实 验 三 UDP 文件传输

3.1 实验目的

本实验项目自定义协议包采用 UDP 协议实现对任意文件的发送和接收，项目包括两个窗体程序程序分别用于文件发送与接收。项目中多线程中事件同步控制与 Socket 异步接收方式结合是主要难点。

3.2 UDP 文件传输原理

Socket 类支持 UDP 数据传输模式，它使用 SendTo 方法发送数据 ReceiveFrom 方法接收数据，应用程序通常采用线程循环方式发送与接收数据，图3-1演示了 UDP 数据发送接收接口与线程的关系。

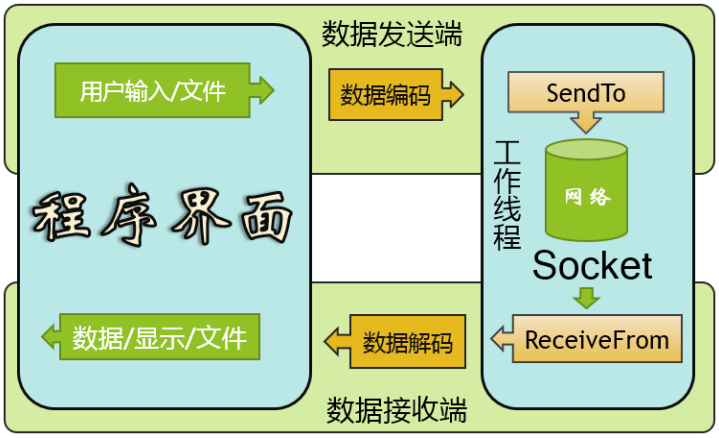

图 3-1 UDP 数据发送接收接口与线程关系

3.2.1 UDP 文件传输功能分析

本项目包含两个窗体程序，分别为文件发送端与文件接收端，发送端选择文件后线程自动完成数据的传输，接收端自动完成文件接收，其运行的效果可参考图3-2与图3-3。

使用 Socket 的 UDP 方式进行网络通信的过程包括：端口绑定、执行 SendTo 和 ReceiveFrom 方法实现数据的发送和接收。UDP 通信没有连接或者断开操作，UDP 通信不存在连接状态按制，它会发生数据丢包和乱序的现象。本项目中为了发送有序的文件数据，设计了发送和应答模式下的简单传输协议，以 1000 字节为单位对文件划分数据块，接收端对每块数据回复对应的

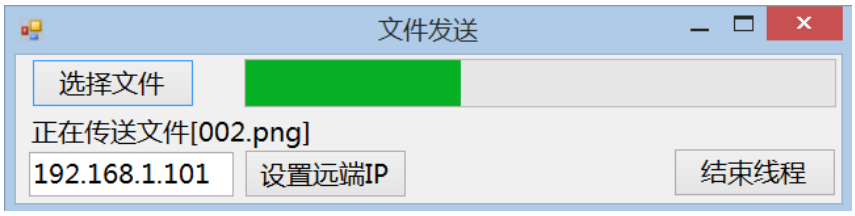

图 3-2 UDP 文件发送端

图 3-3 UDP 文件接收端

块编号。程序使用的是异步的 UDP 接收，使用了 Socket 的 BeginReceiveFrom 方法，这个方法需要由函数 UdpReceiveCallBack 配合完成回调，这增加了编程的难度换来的是程序的灵活性。

窗体接收用户输入及显示文件传输进度，发送端工作线程从文件读数据并封装为指定数据包，通过 UDP 数据包发送，并具有简单的超时重传机制。接收端工作线程采用 Socket 的异步方式接收数据并写入文件，接收端对每次成功的数据包都回复其块编号。

3.2.2 UDP 文件传输协议设计

结合 UDP 协议特点与本项目需求，协议能够控制文件的有序和完整，为方便数据解析数据包的格式字段统一规整。网络传输的内容包括文件名和块数以及文件数据两部分，传输的结果为成功或失败（网络中断）。Socket 组件的 UDP 接口提供的是字节数组传输的接口，文件对象并不等同于字节数组，通过网络传输文件时，文件对象转化为 UDP 传输的字节数组过程如图3-4。文件的长度和文件名经过序列化转化为字节序列与文件数据一起共同作为有效载荷，有效载荷还需添加为保证数据完整和有序的控制头。

满足本项目要求的基于 UDP 协议传输文件的控制协议格式由图3-5所示，它指明了字节序列的对应意义，并且容易实现。

协议包括发送端发送请求和数据两种数据包，接收端只对编号进行回复，文件数据块的编号由 1 开始。

3.3 文件发送端程序实现

发送端窗体界面任务包括设置目标机器 IP 地址，用户选择文件，显示文件传输进度和线程运行信息。工作线程向窗体线程发送消息须用到 SendMessage 内核函数，窗体线程则通过 ManualResetEvent 对象与窗体线程进行同步控制。窗体线程的程序简单，工作线程的任务比较复杂。

首先是程序中的各种变量的声明：

```
[DllImport("User32.dll")]
```

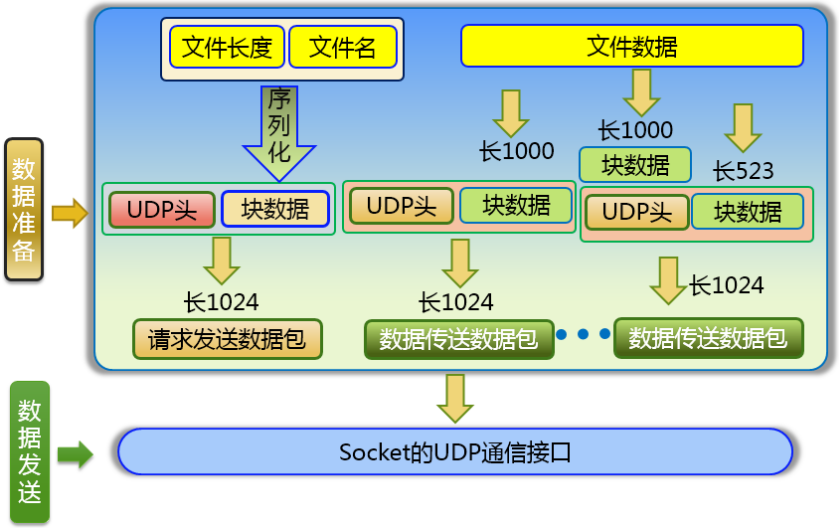

图 3-4 文件对象转化为 UDP 传输字节数组过程

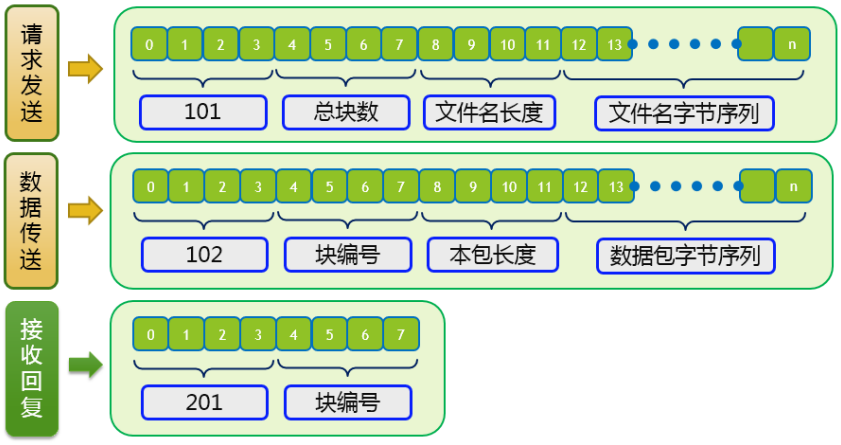

图 3-5 UDP 文件传送协议格式与意义

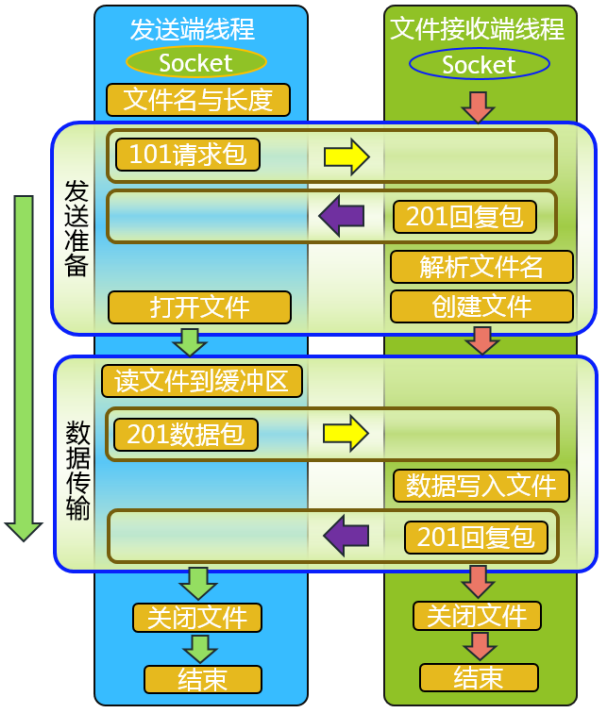

图 3-6 UDP 文件传送流程

```
private static extern int SendMessage(
IntPtr hWnd,
int Msg,
int wParam,
int lParam
);
public const int FILETRAN_UPPROG = 0x502;
public const int FILETRAN_UPTEXT = 0x503;
public const int MAX_REPEAT_TIMES = 3;
public static string strFileFullName;//要传输的文件名称
public static string strMsg;//要显示的线程信息
public static bool fileWaitToSend = false;
public static ManualResetEvent meEndThr; //结束线程 0
public static ManualResetEvent mePkgCome;//数据到达 1
public static ManualResetEvent meStartSend;//开始发送 2
public static ManualResetEvent[] meAry;
//UDP 数据发送端口与接收端口值 -文件发送端
public const int UDPRECV_PORT = 9095;
public const int UDPSEND_PORT = 9096;
public static Socket skUdpSend;
public static Socket skUdpRecv;
```

```
public static EndPoint remoteEp;
public static IPEndPoint remoteIPEp;
public static IPAddress remoteIPAddr;
public static int iUdpRecvPkgLen;
public static IntPtr wndHandle; //保存窗体句柄
public static byte[] udpRecvDataBuf;//接收数据的缓冲区
public static byte[] udpSendDataBuf;//发送数据的缓冲区
public static int iRepeatTry = 0;
public static int iCurPkgNum = 0;//当前正发送的数据块编号
public static int iRlyPkgNum = 0;//回复的数据块编号
public static int iTotalPkgCount = 0;//数据块总数
public static int iReadFilePkgNum = 0;//数据块总数
public static int iRpyPkgNum = 0;//接收到的确定块编号
public static FileStream fsCurFile;//当前要传输的文件流对象
public static long lCurFilePos;//当前文件流读写位置
public static bool bThrIsIdle = true;//线程空闲状态指示
public static int iUdpBufLen = 0;//要发送的 UDP 缓冲区长度
public static bool ipIsOK = false;
public static FileInfo fiTran;
public static Byte[] tmpByte;//数据序列化的字节临时数组
public static Byte[] tmpFN;//文件名序列化的字节临时数组
```

重载窗体消息处理函数来响应自定义消息：

```
protected override void DefWndProc(ref Message m)
{//窗体消息处理重载
  switch (m.Msg)
  {
    case FILETRAN_UPPROG://更新进度条值
      progressBar1.Value = (int)m.LParam;
      break;
    case FILETRAN_UPTEXT://更新文本信息
      label1.Text = strMsg;
      break;
    default:
      base.DefWndProc(ref m);
      break;
  }
}
```

InitData 函数对数据初始化和启动工作线程，并绑定 UDP 数据接收端口。

```
public void InitData()
{
  meAry = new ManualResetEvent[3];
  meEndThr = new ManualResetEvent(false);//0
  mePkgCome = new ManualResetEvent(false);//1
  meStartSend = new ManualResetEvent(false);//2
  meAry[0] = meEndThr;
  meAry[1] = mePkgCome;
  meAry[2] = meStartSend;
  //绑定接收 UDP 数据回调函数
  udpRecvDataBuf = new byte[1024];
  remoteIPEp = new IPEndPoint(IPAddress.Any, 0);
  skUdpRecv = new Socket(AddressFamily.InterNetwork,
    SocketType.Dgram, ProtocolType.Udp);
  IPEndPoint iep = new IPEndPoint(IPAddress.Any, UDPRECV_PORT);
  skUdpRecv.Bind(iep);
  remoteEp = (EndPoint)remoteIPEp;
  skUdpRecv.BeginReceiveFrom(udpRecvDataBuf, 0, 1024,
    SocketFlags.None, ref remoteEp, UdpReceiveCallBack, new object());
  //创建发送数据 Socket 对象与数据缓冲区
  udpSendDataBuf = new byte[1024];
  skUdpSend = new Socket(AddressFamily.InterNetwork,
    SocketType.Dgram, ProtocolType.Udp);
  ThreadStart thrStart = new ThreadStart(FileSendThr);
  Thread thrT = new Thread(thrStart);
  thrT.IsBackground = true;
  thrT.Start();
}
```

窗体的 Load 事件如下：

```
private void FrmMain_Load(object sender, EventArgs e)
{
  wndHandle = this.Handle;
  InitData();
}
```

UDP 数据接收采用异步方式最能适应网络情况，也增加了编程复杂性，通过设置事件对象状态有效通知工作线程处理接收到的数据，下面是发送文件端的 UDP 回调函数：

```
public static void UdpReceiveCallBack(IAsyncResult ar)
{
  try
  {
```

```
        EndPoint tmpRemoteEp = (EndPoint)remoteIPEp;
        iUdpRecvPkgLen = skUdpRecv.EndReceiveFrom(ar, ref tmpRemoteEp);
        mePkgCome.Set();
        skUdpRecv.BeginReceiveFrom(udpRecvDataBuf, 0, 1024,
        SocketFlags.None, ref remoteEp, UdpReceiveCallBack, new object());
      }
      catch (SocketException se)
      {
        MessageBox.Show(se.Message);
      }
    }
```

设置目标 IP 地址的按钮事件函数：

```
    private void button2_Click(object sender, EventArgs e)
    {
      ipIsOK = IPAddress.TryParse(textBox1.Text, out remoteIPAddr);
      if (!ipIsOK)
      {
        label1.Text = " 输入 IP 信息不正确。";
        textBox1.Focus();
      }
      else
      {
        label1.Text = "IP 地址解析 OK。";
      }
    }
```

用户选择要传送的文件后，首先检查 IP 地址是否输入正确，然后获取文件基本信息并计算文件的块数，基本信息序列化到数据发送缓冲区，设置事件状态有效通知线程发送文件请求信息：

```
    private void button1_Click(object sender, EventArgs e)
    {
      if (ipIsOK)
      {
        if (openFileDialog1.ShowDialog() == DialogResult.OK)
        {
          strFileFullName = openFileDialog1.FileName;
          //fileWaitToSend = true;//此变量是否有用？
          //打开文件流，设置文件读写位置为 0
          fsCurFile = new FileStream(strFileFullName, FileMode.Open, FileAccess.Read);
          lCurFilePos = 0L;
          //设置远端 IP 地址与端口
```

```
    remoteIPEp.Address = remoteIPAddr;
    remoteIPEp.Port = UDPSEND_PORT;
    //计算数据包总数
    fiTran = new FileInfo(strFileFullName);
    if ((fiTran.Length % 1000L) == 0)
    {
      iTotalPkgCount = (int)(fiTran.Length / 1000L);
    }
    else
    {
      iTotalPkgCount = (int)(fiTran.Length / 1000L + 1);
    }
    //设置进度条值，以文件块数为显示值。
    progressBar1.Minimum = 0;
    progressBar1.Maximum = iTotalPkgCount;
    strMsg = string.Format(" 正在传送文件 [{0}]",fiTran.Name);
    SendMessage(wndHandle, FILETRAN_UPTEXT, 100, 100);
    //文件读写块为 0
    iReadFilePkgNum = 0;
    iRepeatTry = 0;//重发次数重置为 0
    //数据缓冲区清零
    Array.Clear(udpSendDataBuf, 0, 1024);
    //[Int32]101
    tmpByte = BitConverter.GetBytes((Int32)101);
    Array.Copy(tmpByte, 0, udpSendDataBuf, 0, 4);
    //[int32] 总块数 (4 字节)
    tmpByte = BitConverter.GetBytes(iTotalPkgCount);
    Array.Copy(tmpByte, 0, udpSendDataBuf, 4, 4);
    //不包含路径的文件名字符串
    tmpFN = Encoding.UTF8.GetBytes(fiTran.Name);
    tmpByte = BitConverter.GetBytes(tmpFN.Length);
    //[int32] 文件名字节长度 (4 字节)
    Array.Copy(tmpByte, 0, udpSendDataBuf, 4 + 4, 4);
    //[int32] 文件名长度 (4 字节)
    Array.Copy(tmpFN, 0, udpSendDataBuf, 4 + 4 + 4, tmpFN.Length);
    bThrIsIdle = false;//线程进入工作状态
    //设置文件传送开始事件对象。
    meStartSend.Set();
  }
}
```

```
    else
    {
        label1.Text = " 请设置正确的 IP 地址";
        textBox1.Focus();
    }
}//button1 按下事件
```

工作线程流程最为复杂，它的主流程与窗体线程的协同任务可由图3-7表示。

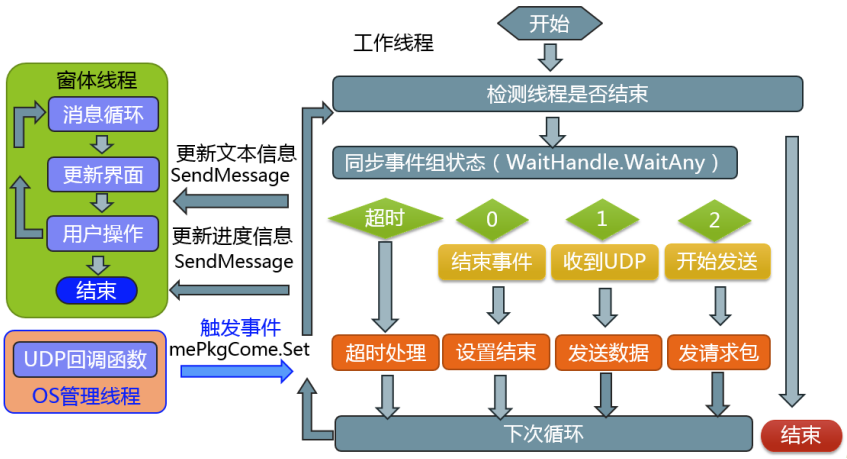

图 3-7 UDP 文件传送工作线程与窗体线程

根据图中的流程设计下面的线程代码：

```
public static void FileSendThr()
{
    int iwr;//同步等待事件的返回值
    bool thrNotFinished = true;
    while (thrNotFinished)//0 表示要求线程结束
    {
        int iReadFileLen = 0;
        //每次循环的的同步等待操作
        iwr = WaitHandle.WaitAny(meAry,1000);
        switch (iwr)
        {
            case 0://0 表示用户要求线程结束
            case 1://收到 UDP 数据
                break;
            case 2://发送文件信息请求包，无响应时会重发
                break;
            case WaitHandle.WaitTimeout://超时
                }
                break;
```

```
        }
    }//线程循环主体
}//线程体
```

用户要求线程结束要执行的代码如下：

```
strMsg = " 线程被结束";
SendMessage(wndHandle, FILETRAN_UPTEXT, 100, 100);
thrNotFinished = false;//结束循环
```

收到接收端的回复包时才开始传输后续数据包，并且在读文件前还要检测回复的编号是否是下一个要传送的编号。

```
mePkgCome.Reset();//消费本次数据包，即重置事件对象
//解析回复数据包
//回复的块编号
iRpyPkgNum = BitConverter.ToInt32(udpRecvDataBuf, 4);
if (iRpyPkgNum == iTotalPkgCount)
{
    //最后数据块发送成功，当前文件传输任务结束
    strMsg = " 文件传输已成功完成";
    SendMessage(wndHandle, FILETRAN_UPTEXT, 100, 100);
    //更新数据进度
    SendMessage(wndHandle, FILETRAN_UPPROG, 100, iTotalPkgCount);
    //关闭当前文件流
    fsCurFile.Close();
    fsCurFile.Dispose();
    bThrIsIdle = true;//线程进入空闲状态
    //重置文件传输开始
    meStartSend.Reset();
    break;
}
if (iRpyPkgNum==(iReadFilePkgNum))
{
    //根据回复编号发送相应编号的数据块
    iCurPkgNum = iRpyPkgNum + 1;
    iRepeatTry = 0;//重置重发次数，每次超时会递增本变量
    //数据缓冲区清零
    Array.Clear(udpSendDataBuf, 0, 1024);
    //要发送数据包格式:[Int32]104(非整块数据)+
    //[int32] 块编号 ( 4 字节)+[int32] 数据字节长度 + 数据内容 (1-1000 字节值)
    tmpByte = BitConverter.GetBytes((Int32)102);//[Int32]102
    Array.Copy(tmpByte, 0, udpSendDataBuf, 0, 4);
```

```
    tmpByte = BitConverter.GetBytes(iCurPkgNum);//[int32] 块编号 (4 字节)
    Array.Copy(tmpByte, 0, udpSendDataBuf, 4, 4);
    //从文件流中读出数据内容，其长度为 1-1000 字节
    iReadFileLen = fsCurFile.Read(udpSendDataBuf, 4 + 4 + 4, 1000);
    tmpByte = BitConverter.GetBytes(iReadFileLen);//[int32] 数据字节长度 (4 字节)
    Array.Copy(tmpByte, 0, udpSendDataBuf, 4 + 4, 4);
    //发送编号为 iCurPkgNum 的数据块，共 [4+4+4+iReadFileLen] 字节
    iUdpBufLen = 4 + 4 + 4 + iReadFileLen;
    SendMessage(wndHandle, FILETRAN_UPPROG, 100, iCurPkgNum);
    iReadFilePkgNum++;
    //发出 UDP 数据包
    skUdpSend.SendTo(udpSendDataBuf, iUdpBufLen, SocketFlags.None, remoteIPEp);
  }
```

准备发送文件时发送的是请求包，包含文件基本信息而不是文件的数据信息。

```
  meStartSend.Reset();//重置开始发送事件，
  //发送 UDP 数据包:[Int32]101+[int32] 总块数 (4 字节) +
  //[int32] 文件名长度 (4 字节)+ 文件名字符串 (字节值)
  iUdpBufLen = 12+tmpFN.Length;
  skUdpSend.SendTo(udpSendDataBuf, iUdpBufLen, SocketFlags.None, remoteIPEp);
```

超时处理将根据程序运行状态，选择对数据包的重发或者判断传送失败或者啥也不做 (没有传输任务)。

```
  if (bThrIsIdle)
  {//线程无事可做
  }
  else
  {//线程有任务
    iRepeatTry++;//累加重发次数
  //检查重发次数，如果超过指定次数则结束文件传输
    if (iRepeatTry == MAX_REPEAT_TIMES)
    {
      if (meStartSend.WaitOne(1))
      {//是文件请求
        strMsg = " 文件传输失败";
      }
      else
      { //数据传输
        strMsg = " 文件传输中断";
      }
      SendMessage(wndHandle, FILETRAN_UPTEXT, 100, 100);
      //关闭当前文件流
```

```
        fsCurFile.Close();
        fsCurFile.Dispose();
        bThrIsIdle = true;//线程进入空闲状态
        //重置文件传输开始
        meStartSend.Reset();
    }
    else
    { //重发数据包，即发相同的数据包
        skUdpSend.SendTo(udpSendDataBuf, iUdpBufLen,
            SocketFlags.None, remoteIPEp);
    }
}
```

调试上述代码时请将其安排在线程合适的位置。

3.4 文件接收端程序实现

接收端窗体界面任务包括显示文件信息和传输进度和线程运行信息，图3-8表示文件接收工作线程与窗体线程的逻辑结构。接收端程序流程与发送端非常类似，在此仅列出工作线程代码，读者可参考发送端自己完成其余部分，完整代码可参考本书所附光盘中的源代码。

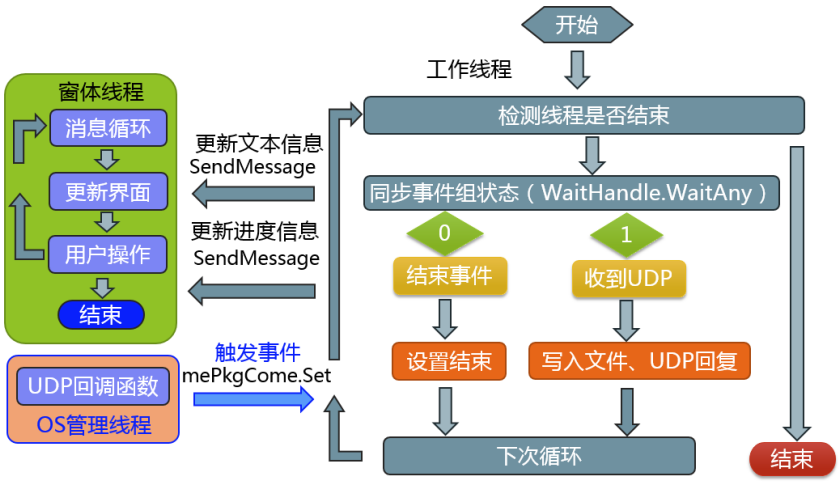

图 3-8 UDP 文件接收工作线程与窗体线程

接收文件端程序的回调函数与发送端非常类似，接收文件端在回调函数中可获得发送者的 IP 地址，相比发送端增加下面两行代码，设置 IP 地址后接收端向发送端回复：

```
//获取远端 IP 地址，并设置端口值，准备回复
remoteIPEp = (IPEndPoint)tmpRemoteEp;
remoteIPEp.Port=UDPSEND_PORT;
```

下面是文件接收端主线程结构源代码：

```
public static void FileRecvThr()
{
```

```
    int iwr;
    iwr = WaitHandle.WaitAny(meAry);
    iWritePkgNum = -1;
    bool thrNotFinished = true;
    while (thrNotFinished)
    {
      int iWriteFileLen = 0;
      iwr = WaitHandle.WaitAny(meAry,1000);//每次循环要执行的的同步等待
      switch (iwr)
      {
        case 0://0 表示用户要求线程结束
          break;
        case 1://收到 UDP 数据
      }
    }//总事件循环
  }
```

线程在收到发来的数据将对各字段进行解析，如果是文件传送请求则创建文件流准备保存数据，对每次数据包检测编号是否写入文件，总块数控制传输的结束。每个接收的数据都将回复相应的包编号，而对于传送请求包对应的编号值为 0，实现代码如下：

```
  mePkgCome.Reset();//消费本次数据包，重置事件对象
  //解析接收的 UDP 数据包
  //数据包类型
  iPkgType = BitConverter.ToInt32(udpRecvDataBuf,0);
  iRecvPkgNum = BitConverter.ToInt32(udpRecvDataBuf, 4);
  //获取数据块长度，设置文件写入长度
  iWriteFileLen = BitConverter.ToInt32(udpRecvDataBuf, 8);
  if (iPkgType == 101)
  {//收到文件传送请求包
    if (iWritePkgNum == -1)
    {
      iWritePkgNum = 0;
      iTotalPkgCount = iRecvPkgNum;
      int iFileNameLen = BitConverter.ToInt32(udpRecvDataBuf, 8);
      string strRecvFileName = Encoding.UTF8.GetString(udpRecvDataBuf, 12, iFileNameLen);
      strSaveFileName = @"d:\"+ strRecvFileName;
      fsSaveFile = new FileStream(strSaveFileName, FileMode.Create, FileAccess.Write);
      strMsg = string.Format(" 正在接收文件 [{0}]", strRecvFileName);
      SendMessage(wndHandle, FILETRAN_UPTEXT, 100, 100);
      SendMessage(wndHandle, FILETRAN_SETPROG, 0, iRecvPkgNum);
```

```
    }
    //忽略重复的请求包
  }
  else {
    if (iRecvPkgNum == (iWritePkgNum + 1))
    {//检查数据包，编号不重复时才写入文件
      iWritePkgNum = iRecvPkgNum;
      fsSaveFile.Write(udpRecvDataBuf, 12, iWriteFileLen);
      SendMessage(wndHandle, FILETRAN_UPTEXT, 100, iWritePkgNum);
      //更新当前进度
      SendMessage(wndHandle, FILETRAN_UPPROG, 100, iRecvPkgNum);
      if ((iWritePkgNum == (iTotalPkgCount)) && fsSaveFile.CanWrite)
      {//所有块接收完成，关闭文件流
        strMsg = " 文件传输成功";
        SendMessage(wndHandle, FILETRAN_UPTEXT, 100, 100);
        fsSaveFile.Flush();
        fsSaveFile.Close();
        fsSaveFile.Dispose();
        //线程进入空闲状态
        bThrIsIdle = true;
      }
    }
  }
  //发送回复包，iCurPkgNum 指示已确认的数据块编号，值为 0 代表准备接收
  tmpByte = BitConverter.GetBytes((int)201);
  Array.Copy(tmpByte,0,udpSendDataBuf,0,4);
  tmpByte = BitConverter.GetBytes(iWritePkgNum);
  Array.Copy(tmpByte,0,udpSendDataBuf,4,4);
  skUdpSend.SendTo(udpSendDataBuf, 8, SocketFlags.None, remoteIPEp);
```

项目实现文件数据的传输，所设计的协议包比较简单，而在复杂网络情况下的文件传输对数据包的较验将能更保证传输数据的可靠性。

3.5 实验作业

1. 调试并完善程序代码，完成本实验中的程序项目。
2. 将原协议添加较验字段对传输的数据进行较验。