实 验 九　HTTP 协议的断点续传应用

9.1　实验目的

深入理解 HTTP 协议，结合协议分析断点续传工作原理与实现，基于 Web 服务器软件 Apache，开发客户端程序实现多线程下载。要求掌握带参数线程的编写与启动，程序中事件数组的同步控制方法。

9.2　断点续传与多线程下载

本小节实现 HTTP 协议的断点续传和多线程下载功能。为了模拟实际的网络应用，客户机与服务器都运行在 Windows XP 系统，并且分别位于不同的机器上，实验中 Apache 软件服务器地址为 192.168.1.100，客户机地址为 192.168.1.102，读者应根据自己机器环境进行相应设置。服务器要提供 Web 服务，需设置防火墙例外条件，方法是开始 | 控制面版 | 安全中心 | Windows 防火墙，在例外标签页 | 添加程序，添加程序 c:\Apache2\bin\httpd.exe，图9-1显示防火墙成功添加了 httpd 例外的载图。如果从客户机浏览器能够访问服务器的网页，则说明设置成功。

HTTP 协议不但能够传输 HTML 网页文本，还可以传其它多种格式文件，如 JPG、MP3 和 RAR 等类型文件，其它格式的文件的尺寸往往较大，网络环境容易发生网络连接中断的情况，为应对复杂的网络条件，HTTP 协议支持对大文件的断点续传和多线程下载功能。HTTP 协议有专门的报头域对要传输的文件进行分段标识，Range 实体域用在请求头中，指定第一个字节和最后一个位置：如 Range:200-300，相应的服务端的响应头中有 Content-Range 实体域，它除了响应发来的数据范围，还给出文件的总大小。如 Content-Range: bytes 200-300/2350，其中 2350 指出了文件总大小。

对于客户采用 GET 或者 POST 发出的请求，服务器在响应报文中将协议头部分与实际数据拼在一起发送，这给多线程下载中根据响应的内容来确定线程究竟从何处下载分段的文件数据带了一定的麻烦。例如需要先获得文件的总长度再确定线程要下载的位置，这可以采用 HEAD 请求，服务器收到 HEAD 请求后，与 GET 或者 POST 方式在响应头部分的内容是一样的，区别是服务器不提供数据部分。例如图9-2显示向服务器发送如下的请求包：

```
HEAD /httpd.rar HTTP/1.1
Connection: close
Range: bytes=100-1000
Host: 192.168.1.100
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Trident/5.0)
```

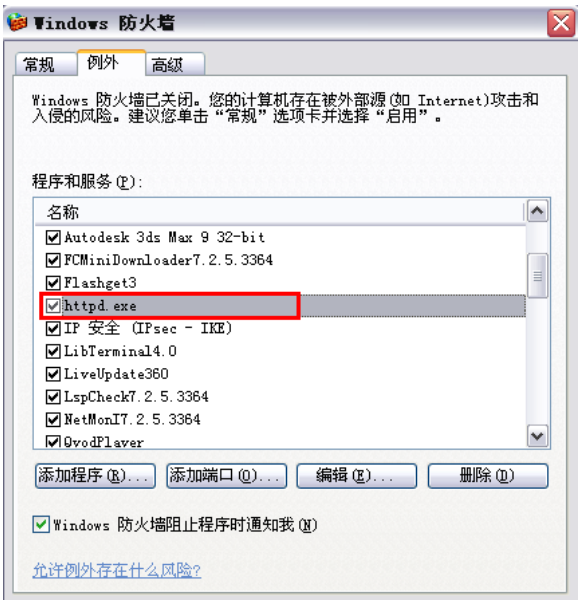

图 9-1 设置 Windows XP 网络防火墙

图 9-2 发送 HEAD 请求包与响应

服务器返回的响应消息如下：

HTTP/1.1 206 Partial Content

Date: Mon, 07 Jan 2008 14:52:48 GMT

Server: Apache/2.2.22 (Win32)

Last-Modified: Mon, 07 Jan 2008 09:08:33 GMT

ETag: "24000000025534-4bfa49-4431e345ab0e2"

Accept-Ranges: bytes

Content-Length: 901

Content-Range: bytes 100-1000/4979273

Connection: close

Content-Type: application/x-rar-compressed

服务端返回的 Content-Range 域指出了文件的大小为 4979273 字节，在这里采用两个线程实例分别下载文件的前半部分与后半部分，下载完成后，将获得的数据合并成一个文件，这就是多线程下载/断点续传的具体实现。编写线程代码函数 MultiThreadsDown，两次启动这个线程，通过每次传入不同的参数指定下载文件的不同部分。

在获取网页的线程里，修改原来的的 GET 请求命令，换成本小节的 HEAD 请求报文，解析收到的返回报文获取要下载的文件大小，参考代码如下：

```
//获取文件长度信息
long dFileLen = 0;
string strFilelen;
string[] responLines = totalrespon.Split("\r\n".ToCharArray(), StringSplitOptions.RemoveEmptyEntries);
for (int i = 0; i < responLines.Length;i++ )
{
  if (responLines[i].IndexOf("Content-Range")>-1)
```

```
    {
      strFilelen = responLines[i].Substring(responLines[i].IndexOf("/") + 1);
      dFileLen=Int64.Parse(strFilelen);
      break;
    }
  }
```

要启用相同的线程代码两次，生成两个线程实例，需要使用不同的线程参数下载文件的不同部分，要使用的内存对象与线程参数类 ThreadPara 参考定义如下：

```
  //用于同步下载线程
  static ManualResetEvent[] threadsDone = new ManualResetEvent[2];
  //线程使用的内存区，暂存不同的数据部分
  public static MemoryStream msFileData0;
  public static MemoryStream msFileData1;
  //线程参数类，用于标识文件下载位置，使用的内存区
  public class ThreadPara
  {
    public long StartPos;
    public long EndPos;
    public ManualResetEvent eventThreadDone;
    public MemoryStream msFData;
    public long dataLen;
  }
```

文件多线程下载函数 MultiThreadsDown 的定义如下，注意线程在定义时使用了参数，在启动线程时将根据不同参数值下载文件的不同位置：

```
  static void MultiThreadsDown(Object tdata)
  {
    try
    {
      // Establish the remote endpoint for the socket.
      IPAddress ipAddress = Dns.GetHostEntry(url_str).AddressList[0];
      IPEndPoint remoteEP = new IPEndPoint(ipAddress, port);
      // Create a TCP/IP socket.
      Socket client = new Socket(AddressFamily.InterNetwork,
        SocketType.Stream, ProtocolType.Tcp);
      // Connect to the remote endpoint.
      client.Connect(remoteEP);
      ThreadPara thdata = (ThreadPara)tdata;
      string httprequest;
      httprequest = "GET /" + httpfile + " HTTP/1.1\r\n";
      httprequest += "Connection: close\r\n";
```

```
        httprequest += "Range: bytes=" + thdata.StartPos.ToString() + "-" + thdata.EndPos.ToString()
+ "\r\n";
        httprequest += "Host: 192.168.1.100\r\n";
        httprequest += "User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1; Tri-
dent/5.0)\r\n\r\n";
        thdata.dataLen = thdata.EndPos - thdata.StartPos+1;
        thdata.msFData.Seek(0, SeekOrigin.Begin);
        byte[] sendData = Encoding.ASCII.GetBytes(httprequest);
        client.Send(sendData, SocketFlags.None);
        byte[] receiveData = new byte[1024];
        Int32 recvLen = client.Receive(receiveData, 1024, SocketFlags.None);
        thdata.msFData.Write(receiveData, 0, recvLen);
        while (recvLen > 0)
        {
          recvLen = client.Receive(receiveData, 1024, SocketFlags.None);
          thdata.msFData.Write(receiveData, 0, recvLen);
        }
        client.Shutdown(SocketShutdown.Both);
        client.Close();
        thdata.eventThreadDone.Set();
      }
      catch (Exception e)
      {
        MessageBox.Show(e.ToString());
      }
    }
```

在本实验中根据文件总长度以及 HTTP 断点续传报头定义生成两个参数，使用两个参数值启动线程两次，参考代码如下：

```
  long partpos = dFileLen/2;
  msFileData0 = new MemoryStream(5000000);
  msFileData1 = new MemoryStream(5000000);
  threadsDone[0] = new ManualResetEvent(false);
  threadsDone[1] = new ManualResetEvent(false);
  //初始化线程参数
  ThreadPara thrPara0 = new ThreadPara();
  thrPara0.msFData = msFileData0;
  thrPara0.eventThreadDone = threadsDone[0];
  thrPara0.StartPos = 0;
  thrPara0.EndPos = partpos;
  ThreadPara thrPara1 = new ThreadPara();
```

```
thrPara1.msFData = msFileData1;
thrPara1.eventThreadDone = threadsDone[1];
thrPara1.StartPos = partpos+1;
thrPara1.EndPos = dFileLen-1;
//启动线程下载文件不同部分
Thread Thread0 = new Thread(new ParameterizedThreadStart(MultiThreadsDown));
Thread0.Start(thrPara0);
Thread Thread1 = new Thread(new ParameterizedThreadStart(MultiThreadsDown));
Thread1.Start(thrPara1);
//同步两个下载线程的结束
WaitHandle.WaitAll(threadsDone);
```

下载线程收到的服务器响应报文是报文头部分 + 文件数据部分，需要界定出报文头部分，报文头部分在结束时使用连续的 \r\n\r\n 与数据区别，其字节值为 13、10、13、10，合并生成文件时只复制数据部分，参考代码如下：

```
//合并文件内容
byte[] temBuf=new byte[1024];
string filename = "d:\\" + httpfile ;
FileStream fs = new FileStream(filename, FileMode.CreateNew);
ThreadPara[] thParas = new ThreadPara[2];
thParas[0] = thrPara0;
thParas[1] = thrPara1;
for (int i = 0; i < 2;i++ )
{
  thParas[i].msFData.Seek(0, SeekOrigin.Begin);
  byte[] tData = new byte[4];
  int j = 0;
  //确定报头后的数据位置
  for (; j < 1000; j++)
  {
    thParas[i].msFData.Seek(j, SeekOrigin.Begin);
    thParas[i].msFData.Read(tData, 0, 4);
    if ((tData[0] == 13) && (tData[1] == 10) && (tData[2] == 13) && (tData[3] == 10))
    {
      thParas[i].msFData.Seek(j + 4, SeekOrigin.Begin);
      break;
    }
  }
  long totalLen = 0;
  int writeLen = 0;
  do
```

```
    {
        writeLen = thParas[i].msFData.Read(temBuf, 0, 1024);
        fs.Write(temBuf, 0, writeLen);
        totalLen += writeLen;
    }
    while (totalLen < thParas[i].dataLen);
}
fs.Flush();
fs.Close();
fs.Dispose();
```

适当修改获取网页的线程 thread_GET_html，启用两个线程下载文件数据。实际的网络平台会有一定变化，需要用户根据实际情况调整程序以适应。

9.3 WireShark 抓包软件

网络数据在局域网内以网络广播方式传播，可通过软件方式获取同网段内数据包，比较出名的软件有 sniffer 和 WireShark，本实验内容基于 WireShark 软件进行网络协议分析。

WireShark 软件曾叫作"Ethereal"，它是一个网络数据分析工具，可以在线捕获网络数据包，网络管理人员使用它监视网络或者进行故障排错。WireShark 软件还是用于研究学习多种网络协议的优秀工具，抓取已有软件的网络包通过模拟的方法能迅速掌握协议应用。WireShark 软件包含一个名为 WinpCap 的软件包，WinpCap 完成抓包的任务。在 Windows XP 平台安装 WireShark 软件的过程非常简单，如果在 Win8 平台安装这个软件包时需设置为兼容的运行模式，并且要以管理员身份运行，右键点击软件后选择属性菜单项进行设置，操作如图9-3所示，忽略安装过程的兼容性提示如图9-4所示。

Wireshark 软件成功安装后的初始运行界面如图9-5。

在 WireShark 软件初始运行界面，列举了机器上可用的网卡列表 (Interface List)，图9-5表示了 Brodcom 的以太网卡和 Intel 的 4965AGN 无线网卡。点击需要进行抓包的网卡，WireShark 马上进行抓包过程，机器在运行时一般都会有大量的实时网络数据包，抓包时间不宜过长，要及时停止抓包过程，防止产生的抓包文件过大。例如用户希望掌握实际的 HTTP 协议的请求包和服务器的响应数据包，操作方法如下：

1. 启动 WireShark 抓包功能；
2. 让浏览器向服务器发送请求，例如输入地址 http://192.168.1.100/ee.rar；
3. 得到服务器请求后，停止 WireShark 的抓包功能；
4. 在过滤条件栏中输入 http，只查看 http 协议数据，找到目标地址为 192.168.1.100 的记录行；
5. 观察和学习 http 协议的请求数据包和响应数据包；

图9-6演示了从地址 http://192.168.1.100/ee.rar 下载文件的网络抓包载图。

9.4 实验作业

1. 调试程序实现多线程下载文件任务。

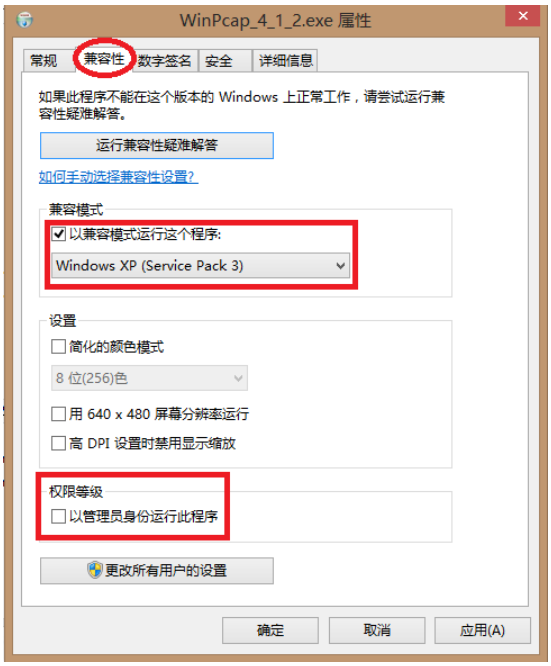

图 9-3 设置 Wireshark 的兼容安装方式

图 9-4 忽略软件兼容性提示

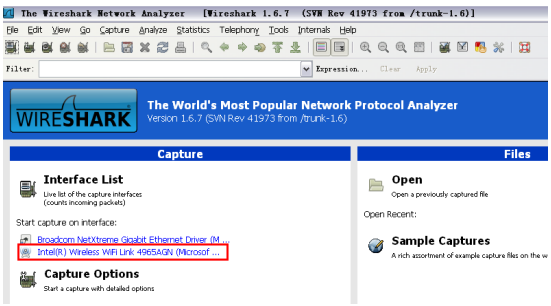

图 9-5 WireShark 软件初始运行界面

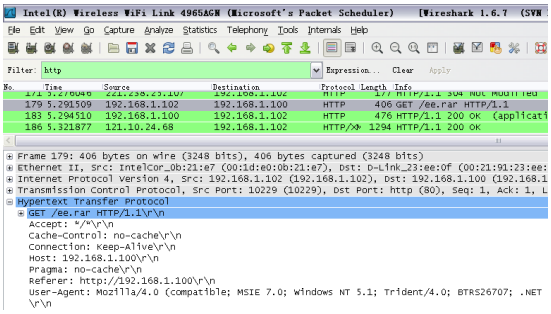

图 9-6 WireShark 抓取网络数据包界面