the data
Array.Copy( packet.Data, 0, Buffer, Index, PingData );
Index += PingData;
if( Index != PacketSize/* sizeof(IcmpPacket) */)
{
  cbReturn = -1;
  return cbReturn;
}
cbReturn = Index;
return cbReturn;
}
```

用于实现 Ping 功能的 PingHost 函数：

```
public string PingHost(string host)
{
// 声明 IPHostEntry
IPHostEntry serverHE, fromHE;
int nBytes = 0;
int dwStart = 0, dwStop = 0;
//初始化 ICMP 的 Socket
Socket socket = new Socket(AddressFamily.InterNetwork, SocketType.Raw, ProtocolType.Icmp);
socket.SetSocketOption(SocketOptionLevel.Socket, SocketOptionName.SendTimeout, 1000);
// 得到 Server EndPoint
try
{
  serverHE = Dns.GetHostEntry(host);
}
catch(Exception)
{
  return " 没有发现主机";
}
// 把 Server IP_EndPoint 转换成 EndPoint
IPEndPoint ipepServer = new IPEndPoint(serverHE.AddressList[0], 0);
```

```
EndPoint epServer = (ipepServer);
// 设定客户机的接收 Endpoint
fromHE = Dns.GetHostByName(Dns.GetHostName());
IPEndPoint ipEndPointFrom = new IPEndPoint(fromHE.AddressList[0], 0);
EndPoint EndPointFrom = (ipEndPointFrom);
int PacketSize = 0;
IcmpPacket packet = new IcmpPacket();
// 构建要发送的包
packet.Type = ICMP_ECHO; //8
packet.SubCode = 0;
packet.CheckSum = UInt16.Parse("0");
packet.Identifier = UInt16.Parse("45");
packet.SequenceNumber = UInt16.Parse("0");
int PingData = 32; // sizeof(IcmpPacket) - 8;
packet.Data = new Byte[PingData];
// 初始化 Packet.Data
string Tempstr = @"http://abc.12.12345678.net/xml/#";
for (int i = 0; i < PingData; i++)
{
  packet.Data[i] =(byte)Tempstr[i];
}
//Variable to hold the total Packet size
PacketSize = PingData + 8;
Byte [] icmp_pkt_buffer = new Byte[ PacketSize ];
Int32 Index = 0;
//计算数据包总字节数
Index = Serialize(
  packet,
  icmp_pkt_buffer,
  PacketSize,
  PingData );
//Error in Packet Size
if( Index == -1 )
{
  return "Error Creating Packet";
}
// convert into a UInt16 array
//Get the Half size of the Packet
Double double_length = Convert.ToDouble(Index);
Double dtemp = Math.Ceiling( double_length / 2);
```

```
int cksum_buffer_length = Convert.ToInt32(dtemp);
//Create a Byte Array
UInt16 [] cksum_buffer = new UInt16[cksum_buffer_length];
//Code to initialize the Uint16 array
int icmp_header_buffer_index = 0;
for( int i = 0; i < cksum_buffer_length; i++ )
{
  cksum_buffer[i] =BitConverter.ToUInt16(icmp_pkt_buffer,icmp_header_buffer_index);
  icmp_header_buffer_index += 2;
}
//Call a method which will return a checksum
UInt16 u_cksum = checksum(cksum_buffer, cksum_buffer_length);
//Save the checksum to the Packet
packet.CheckSum = u_cksum;
// Now that we have the checksum, serialize the packet again
Byte [] sendbuf = new Byte[ PacketSize ];
//again check the packet size
Index = Serialize(
  packet,
  sendbuf,
  PacketSize,
  PingData );
//if there is a error report it
if( Index == -1 )
{
  return "Error Creating Packet";
}
dwStart = System.Environment.TickCount; // Start timing
//send the Packet over the socket
if ((nBytes = socket.SendTo(sendbuf, PacketSize, 0, epServer)) == SOCKET_ERROR)
{
  return "Socket Error: cannot send Packet";
}
// Initialize the buffers. The receive buffer is the size of the
// ICMP header plus the IP header (20 bytes)
Byte [] ReceiveBuffer = new Byte[256];
nBytes = 0;
//Receive the bytes
bool recd =false ;
int timeout=0 ;
```

```
//loop for checking the time of the server responding
while(!recd)
{
  nBytes = socket.ReceiveFrom(ReceiveBuffer, 256, 0, ref EndPointFrom);
  if (nBytes == SOCKET_ERROR)
  {
  return " 主机没有响应" ;
  }
  else if(nBytes>0)
  {
  dwStop = System.Environment.TickCount - dwStart; // stop timing
  return "Reply from "+epServer.ToString()+" in "
    +dwStop+"ms. Received: "+nBytes+ " Bytes.";
  }
  timeout=System.Environment.TickCount - dwStart;
  if(timeout>1000)
  {
  return " 超时" ;
  }
}
//close the socket
socket.Close();
return "";
}
```

13.3 作业与思考

1. 完成本实验中的程序。
2. 思考采用 RawSock 实现 UDP 数据包伪装。