Федеральное агентство по образованию.

Тамбовский государственный технический университет.

Кафедра:

Отчетпо лабораторной работе №3

по дисциплине: Сети и телекоммуникации

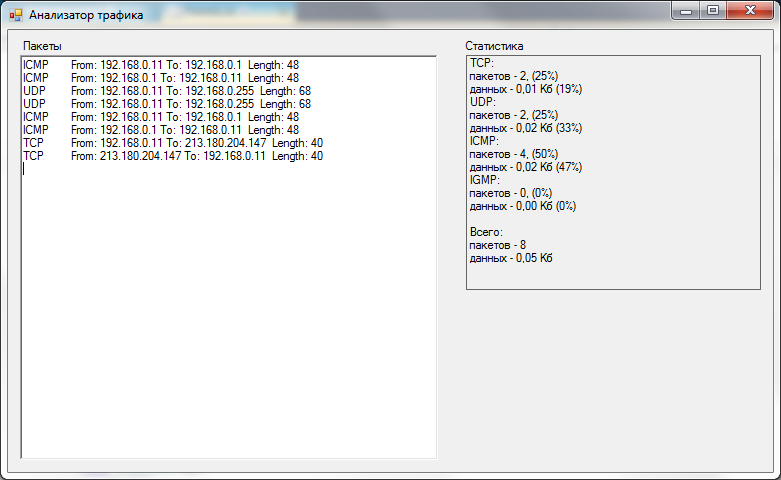
Выполнил: студент гр. -31

Проверил: В.И.

Тамбов **Постановка задачи**

Используя библиотеку winsock32.dll, реализовать анализ трафика компьютера.

# Результаты работы программы



# Код программы

## Класс формы

public partial class Form1 : Form

    {

        public Form1()

        {

            InitializeComponent();

        }

        private Socket socket; //собственно это и есть наше прослушивающее устройство

        private byte[] buffer; //а сюда мы будем записывать полученные пакеты

        //Создаем и инициализируем низкоуровневый сокет

        private void Form1\_Load(object sender, EventArgs e)

        {

            //IP-адреса этой машины (нужен реальный локальный адрес)

            IPHostEntry ipHostEntry = Dns.GetHostByName(Dns.GetHostName());

            IPAddress ip = ipHostEntry.AddressList[0];

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

            //socket.Bind(new IPEndPoint(IPAddress.Parse("192.168.0.11"), 0));

            socket.Bind(new IPEndPoint(ip, 0));

            socket.SetSocketOption(SocketOptionLevel.IP, SocketOptionName.HeaderIncluded, true);

            byte[] byInc = new byte[] { 1, 0, 0, 0 };

            byte[] byOut = new byte[4];

            buffer = new byte[4096];

            socket.IOControl(IOControlCode.ReceiveAll, byInc, byOut);

            //Начинаем прослушивание

            socket.BeginReceive(buffer, 0, buffer.Length, SocketFlags.None, OnReceive, null);

        }

        //Пришел пакет

        private void OnReceive(IAsyncResult ar)

        {

            try

            {

                int nReceived = socket.EndReceive(ar);

                Print(buffer, nReceived);

                buffer = new byte[4096];

                socket.BeginReceive(buffer, 0, buffer.Length, SocketFlags.None,

                    OnReceive, null);

            }

            catch { }

        }

        //Счетчики для статистики

        int packetsTCP = 0, packetsUDP = 0, packetsICMP = 0, packetsIGMP = 0, allPackets = 0;

        long dataTCP = 0, dataUDP = 0, dataICMP = 0, dataIGMP = 0, allData;

        private void Print(byte[] buf, int len)

        {

            //Парсим пакет

            IPHeader ipHeader = new IPHeader(buf, len);

            //Пишем в лог информацию о пакете

            string info =

                string.Format("{0}\tFrom: {1} To: {2}  Length: {3}\n",

                ipHeader.ProtocolType.ToString(),

                ipHeader.SourceAddress,

                ipHeader.DestinationAddress,

                ipHeader.TotalLength);

            richTextBox1.BeginInvoke(new Action(() => richTextBox1.AppendText(info)));

            //Считаем статистику

            allPackets++;

            allData += ipHeader.TotalLength;

            switch (ipHeader.ProtocolType)

            {

                case Protocol.TCP:

                    {

                        packetsTCP++;

                        dataTCP += ipHeader.TotalLength;

                        break;

                    }

                case Protocol.UDP:

                    {

                        packetsUDP++;

                        dataUDP += ipHeader.TotalLength;

                        break;

                    }

                case Protocol.ICMP:

                    {

                        packetsICMP++;

                        dataICMP += ipHeader.TotalLength;

                        break;

                    }

                case Protocol.IGMP:

                    {

                        packetsIGMP++;

                        dataIGMP += ipHeader.TotalLength;

                        break;

                    }

            }

            //Выводим статистику

            string stat = string.Format(

                "TCP: \nпакетов - {0}, ({1}%) \nданных - {2:0.00} Кб ({3}%)\n" +

                "UDP: \nпакетов - {4}, ({5}%) \nданных - {6:0.00} Кб ({7}%)\n" +

                "ICMP: \nпакетов - {8}, ({9}%) \nданных - {10:0.00} Кб ({11}%)\n" +

                "IGMP: \nпакетов - {12}, ({13}%) \nданных - {14:0.00} Кб ({15}%)\n" +

                "\nВсего: \nпакетов - {16} \nданных - {17:0.00} Кб",

                packetsTCP, (int)100 \* packetsTCP / allPackets, (double)dataTCP / (8 \* 1024), (int)100 \* dataTCP / allData,

                packetsUDP, (int)100 \* packetsUDP / allPackets, (double)dataUDP / (8 \* 1024), (int)100 \* dataUDP / allData,

                packetsICMP, (int)100 \* packetsICMP / allPackets, (double)dataICMP / (8 \* 1024), (int)100 \* dataICMP / allData,

                packetsIGMP, (int)100 \* packetsIGMP / allPackets, (double)dataIGMP / (8 \* 1024), (int)100 \* dataIGMP / allData,

                allPackets, (double)allData / (8 \* 1024));

            lblStat.BeginInvoke(new Action(() => lblStat.Text = stat));

        }

    }

## Класс IPHeader

public class IPHeader

    {

        //IP Header fields

        private byte byVersionAndHeaderLength;   //Eight bits for version and header length

        private byte byDifferentiatedServices;    //Eight bits for differentiated services (TOS)

        private ushort usTotalLength;              //Sixteen bits for total length of the datagram (header + message)

        private ushort usIdentification;           //Sixteen bits for identification

        private ushort usFlagsAndOffset;           //Eight bits for flags and fragmentation offset

        private byte byTTL;                      //Eight bits for TTL (Time To Live)

        private byte byProtocol;                 //Eight bits for the underlying protocol

        private short sChecksum;                  //Sixteen bits containing the checksum of the header

        //(checksum can be negative so taken as short)

        private uint uiSourceIPAddress;          //Thirty two bit source IP Address

        private uint uiDestinationIPAddress;     //Thirty two bit destination IP Address

        //End IP Header fields

        private byte byHeaderLength;             //Header length

        private byte[] byIPData = new byte[4096];  //Data carried by the datagram

        public IPHeader(byte[] byBuffer, int nReceived)

        {

            try

            {

                //Create MemoryStream out of the received bytes

                MemoryStream memoryStream = new MemoryStream(byBuffer, 0, nReceived);

                //Next we create a BinaryReader out of the MemoryStream

                BinaryReader binaryReader = new BinaryReader(memoryStream);

                //The first eight bits of the IP header contain the version and

                //header length so we read them

                byVersionAndHeaderLength = binaryReader.ReadByte();

                //The next eight bits contain the Differentiated services

                byDifferentiatedServices = binaryReader.ReadByte();

                //Next eight bits hold the total length of the datagram

                usTotalLength = (ushort)IPAddress.NetworkToHostOrder(binaryReader.ReadInt16());

                //Next sixteen have the identification bytes

                usIdentification = (ushort)IPAddress.NetworkToHostOrder(binaryReader.ReadInt16());

                //Next sixteen bits contain the flags and fragmentation offset

                usFlagsAndOffset = (ushort)IPAddress.NetworkToHostOrder(binaryReader.ReadInt16());

                //Next eight bits have the TTL value

                byTTL = binaryReader.ReadByte();

                //Next eight represnts the protocol encapsulated in the datagram

                byProtocol = binaryReader.ReadByte();

                //Next sixteen bits contain the checksum of the header

                sChecksum = IPAddress.NetworkToHostOrder(binaryReader.ReadInt16());

                //Next thirty two bits have the source IP address

                uiSourceIPAddress = (uint)(binaryReader.ReadInt32());

                //Next thirty two hold the destination IP address

                uiDestinationIPAddress = (uint)(binaryReader.ReadInt32());

                //Now we calculate the header length

                byHeaderLength = byVersionAndHeaderLength;

                //The last four bits of the version and header length field contain the

                //header length, we perform some simple binary airthmatic operations to

                //extract them

                byHeaderLength <<= 4;

                byHeaderLength >>= 4;

                //Multiply by four to get the exact header length

                byHeaderLength \*= 4;

                //Copy the data carried by the data gram into another array so that

                //according to the protocol being carried in the IP datagram

                Array.Copy(byBuffer,

                           byHeaderLength,  //start copying from the end of the header

                           byIPData, 0,

                           usTotalLength - byHeaderLength);

            }

            catch (Exception ex)

            {

                MessageBox.Show(ex.Message, "MJsniffer", MessageBoxButtons.OK,

                    MessageBoxIcon.Error);

            }

        }

        public string Version

        {

            get

            {

                //Calculate the IP version

                //The four bits of the IP header contain the IP version

                if ((byVersionAndHeaderLength >> 4) == 4)

                {

                    return "IP v4";

                }

                else if ((byVersionAndHeaderLength >> 4) == 6)

                {

                    return "IP v6";

                }

                else

                {

                    return "Unknown";

                }

            }

        }

        public string HeaderLength

        {

            get

            {

                return byHeaderLength.ToString();

            }

        }

        public ushort MessageLength

        {

            get

            {

                //MessageLength = Total length of the datagram - Header length

                return (ushort)(usTotalLength - byHeaderLength);

            }

        }

        public string DifferentiatedServices

        {

            get

            {

                //Returns the differentiated services in hexadecimal format

                return string.Format("0x{0:x2} ({1})", byDifferentiatedServices,

                    byDifferentiatedServices);

            }

        }

        public string Flags

        {

            get

            {

                //The first three bits of the flags and fragmentation field

                //represent the flags (which indicate whether the data is

                //fragmented or not)

                int nFlags = usFlagsAndOffset >> 13;

                if (nFlags == 2)

                {

                    return "Don't fragment";

                }

                else if (nFlags == 1)

                {

                    return "More fragments to come";

                }

                else

                {

                    return nFlags.ToString();

                }

            }

        }

        public string FragmentationOffset

        {

            get

            {

                //The last thirteen bits of the flags and fragmentation field

                //contain the fragmentation offset

                int nOffset = usFlagsAndOffset << 3;

                nOffset >>= 3;

                return nOffset.ToString();

            }

        }

        public string TTL

        {

            get

            {

                return byTTL.ToString();

            }

        }

        public Protocol ProtocolType

        {

            get

            {

                //The protocol field represents the protocol in the data portion

                //of the datagram

                switch (byProtocol)        //A value of six represents the TCP protocol

                {

                    case 6: return Protocol.TCP;

                    case 17: return Protocol.UDP;

                    case 1: return Protocol.ICMP;

                    case 2: return Protocol.IGMP;

                }

                return Protocol.Unknown;

            }

        }

        public string Checksum

        {

            get

            {

                //Returns the checksum in hexadecimal format

                return string.Format("0x{0:x2}", sChecksum);

            }

        }

        public IPAddress SourceAddress

        {

            get

            {

                return new IPAddress(uiSourceIPAddress);

            }

        }

        public IPAddress DestinationAddress

        {

            get

            {

                return new IPAddress(uiDestinationIPAddress);

            }

        }

        public ushort TotalLength

        {

            get

            {

                return usTotalLength;

            }

        }

        public string Identification

        {

            get

            {

                return usIdentification.ToString();

            }

        }

        public byte[] Data

        {

            get

            {

                return byIPData;

            }

        }

    }