南京邮电大学

网络安全课程设计报告

题　　目：木马软件设计

专　　业　网络工程

学　号　姓　名　B15070204

刘美含

指　导　教　师　董小燕

日　　期　2017.11.27-12.10

评分	评分项	优秀	良好	中等	差
	遵守机房规章制度				
	实验原理分析与设计				
	课题功能实现情况				
	设计验收与答辩				
	课程设计报告书写				
简短评语	教师签名：_____ _____年____月____日				
评分等级					
备注					

木马程序设计

一、实验目的和任务

题目：根据木马原理，设计木马 Client 端程序和 Server 端控制程序，Client 端可以控制驻入到被攻击主机的 Server 木马程序，查看被攻击主机的屏幕，进程、建立文件、控制鼠标以及关机等各项操作。

题目解读：

●目标：根据木马原理，设计木马程序；

●模式： C/S 模式；

●功能：

（1）实现远程文件操作（文件的建立、删除等）；

（2）实现远程任务管理（查看远程受控端的当前运行的任务进程）；

（3）远程屏幕监控（查看远程受控端的桌面、操作远程桌面、关机）。

二、攻防原理

木马是是一个通过端口进行通信的网络客户/服务程序，它驻留在目标计算机里，可以随计算机自动启动并在某一端口进行侦听，在对接收的数据识别后，对目标计算机执行特定的操作。其原理是一台主机提供服务(服务器)，另一台主机接受服务(客户机)，作为服务器的主机一般会打开一个默认的端口进行监听，如果有客户机向服务器的这一端口提出连接请求，服务器上的相应程序就会自动运行，来应答客户机的请求。其工作过程见图 1、图 2。

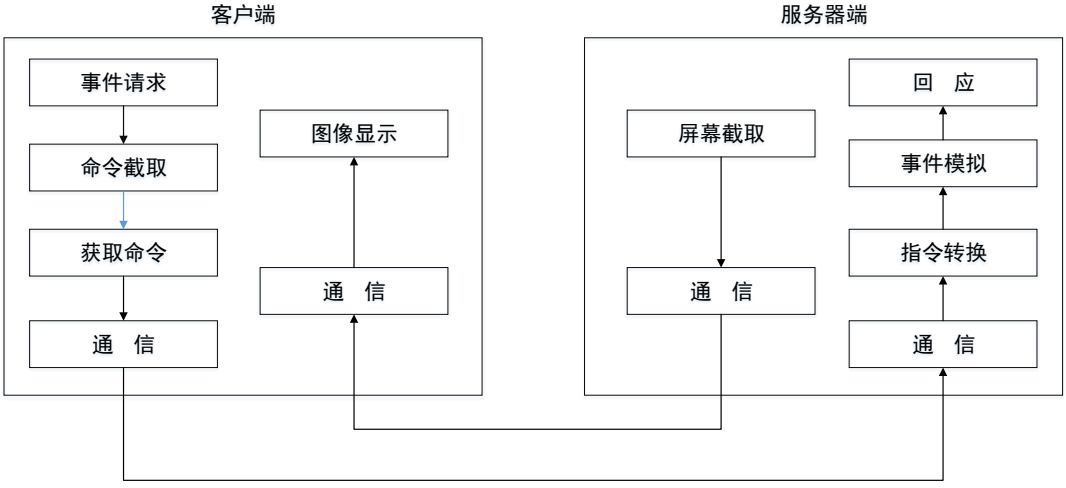

图 1 木马程序工作与 C/S 模式的过程示意图

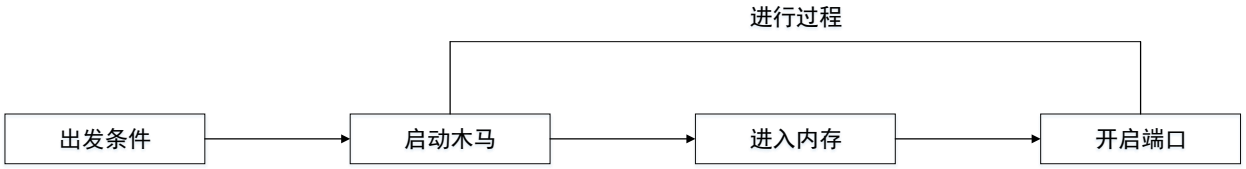

图 2 服务端程序运行过程图

1、木马攻击原理

攻击者利用木马进行网络入侵的攻击过程一般为:首先将木马植入目标系统;然后,木马程序必须能够自动加载运行,并且能够很好地隐藏自己;最后,木马必须可以实现一些攻击者感兴趣的功能。下面从植入技术、自动加载运行技术、隐藏技术和远程监控技术四个方面予以详述。

（1） 植入技术

攻击者向目标主机植入木马,是指攻击者通过各种方式将木马的服务端程序上传到目标主机的过程。木马植入技术可以大致分为主动植入与被动植入两类。所谓主动植入,就是攻击者主动将木马程序种到本地或者是远程主机上,这个行为过程完全由攻击者主动掌握。一般需要通过某种方法获取目标主机的一定权限,然后由攻击者自己动手进行安装。而被动植入是指攻击者预先设置某种环境,然后被动等待目标系统用户的某种可能的操作,只有这种操作执行,木马程序才有可能被植入目标系统。就目前的情况,被动植入技术主要采取欺骗手段,通过各种方式诱使用户运行木马程序。可能的几种方式有：利用 E-mail 附件、利用网页浏览植入、利用移动存储设备植入、与其他可执行程序绑定等。

（2） 自动加载技术

木马程序在被植入目标主机后,不可能寄希望于用户双击其图标来运行启动,只能不动声色地自动启动和运行。在 Windows 系统申,木马程序的自启动方式主要有六种。

①修改系统启动时运行的批处理文件：通过修改系统启动时运行的批处理文件来实现自动启动。通常修改的对象是 autoⅨec.bat、winstall.bat、dosstall.bat 三个批处理文件。

②修改系统文件：木马程序为了达到在系统启动时自动运行的效果,一般需要在第一次运行时修改目标系统的配置文件。通常使用的方法是修改系统配置文件 win.ini 或 system.ini 来达到自动运行的目的。

Win.ini 是位于 C:windows 目录下的一个系统初始化文件。系统在启动时会检索该文件中的相关项,以便对系统环境进行初始设置。在该文件中的“[windows]”字段中有两个数据项“load=”和“run=”,这两项是在系统启动的时候自动加载和运行相关的程序。正

常情况下,这两项为空。如果想要在系统启动时自动加载并运行一个程序,可以将该程序的可执行文件路径信息添加到该数据项后面,系统启动后就会自动运行该程序。System.ini文件中,“[boot]”字段的“shell=程序名”项是系统的引导文件位置,从这里也可以自动加载运行,是木马常用的藏匿之处。

③修改系统注册表：系统注册表保存着系统的软件、硬件及其他与系统配置有关的重要信息：通过设置一些启动加载项目,也可以使木马程序达到自动加载运行的目的,而且这种方法更加隐蔽。通过向 Windows 系统注册表项

HKEY_LOCAL⊥MACHINE\Software\Microsoft\Windows\CurrentVersion\

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\

下的 Run、RunOnce、RmOnceEx、RunServices 或 RunServicesOnce 添加键值可以容易地实现木马程序的自启动。如果在带有 Once 的项中添加木马的启动项,则更具隐蔽性,因为带有 Once 的项中的键值在程序运行后将被系统自动删除。

④添加系统服务：Windows NT 引入了系统服务的概念,其后的 NT 内核操作系统如、Windows 2000/XP/2003 都大量使用服务来实现关键的系统功能。服务程序是一类长期运行的应用程序,它不需要界面或可视化输出,能够设置为在操作系统启动时自动开始运行,而不需要用户登录来运行它。除了操作系统内置的服务程序外,用户也可以注册自己的服务程序。木马程序就是利用了这一点,将自己注册为系统的一个服务并设置为自动运行,这样每当 Windows 系统启动时,即使没有用户登录,木马也会自动开始工作。

⑤修改文件关联属性：对于一些常用的文件,如.txt 文件,只要双击文件图标就能打开这个文件。这是因为在系统注册表中,已经把这类文件与某个程序关联起来,双击这类文件时,系统就会自动启动相关联的程序来打开文件。修改文件关联属性是木马程序常用的手段,通过这一方式,即使用户在打开某个正常的文件时,也能在无意中启动木马。著名的国产木马冰河用的就是这种方式,它通过修改注册表中

HKEY_CLASSES_ROOT\txtfile\shell\open\command

下的键值,将：

C:\WINDOWS\NOTEPAD.EXE %1

修改为：

C:\WINDOWS\SYSTEM\SYSEXPLR.EXE %1

这样一旦双击了一个.txt 文件,原本是应该用 Notepad.exe 打开却变成启动木马程序了。其他类似的方式包括修改.htm 文件、.exe 件和 unknown(未知)文件的关联。

⑥利用系统自动运行的程序：Windows 系统中有很多程序是可以自动运行的,如在对磁盘进行格式化后,总是要运行“磁盘扫描”程序(scandisk.exe);按 F1 键时,系统将运行 winhelp.exe 或 hh.exe 打开帮助文件;系统启动时,系统将自动启动“系统栏”程序(SysTray.exe)、“输入法”程序(internat.exe)、“注册表检查”程序(scanreg.exe)、“计划任务”程序(mstask.exe)、“电源管理”程序等。这为木鸟程序提供了机会,通过覆盖相应文件就可获得自动启动的能力,而不必修改系统任何设置。

（3） 隐藏技术

木马程序与普通程序不同,它在启动之后,要想尽一切办法隐藏自己,保证自己不出现在任务栏、任务管理器和服务管理器中。隐藏又分为伪隐藏和真隐藏。伪隐藏是指木马程序的进程仍然存在,只不过是让它消失在进程列表里,但在服务管理器中很容易发现系统注册过的服务;而真隐藏则是让程序彻底地消失,不以一个进程或者服务的方式工作。实现伪隐藏的方式主要是进行进程欺骗。木马隐藏自身的第二种方式是不使用进程,即所谓的真隐藏。这种方法一般使用 Windows 系统动态链接库(DLL)来避开检测。

（4） 监控技术

木马通过客户端/服务端模式来建立与攻击者之间的联系。服务端程序接受传入的连接请求和其他命令,为另一端提供信息和其他服务;客户端程序主动发起连接,向服务端发送命令,并接受对端返回来的信息。建立连接时,木马的服务端(即植入到目标主机的那部分程序)会在目标主机上打开一个特定的端口,监听外界的连接请求。这时,攻击者就可以使用木马的客户端去连接这个服务端,然后从服务端接收木马传回的信息,并向服务端发送命令以控制目标主机。可以在 Visual Basic 中用 Winsock 控件来模拟实现这个过程(以下代码中 G_Server 和 G_Client 均为 Winsock 控件)。

2、木马的防御

一旦怀疑系统感染木马，首先要做的就是检查系统，确定木马是否真的存在。根据木马工作的原理，木马的检测一般有以下一些方法。

（1） 端口扫描和连接检查

扫描端口是检测木马的常用方法。大部分的木马服务端会在系统中监听某个端口，因此通过查看系统上开启了哪些端口能有效地发现远程控制木马的踪迹。操作系统本身就提供了查看端口状态的功能。在命令行下输入 netstat－na 可以查看系统当前已经建立的连接和正在监听的端口，同时可以查看正在连接的远程主机 IP 地址。有的工具甚至可以直接查看与之相关的程序名称，为过滤可疑程序提供了方便。

（2） 检查系统进程

虽然现在也有一些技术使木马进程不显示在进程管理器中，不过绝大多数的木马在运行期都会在系统中生成进程。因此，检查进程列表是一种非常有效的发现木马踪迹的方法。使用进程检查的前提是需要管理员了解系统正常情况下运行的系统进程。这样当有不属于正常的系统进程出现时，管理员能很快发现。

（3） 检查.ini 文件、注册表和服务

为使木马自动运行，大部分的木马都会把自己登记在开机启动的程序当中，这样才能在计算机开机后自动加载。也有少数木马采用文件绑定的方式，将木马与特定的可执行文件进行绑定，木马随着这个文件的运行而自动运行。

（4） 监视网络通信

一些特殊的木马程序使用 ICMP 协议通信，被控制不需要打开任何监听端口，也无须反向连接，更不会有什么已经建立的固定连接，这使得 netstat 或 fport 等工具很难发挥作用。对付这种木马，除了检查可疑进程之外，还可以通过嗅探器软件（也称 sniffer 软件）监视网络通信来发现可疑情况。首先关闭所有有网络行为的已知合法程序，然后打开嗅探器软件进行监听，若在这种情况下仍然有大量的数据传输，则基本可以确定后台正运行着恶意程序。这种方法并不是非常准确，而且要求对系统和应用软件较为熟悉，因为某些带自动升级功能的软件也会产生类似的数据流量。

三、实验环境

1、实验拓扑图

本系统采用 C/S 工作模式 ， 由主控端程序、被控端程序两部分构成。攻防机器均位于同一局域网内，其拓扑结构图如图 3 所示：

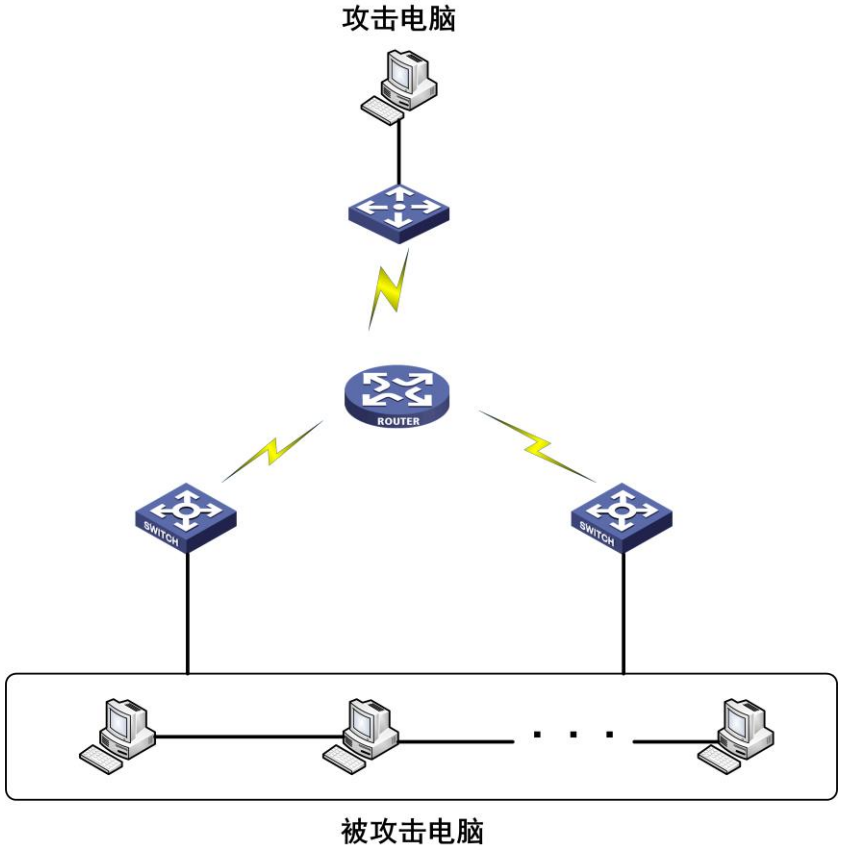

图 3 拓扑结构图

2、软硬件配置

本实验所采用的机器配置如表 1 所示：

表 1 机器配置

	硬件环境		软件环境	
	CPU	内存	操作系统	开发工具
攻击机器	4G	256G	Windows XP	Visual C++ 6.0
被攻击机器	4G	256G	Windows XP	Visual C++ 6.0

四、实验设计与实现

本系统包含两个独立的应用程序：客户端程序和服务器端程序。使用 C / S 模型设计：客户向服务器提出请求，服务器接收请求后，提出相关的服务。服务器程序通常在一个已经设定的端口 5180 进行监听，直到一个客户机程序提出了请求信息，此时，服务器程序被唤醒并且为客户提供服务。以上过程的网络通信是基于 TCP / IP 进行的，文件传输通过 UDP 协议进行。

1、 C/S 建立的基本步骤（流程图如图 4 所示）

服务端：

Step1：初始化 WSA

Step2：建立 SOCKET

Step3：绑定 SOCKET

Step4：在指定的端口监听

Step5：接受一个连接

Step6：发送接受数据

Step7：断开连接

客户端：

Step1：初始化 WSA

Step2：建立 SOCKET

Step3：连接到服务器

Step4：发送接受数据

Step5：断开连接。

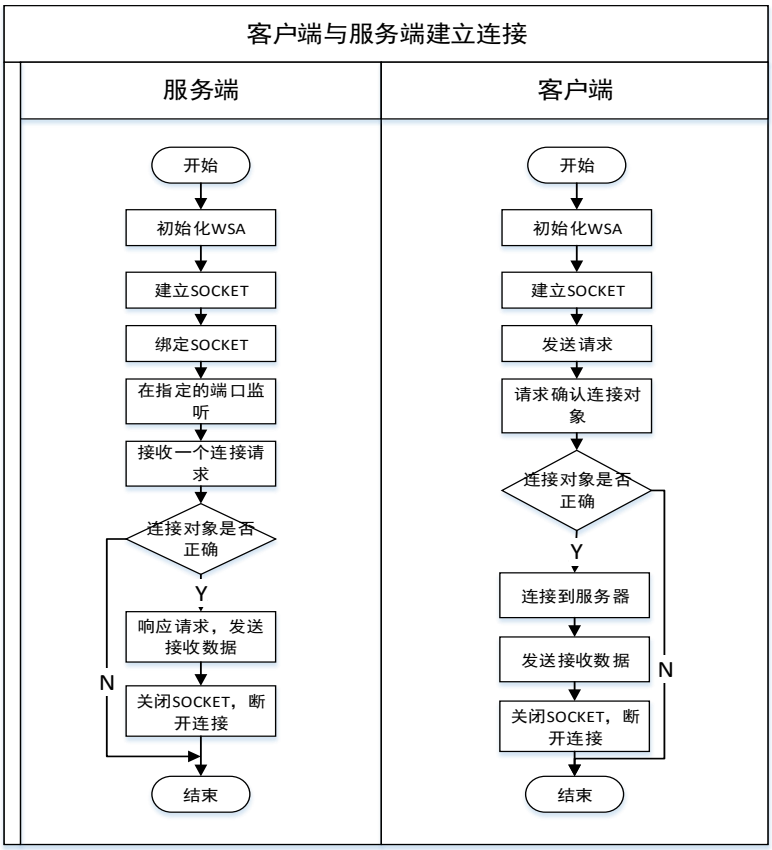

图 4 C/S 建立流程图

2、 框架设计

本系统由客户端和服务端两部分组成。服务端执行程序后，则处于端口监听状态，直到与客服端连接成功；然后根据客户端传送的命令执行相应服务；若客户端发送断开连接请求，则断开连接；其框架设计图如图 5 所示。客户端程序执行后，输入服务器端的 IP 地址建立连接，根据需要发送命令，并等待服务器端返回命令执行结果，其框架设计图如图 6 所示。

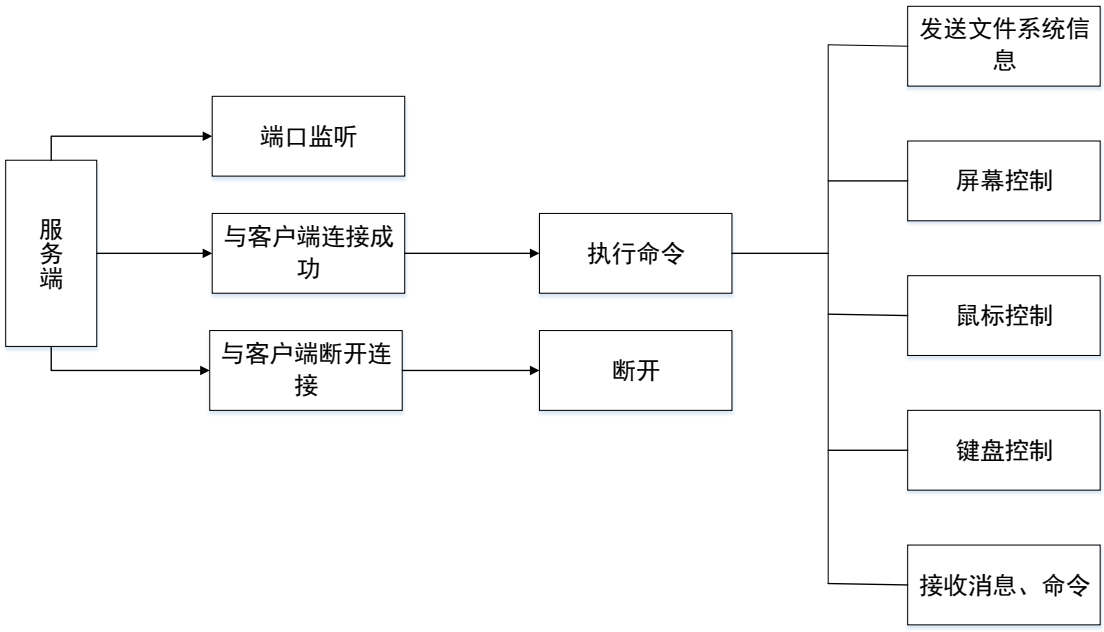

图 5 服务端框架设计图

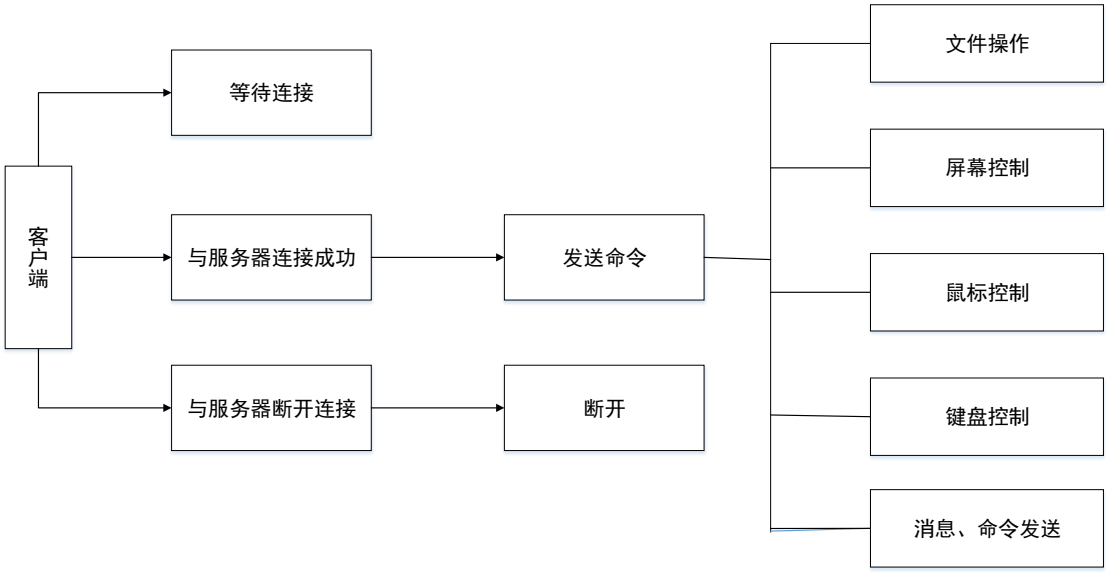

图 6 客户端框架设计图

3、 主要类定义

表 2 主要类定义

类名	解释
CAboutDlg	对话框类定义
CChildFrame	子框架类定义
CCompress	压缩类定义
CDecodeBitArray	解码 bitarray 类，有效处理 bool 值集合，以防越界
CEncodeBitArray	编码 bitarray 类，有效处理 bool 值集合，以防越界
CLZWDecode	LZW 压缩解码
CLZWEncode	LZW 压缩解码类定义
CMainClientWnd	客户端口主界面类定义
CMainFrame	主框架类定义
CPeeperBar	框架缓冲池
CPeeperClientDoc	客户端文件操作函数类定义
CPeeperThread	线程类定义
CPeeperSetdlg	对话框建立类定义
CPeeperWnd	窗口类定义
CRegisterDlg	注册表类定义
LZWDecodeEntry	LZW 解码入口类定义
LZWEncodeEntry	LZW 编码入口类定义
PEEPERMENU	主菜单类定义
PL_CopyFileClient	客户端复制文件类定义
PL_CopyFileServer	服务端复制文件类定义
PL_CreateDIBPalette	DIB 图形调色板类定义
PL_DeleteFile	删除文件类定义
PL_MoveFile	移动文件类定义
PL_GetHostName	获取用户名类定义
PL_GetScreenSize	获取屏幕类定义
PL_InitSocket	初始化 SOCKET 类定义
PL_KeyDown	键盘锁定类定义

PL_KeyUp	键盘解锁类定义
PL_LockDesktop	锁定桌面类定义
PL_MouseLButtonDown	鼠标左键锁定类定义
PL_MouseLButtonUp	鼠标左键解锁类定义
PL_MouseMove	移动鼠标类定义
PL_MouseRButtonDown	鼠标右键锁定类定义
PL_MouseRButtonUp	鼠标右键解锁类定义
PL_PaletteSize	调色板大小类定义
PL_SendMail	客户端发送消息类定义
PL_SendMail2	客户端发送命令类定义

4、 主要模块分析及实现

主要分为服务器主功能模块、SOCKET 消息传输模块、和客户端主功能三个模块。客户端主要功能包括:连接测试、屏幕监控和操作、文件操作、停止控制、锁定键盘和解锁、发送消息和命令等功能；除此之外，客户端还可选择**不同的压缩算法**对传输来的图像进行压缩，并根据不同需求选择不同的显示方案。

（1）服务器主功能模块（队友负责）

服务器端运行后，首先开始监听端口，直到接收到客户机 IP 地址，从客户端获取所有配置信息，在跟据客户端的要求进行后面的操作。服务器主功能模块活动图如图 7 所示：

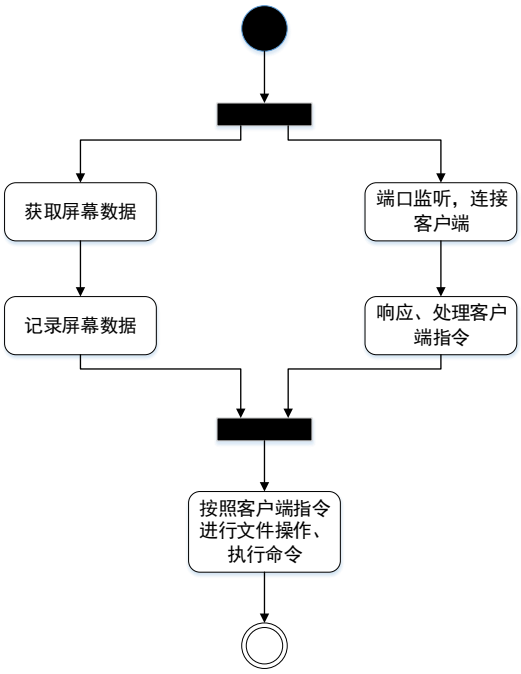

图 7 服务器主功能模块活动图

- 获得主机 IP 和端口

```
BOOL WINAPI PL_GetHostName(char *chIP, char *chName)
{
    BOOL bRet = FALSE;
    char chTemp[256];
    hostent* pEnt = NULL;
    ZeroMemory(chTemp, 256);
    int nRet = ::gethostname(chTemp, 256);
    if(nRet == 0)
    {
        if(AfxIsValidAddress(chName, strlen(chTemp)))
        {
            strcpy(chName, chTemp);
            bRet = TRUE;
        }
        if(AfxIsValidAddress(chIP, 16))
        {
            pEnt = ::gethostbyname(chTemp);
            if(pEnt)
            {
                sprintf(chIP, "%d.%d.%d.%d",
                    BYTE(pEnt->h_addr_list[0][0]), BYTE(pEnt->h_addr_list[0][1]),
                    BYTE(pEnt->h_addr_list[0][2]), BYTE(pEnt->h_addr_list[0][3]));
                bRet = TRUE;
            }
            else
            {
                bRet = FALSE;
            }
        }
    }
    return bRet;
}
```

- 清除网络连接

```
BOOL WINAPI PL_DetectInternetConnect(CString strHTTP)//清楚网络连接，session
{
    BOOL bRet = FALSE;
    HINTERNET hSession = NULL;
    hSession = ::InternetOpen(_T("peeper"), PRE_CONFIG_INTERNET_ACCESS, NULL, NULL, 0);//它告诉 Internet DLL 初始化内部数据结构并准备接收应用
    /*
lpszAgent
指向一个空结束的字符串，该字符串指定调用WinInet函数的应用程序或实体的名称。使用此名称作为用户代理的HTTP协议。
dwAccessType
指定访问类型，参数可以是下列值之一：
Value   Meaning
INTERNET_OPEN_TYPE_DIRECT   使用直接连接网络。
INTERNET_OPEN_TYPE_PRECONFIG    获取代理或直接从注册表中的配置，使用代理连接网络。
INTERNET_OPEN_TYPE_PRECONFIG_WITH_NO_AUTOPROXY  获取代理或直接从注册表中的配置，并防止启动Microsoft JScript或Internet设置（INS）文件的使用。
INTERNET_OPEN_TYPE_PROXY    通过代理的请求，除非代理旁路列表中提供的名称解析绕过代理.在这种情况下，该功能的使用。
成功：返回一个有效的句柄，该句柄将由应用程序传递给接下来的WinINet函数。
失败：返回NULL。
    */
    if(hSession)
    {
        HINTERNET hHTTPSession = ::InternetOpenUrl(hSession,
            strHTTP, NULL, 0, INTERNET_FLAG_RELOAD, 0);//通过一个完整HTTP打开一个资源。
        if(hHTTPSession)
        {
            bRet = TRUE;
            ::InternetCloseHandle(hHTTPSession);//关闭
        }
        ::InternetCloseHandle(hSession);
    }
    return bRet;
}
```

（2）SOCKET 消息传输模块（我负责）

为了简化基于 Windows 的 Socket 网络编程，而更专注于应用程序算法的设计，我使用基本类库(MFC)提供的 CSocket 类,CSocket 类为 WinSockAPI 提供了高级编程接口，CSocket 类可以和 CSocketFile 类、CAchive 类一起工作来处理数据的发送和接收，从而使用户避免手工处理字节顺序字符串顺序等问题，而且 CSocket 类提供的阻塞调用功能恰好是使用 CAchive 类进行同步数据传输的最基本要求。远程控制用三个类实现控制端通信层,分别是：CMsg、CsktClient 和 CzclientDlg，实现建立连接、数据收发过程的功能。

- 初始化：

```
#pragma comment(lib, "Wininet.lib")
/*
对套接字进行编程，首先初始化Socket端口，
并在初始化成功的前提下创建一个套接字，
然后将该套接字和本地网络地址联系在一起，
再套接字做好侦听的准备，并规定它的请求队列的长度。

该套接字被设为“”被动“”模式，负责响应到来的联接，
并由进程将到来的联接排队挂起。该函数典型地用于需要同时有多个联接的服务器：
如果一个联接请求到达且队列已满，客户端将收到一个 WSAECONNREFUSED 的错误。
*/
BOOL WINAPI PL_InitSocket()//Socket初始化
{
#define MAJOR_VERSION 1
#define MINOR_VERSION 2

    int nStatus = 0;
    WORD wMajorVersion = MAJOR_VERSION;//主版本号
    WORD wMinorVersion = MINOR_VERSION;//次版本号
    WORD wVersionReqd = MAKEWORD(wMajorVersion, wMinorVersion);//调用版本
    WSADATA lpmyWSAData;//这个结构被用来存储被WSAStartup函数调用后返回的Windows Sockets数据。它包含Winsock.dll执行的数据。
    nStatus = ::WSAStartup(wVersionReqd, &lpmyWSAData);//创建套接字
    if(nStatus != 0)
    {
        return FALSE;
    }

    return TRUE;
}
```

- 释放套接字空间：

```
BOOL WINAPI PL_TermSocket()
{
    //应用程序在完成对请求的Socket库的使用后，要调用WSACleanup函数来解除与Socket库的绑定并且释放Socket库所占用的系统资源。
    return (::WSACleanup() == 0)?TRUE : FALSE;
}
```

- 发送套接字数据：

```
//发送Stocket数据
int WINAPI PL_SendSocketData(SOCKET s, BYTE *chData, int nLen, BYTE chFlag, UINT uFlag)
{                                                                                //32位无符号整数
    int nRet = INVALID_SOCKET;//把socket设置成无效套接字
    if(s != INVALID_SOCKET)//若创建套接字成功
    {
        char *chTemp = new char[nLen + 3];
        ZeroMemory(chTemp, nLen + 3);//将结构中所有字节置0
        if(chFlag == PL_NONE)
        {
            if(chData != NULL)
            {
                if(uFlag == MSG_OOB)//现在进程使用以MSG_OOB 为参数的send()函数写入一个单字节的”带外数据”
                {
                    //实际数据大小为N,以MSG_OOB发送时数据长度要加1
                    nLen += 1;
                }
                nRet = ::send(s, (char *)chData, nLen, uFlag);
            }
            else
            {
                nRet = 0;
            }
        }

        else
        {
            chTemp[0] = chFlag;
            if(chData != NULL)
            {
                memcpy(chTemp + 1, chData, nLen);
            }
            else
            {
                nLen = 0;
            }
            if(uFlag == MSG_OOB)
            {
                //实际数据大小为N,以MSG_OOB发送时数据长度要加1
                nLen += 1;
            }
            nRet = ::send(s, chTemp, nLen+1, uFlag);
        }
        delete []chTemp;
    }

    return nRet;
}
```

- 读套接字传递的数据：

```
int WINAPI PL_ReadSocketData(SOCKET s, BYTE *chData, int nLen, BYTE *chFlag, UINT uFlag)//读数据
{
    int nRet = INVALID_SOCKET;
    if(s != INVALID_SOCKET)
    {
        nRet = ::recv(s, (char *)chData, nLen, uFlag);
        /*
        参数一：指定接收端套接字描述符；
        参数二：指明一个缓冲区，该缓冲区用来存放recv函数接收到的数据；
        参数三：指明buf的长度；
        参数四 ：一般置为0。
        如果协议在传送s的发送缓冲中的数据时出现网络错误，那么recv函数返回SOCKET_ERROR；
        如果s的发送缓冲中没有数据或者数据被协议成功发送完毕后，recv先检查套接字s的接收缓冲区，如果s接收缓冲区中没有数据或者协议正在接收数据，
        那么recv就一直等待，直到协议把数据接收完毕；
        当协议把数据接收完毕，recv函数就把s的接收缓冲中的数据copy到buf中（注意协议接收到的数据可能大于buf的长度，
        所以在这种情况下要调用几次recv函数才能把s的接收缓冲中的数据copy完。recv函数仅仅是copy数据，真正的接收数据是协议来完成的），
        recv函数返回其实际copy的字节数；
        如果recv在copy时出错，那么它返回SOCKET_ERROR；如果recv函数在等待协议接收数据时网络中断了，那么它返回0。
        */
        if(nRet > 0)
        {
            if(chFlag != NULL)
            {
                *chFlag = chData[0];
            }
        }
    }
    return nRet;
}
```

（3）客户端主功能模块：

这个模块集合了绝大部分客户机端所需求的功能，它接受用户对服务器端进行所期望的设置，进行监听，等待服务器的连接，当连接来到时，将配置信息发送给服务器端，并等待服务器端的相应，根据相应内容完成后续传递文件等内容。下图为客户端主功能模块流程：

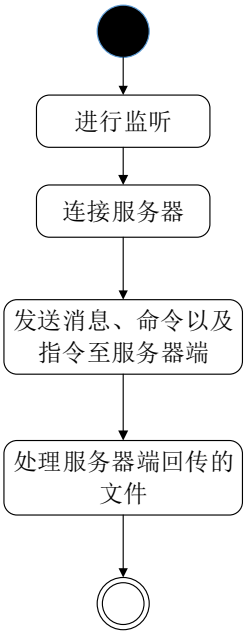

图 8 客户端主功能模块流程

①屏幕控制模块（我负责）

该模块实现客户端以系统管理员身份进入服务器计算机 ，异地控制服务器计算机，可以像对本地计算机一样对其进行操作. 远程模拟鼠标及键盘输入， 运行服务器计算机上的可执行程序，修改其系统设置，控制其硬件设备。控制端发送鼠标动作或键盘输入给服务端，服务端接收后响应其命令，返回响应结果信息给控制端，并截屏，将位图文件传给控制端。屏幕监控流程图如图 9 所示。

服务端在接收到客户端的屏幕数据请求后，通过使用当前屏幕设备的句柄，开始向开辟的内存区域复制屏幕数据，得到与设备相关的 GDI 位图；然后再通过设置位图信息头、调色板等，最后得到与设备无关的 DIB 位图。

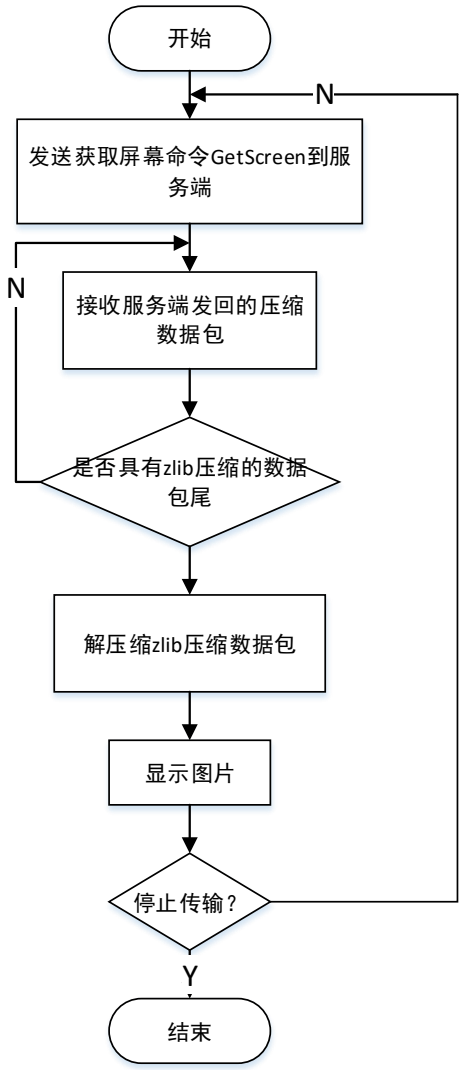

图 9 屏幕监控流程图

- 绘制对方屏幕

```
BOOL WINAPI PL_DrawBmp(HDC hDC, LPRECT lpDCRect, HBITMAP hBmp, LPRECT lpBmpRect, CPalette* pPal)
{
    HDC hMemDC = NULL;
    BITMAP bm;

    hMemDC = ::CreateCompatibleDC(hDC);
    ::GetObject(hBmp, sizeof(bm), &bm);
    ::SelectObject(hMemDC, hBmp);

    if(lpBmpRect == NULL)
    {
        if(lpDCRect != NULL)
        {
            ::SetStretchBltMode(hDC, COLORONCOLOR);
            ::StretchBlt(hDC, lpDCRect->left, lpDCRect->top, RECTWIDTH(lpDCRect), RECTHEIGHT(lpDCRect),
                hMemDC, 0, 0, bm.bmWidth, bm.bmHeight, SRCCOPY);
        }
        else
        {
            ::BitBlt(hDC, 0, 0, bm.bmWidth, bm.bmHeight, hMemDC, 0, 0, SRCCOPY);
        }
    }
    else
    {
        if(lpDCRect != NULL)
        {
            ::SetStretchBltMode(hDC, COLORONCOLOR);
            ::StretchBlt(hDC, lpDCRect->left, lpDCRect->top, RECTWIDTH(lpDCRect), RECTHEIGHT(lpDCRect),
                hMemDC, lpBmpRect->left, lpBmpRect->top, RECTWIDTH(lpBmpRect), RECTHEIGHT(lpBmpRect), SRCCOPY);
        }
        else
        {
            ::BitBlt(hDC, 0, 0, RECTWIDTH(lpBmpRect), RECTHEIGHT(lpBmpRect),
```

```
                hMemDC, lpDCRect->left, lpDCRect->top, SRCCOPY);
        }
    }
    ::ReleaseDC(NULL, hMemDC);

    return TRUE;
}

int WINAPI PL_ColorsNum(LPBYTE lpbi)
{
    WORD wBitCount = 0;
    if(IS_WIN30_DIB(lpbi))
    {
        DWORD dwClrUsed = ((LPBITMAPINFOHEADER)lpbi)->biClrUsed;
        if(dwClrUsed != 0)
            return dwClrUsed;
    }
    if(IS_WIN30_DIB(lpbi))
        wBitCount = ((LPBITMAPINFOHEADER)lpbi)->biBitCount;
    else
        wBitCount = ((LPBITMAPCOREHEADER)lpbi)->bcBitCount;

    switch (wBitCount)
    {
        case 1:
            return 2;

        case 4:
            return 16;

        case 8:
            return 256;
```

```
        default:
            return 0;
    }
}

int WINAPI PL_DIBWidth(LPBYTE lpDIB)
{
    LPBITMAPINFOHEADER lpbmi;
    LPBITMAPCOREHEADER lpbmc;

    lpbmi = (LPBITMAPINFOHEADER)lpDIB;
    lpbmc = (LPBITMAPCOREHEADER)lpDIB;

    if (IS_WIN30_DIB(lpDIB))
        return lpbmi->biWidth;
    else
        return lpbmc->bcWidth;
}
```

- 得到位图的高度和宽度

```
int WINAPI PL_DIBWidth(LPBYTE lpDIB)
{
    LPBITMAPINFOHEADER lpbmi;
    LPBITMAPCOREHEADER lpbmc;

    lpbmi = (LPBITMAPINFOHEADER)lpDIB;
    lpbmc = (LPBITMAPCOREHEADER)lpDIB;

    if (IS_WIN30_DIB(lpDIB))
        return lpbmi->biWidth;
    else
        return lpbmc->bcWidth;
}

int WINAPI PL_DIBHeight(LPBYTE lpDIB)
{
    LPBITMAPINFOHEADER lpbmi;
    LPBITMAPCOREHEADER lpbmc;

    lpbmi = (LPBITMAPINFOHEADER)lpDIB;
    lpbmc = (LPBITMAPCOREHEADER)lpDIB;

    if (IS_WIN30_DIB(lpDIB))
        return lpbmi->biHeight;
    else
        return lpbmc->bcHeight;
}
```

- 不同颜色位图转化

```
//位图转化 lpbi转换的颜色位数
int WINAPI PL_PaletteSize(LPBYTE lpbi)//LPBYTE字节指针
{
   if(IS_WIN30_DIB(lpbi))
      return (WORD)(::PL_ColorsNum(lpbi)*sizeof(RGBQUAD));
   else
      return (WORD)(::PL_ColorsNum(lpbi)*sizeof(RGBTRIPLE));
}
```

②鼠标控制模块（队友负责）

该模块实现的功能是控制服务器计算机对本地用户的鼠标事件的响应与否,可以随时锁定或解锁。

当使用鼠标点击客户端中显示服务端当前屏幕的区域，客户端程序将会记录下具体的左/右键，单/双击等信息，作为鼠标事件发送给服务端，服务端随后进行解析，并做出相

应的响应，从而实现客户端得到服务端屏幕并加以控制的功能。

以下函数分别为：发送双击鼠标信息、发送单击鼠标左键（按下/抬起）、发送单击鼠标右键（按下/抬起）。

```
void WINAPI PL_MouseLButtonDown(POINT point, BOOL bMove)
{
    if(bMove)
    {
        ::PL_MouseMove(point);
    }
    ::mouse_event(MOUSEEVENTF_LEFTDOWN, point.x, point.y, 0, 0);
}

void WINAPI PL_MouseLButtonUp(POINT point, BOOL bMove)
{
    if(bMove)
    {
        ::PL_MouseMove(point);
    }
    ::mouse_event(MOUSEEVENTF_LEFTUP, point.x, point.y, 0, 0);
}

void WINAPI PL_MouseRButtonDown(POINT point, BOOL bMove)
{
    if(bMove)
    {
        ::PL_MouseMove(point);
    }
    ::mouse_event(MOUSEEVENTF_RIGHTDOWN, point.x, point.y, 0, 0);
}

void WINAPI PL_MouseRButtonUp(POINT point, BOOL bMove)
{
    if(bMove)
    {
        ::PL_MouseMove(point);
    }
    ::mouse_event(MOUSEEVENTF_RIGHTUP, point.x, point.y, 0, 0);
}

void WINAPI PL_MouseLButtonDblClk(POINT point, BOOL bMove)
{
    if(bMove)
    {
        ::PL_MouseMove(point);
    }
    ::PL_MouseLButtonDown(point);
    ::PL_MouseLButtonUp(point);
    ::PL_MouseLButtonDown(point);
    ::PL_MouseLButtonUp(point);
}

void WINAPI PL_MouseRButtonDblClk(POINT point, BOOL bMove)
{
    if(bMove)
    {
        ::PL_MouseMove(point);
    }
    ::PL_MouseRButtonDown(point);
    ::PL_MouseRButtonUp(point);
    ::PL_MouseRButtonDown(point);
    ::PL_MouseRButtonUp(point);
}
```

③键盘控制模块（队友负责）

该模块实现的功能是控制服务器计算机对本地用户的键盘事件的响应与否,可以随时锁定或解锁。

```
void WINAPI PL_KeyDown(UINT uChar, UINT uFlag)
{
    ::keybd_event((BYTE)uChar, (BYTE)uChar, 0, 0);
}

void WINAPI PL_KeyUp(UINT uChar, UINT uFlag)
{
    ::keybd_event((BYTE)uChar, (BYTE)uChar, KEYEVENTF_KEYUP, 0);
}
```

④文件操作模块（队友负责）

该模块使用 TCP、UDP 传输协议，在客户端服务端之间进行文件的一系列操作，包括：建立文件，删除文件，复制文件，查看文件等。文件操作流程图如图 10 所示：

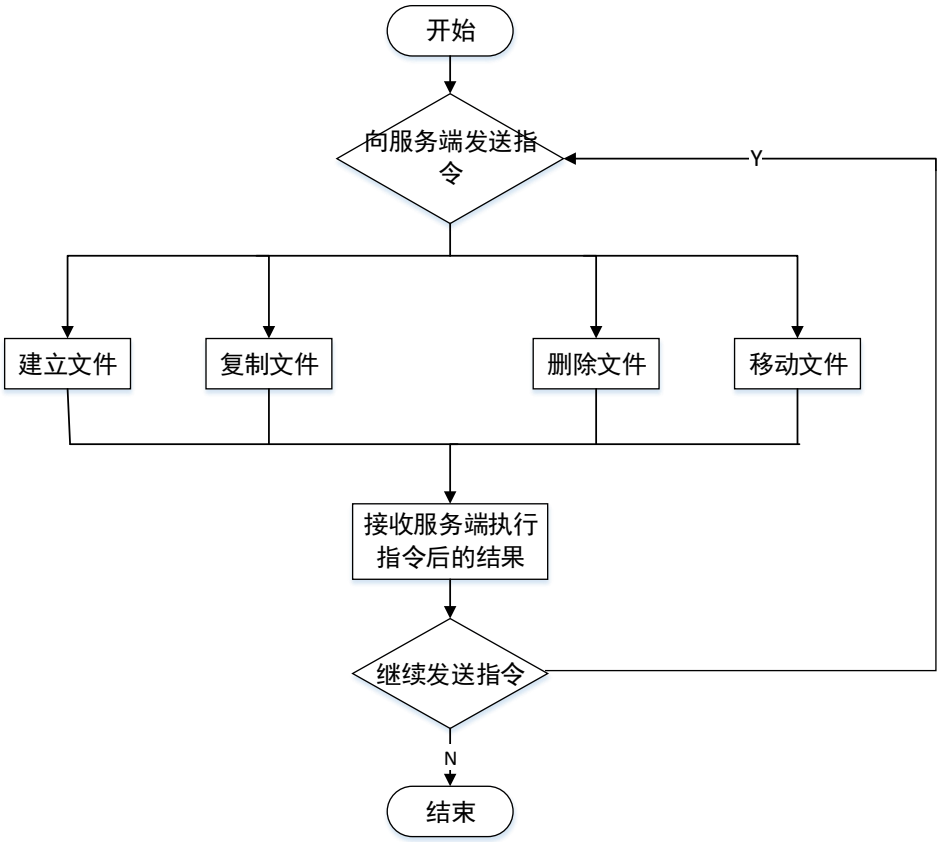

图 10 文件操作流程图

- 复制文件

```
BOOL WINAPI PL_CopyFileClient(CString strIP, UINT uPort, char *chSrc,
                              BOOL bSend, HWND hNotifyWnd)
{
    BOOL bRet = FALSE;
    CSocket sckClient;
    CFile file;
    /*
    虽然使用CArchive类内建的序列化功能是保存和加载持久性数据的便捷方式，
    但有时在程序中需要对文件处理过程拥有更多的控制权，
    对于这种文件输入输出（I/O）服务的需求，Windows提供了一系列相关的API函数，
    并由MFC将其封装为CFile类，
    提供了对文件进行打开，关闭，读，写，删除，重命名以及获取文件信息等文件操作的基本功能，
    足以处理任意类型的文件操作。CFile类是MFC文件类的基类，
    支持无缓冲的二进制输入输出，也可以通过与CArchive类的配合使用而支持对MFC对象的带缓冲的序列化。
    */
    do
    {
        if(bSend)
        {
            bRet = file.Open(chSrc, CFile::modeRead);
        }
        else
        {
            bRet = file.Open(chSrc, CFile::modeCreate | CFile::modeWrite);
        }
        if(!bRet)
        {
            break ;
        }
        bRet = sckClient.Create();
        if(!bRet)
        {
```

```
            break ;
        }
        bRet = sckClient.Connect(strIP, uPort);
        if(!bRet)
        {
            break ;
        }
        BYTE *chFileData = new BYTE[PL_SOCKET_MAXBYTES+1];
        char chData[10];
        int nSize = 0;
        if(bSend)
        {
            nSize = file.GetLength();
            int nRead = nSize;
            ZeroMemory(chData, 10);
            memcpy(chData, &nSize, sizeof(int));
            int nRet = sckClient.Send(chData, sizeof(int));
            if(nRet <= 0)
            {
                bRet = FALSE;
                break ;
            }
            while(nRead > 0)
            {
                int n1 = min(nRead, PL_SOCKET_MAXBYTES);
                int n2 = file.Read(chFileData, n1);
                nRead -= n2;
                nRet = sckClient.Send(chFileData, n2);
                if(nRet <= 0)
                {
                    bRet = FALSE;
                    break ;
                }

                if(hNotifyWnd != NULL)
                {
                    int nProgress = (int)(((float)(nSize - nRead))*100.0f/((float)nSize));
                    ::PostMessage(hNotifyWnd, PL_PEEPER_NOTIFY_COPYFILE,
                        (WPARAM)nProgress, NULL);
                }
                bRet = TRUE;
            }
        }
        else
        {
            char chData[10];
            ZeroMemory(chData, 10);
            nSize = 0;
            int nRet = sckClient.Receive(chData, 10);//读文件大小
            if(nRet <= 0)
            {
                bRet = FALSE;
                break ;
            }
            nSize = *((int*)(chData));
            if(nSize <= 0)
            {
                bRet = FALSE;
                break ;
            }
            BYTE *chFileData = new BYTE[PL_SOCKET_MAXBYTES+1];
            int nWrite = 0;
            while(nWrite < nSize)
            {
                nRet = sckClient.Receive(chFileData, PL_SOCKET_MAXBYTES);
                if(nRet <= 0)
                {
                    bRet = FALSE;
                    break ;
                }
                file.Write(chFileData, nRet);
                nWrite += nRet;
                if(hNotifyWnd != NULL)
                {
                    int nProgress = (int)(((float)nWrite)*100.0f/((float)nSize));
                    ::PostMessage(hNotifyWnd, PL_PEEPER_NOTIFY_COPYFILE,
                        (WPARAM)nProgress, NULL);
                }
                bRet = TRUE;
            }
        }
        delete []chFileData;
        if(!bRet) // 送文件数据出错.
        {
            break ;
        }
    }while(0);
    if(sckClient.m_hSocket !=   INVALID_SOCKET)
    {
        sckClient.Close();
    }
    if(file.m_hFile != CFile::hFileNull)
    {
        file.Close();
    }

    return bRet;
}
```

- 删除文件

```
BOOL WINAPI PL_DeleteFile(char *chFileName)
{
    BOOL bRet = FALSE;
    DWORD dwFlag = ::SetFileAttributes(chFileName, FILE_ATTRIBUTE_NORMAL);
    if(dwFlag != 0xFFFFFFFF)
    {
        bRet = ::DeleteFile(chFileName);
    }

    return bRet;
}
```

- 移动文件

```
BOOL WINAPI PL_MoveFile(char *chSrcFileName, char *chDesFileName)//移动文件
{
    BOOL bRet = FALSE;
    DWORD dwFlag = ::SetFileAttributes(chSrcFileName, FILE_ATTRIBUTE_NORMAL);//读取文件属性
    /*
    设置文件属性： SetFileAttributes(文件名， 属性值)
    读取文件属性：GetFileAttributes(文件名);
    读取文件属性
    SetFileAttributes(文件名， FILE_ATTRIBUTE_READONLY); // 设定为只读
    SetFileAttributes(文件名， FILE_ATTRIBUTE_HIDDEN );  //设定为隐藏
    SetFileAttributes(文件名， FILE_ATTRIBUTE_SYSTEM);   //设定为系统
    SetFileAttributes(文件名， FILE_ATTRIBUTE_ARCHIVE);  //设定为保存
    SetFileAttributes(文件名， FILE_ATTRIBUTE_NORMAL);   //设定为一般 (取消前四种属性)
    */
    if
        (dwFlag != 0xFFFFFFFF)
    {
        bRet = ::MoveFile(chSrcFileName, chDesFileName);
    }

    return bRet;
}
```

- 读文件

```
HGLOBAL WINAPI PL_ReadDataFromFile(CString strFile)//读文件
{
    HGLOBAL hData = NULL;
    CFile file;
    if(file.Open(strFile, CFile::modeRead))
    {
        int nLen = file.GetLength();
        hData = ::GlobalAlloc(GHND, nLen);//该函数从堆中分配一定数目的字节数。
        if(hData != NULL)
        {
            LPVOID lpData = ::GlobalLock(hData);
            file.ReadHuge(lpData, nLen);//大于65535字节数的数据可以被readhuge读入
            GlobalUnlock(hData);
        }
        file.Close();
    }
    return hData;
}
```

- 写文件

```
BOOL WINAPI PL_WriteDataToFile(CString strFile, HGLOBAL hData)//写文件
{
    BOOL bRet = FALSE;
    CFile file;
    if(file.Open(strFile, CFile::modeCreate | CFile::modeWrite))
    {
        if(hData != NULL)
        {
            int nLen = ::GlobalSize(hData);
            LPVOID lpData = ::GlobalLock(hData);
            file.WriteHuge(lpData, nLen);
            GlobalUnlock(hData);
            bRet = TRUE;
        }
        file.Close();
    }
    return bRet;
}
```

⑤压缩算法

不同压缩算法的使用是本软件的一大亮点，由于我主要负责 LZ77 压缩算法，因此这里仅对 LZ77 算法做简要介绍。

LZ77 算法使用"滑动窗口"的方法，来寻找文件中的相同部分，也就是匹配串。我们先对这里的串做一个说明，它是指一个任意字节的序列，而不仅仅是可以在文本文件中显示出来的那些字节的序列。这里的串强调的是它在文件中的位置，它的长度随着匹配的情况而变化

压缩：从文件的开始到文件结束，一个字节一个字节的向后进行处理。用当前处理字节开始的串，和滑动窗口中的每个串进行匹配，寻找最长的匹配串。如果当前处理字节开始的串在窗口中有匹配串，就先输出一个标志位，表明下面是一个(之间的距离，匹配长度）对，然后输出(之间的距离，匹配长度）对，然后从刚才处理完的串之后的下一个字节，继续处理。如果当前处理字节开始的串在窗口中没有匹配串，就先输出一个标志位，

表明下面是一个没有改动的字节，然后不做改动的输出当前处理字节，然后继续处理当前处理字节的下一个字节。

解压缩：从文件开始到文件结束，每次先读一位标志位，通过这个标志位来判断下面是一个(之间的距离，匹配长度)对，还是一个没有改动的字节。如果是一个（之间的距离，匹配长度）对，就读出固定位数的（之间的距离，匹配长度）对，然后根据对中的信息，将匹配串输出到当前位置。如果是一个没有改动的字节，就读出一个字节，然后输出这个字节。

- LZ77 压缩（我负责）

```
HGLOBAL WINAPI PL_LZ77_Zip(HGLOBAL hUnZip)
{
    int _n = ::GetTickCount();//GetTickCount返回（retrieve）从操作系统启动所经过的毫秒数，它的返回值是DWORD。
    int nSize = ::GlobalSize(hUnZip);//数据大小
    LPBYTE lpData = (LPBYTE)::GlobalLock(hUnZip);
    HGLOBAL hZip = NULL;
    if(nSize > 0 && lpData != NULL)
    {
        const int nMax = 65536;
        BYTE byTemp[nMax + 16];
        CCompressLZ77 cc;
        WORD wFlag1 = 0;
        WORD wFlag2 = 0;
        int nLast = nSize;
        int nReal = 0;
        hZip = ::GlobalAlloc(GHND, nMax+16);
        LPBYTE lpZipData = (LPBYTE)::GlobalLock(hZip);//确保你使用的是物理内存,而不是虚拟内存
        int nPos = 0;
        int nZipPos = 0;
        while(nLast > 0)
        {
            nReal = min(nLast, nMax);
            nLast -= nReal;
            if(nReal == nMax)
            {
                wFlag1 = 0;
            }
            else
            {
                wFlag1 = nReal;
            }
            memcpy(lpZipData + nZipPos, &wFlag1, sizeof(WORD));
            nZipPos += sizeof(WORD);

            int nRetLen = 0;
            nRetLen = cc.Compress(lpData + nPos, nReal, byTemp);
            if(nRetLen == 0)
            {
                wFlag2 = wFlag1;
                memcpy(lpZipData + nZipPos, &wFlag2, sizeof(WORD));
                nZipPos += sizeof(WORD);
                memcpy(lpZipData + nZipPos, lpData + nPos, nReal);
                nZipPos += nReal;
            }
            else
            {
                wFlag2 = (WORD)nRetLen;
                memcpy(lpZipData + nZipPos, &wFlag2, sizeof(WORD));
                nZipPos += sizeof(WORD);
                memcpy(lpZipData + nZipPos, byTemp, nRetLen);
                nZipPos += nRetLen;
            }
            nPos += nReal;
            ::GlobalUnlock(hZip);
            if(nLast > 0)
            {
                hZip = ::GlobalReAlloc(hZip, nZipPos + nMax, 0);
            }
            else
            {
                hZip = ::GlobalReAlloc(hZip, nZipPos, 0);
            }

            lpZipData = (LPBYTE)::GlobalLock(hZip);
        }
        ::GlobalUnlock(hZip);
    }

    TRACE(_T("PL_LZ77_Zip--Time:%d, %d-->%d.\n"),
        GetTickCount() - _n, GlobalSize(hUnZip), GlobalSize(hZip));
    return hZip;
}
```

- LZ77 解压缩（我负责）

```
HGLOBAL WINAPI PL_LZ77_UnZip(HGLOBAL hZip)
{
    int _n = ::GetTickCount();
    int nSize = ::GlobalSize(hZip);
    LPBYTE lpZipData = (LPBYTE)::GlobalLock(hZip);
    HGLOBAL hUnZip = NULL;
    if(nSize > 0 && lpZipData != NULL)
    {
        const int nMax = 65536;
        BYTE byTemp[nMax + 16];
        CCompressLZ77 cc;
        WORD wFlag1 = 0;
        WORD wFlag2 = 0;
        int nLast = nSize;
        int nReal = 0;
        hUnZip = ::GlobalAlloc(GHND, nMax+16);
        LPBYTE lpData = (LPBYTE)::GlobalLock(hUnZip);
        int nPos = 0;
        int nZipPos = 0;
        while(nLast > 0)
        {
            memcpy(&wFlag1, lpZipData + nZipPos, sizeof(WORD));
            nZipPos += sizeof(WORD);
            memcpy(&wFlag2, lpZipData + nZipPos, sizeof(WORD));
            nZipPos += sizeof(WORD);
            nLast -= 2*sizeof(WORD);
            if(wFlag1 == 0)
            {
                nReal = nMax;
            }
            else
            {
                nReal = wFlag1;

            }
            nLast -= wFlag2 ? (wFlag2) : nReal;
            if(wFlag2 == wFlag1)
            {
                memcpy(byTemp, lpZipData + nZipPos, nReal);
                nZipPos += nReal;
            }
            else
            {
//              if(AfxIsValidAddress(lpZipData + nZipPos, nReal))
                {
                    if(!cc.Decompress(byTemp, nReal, lpZipData + nZipPos))
                    {
                        break ;
                    }
                }
                nZipPos += wFlag2;
            }
            memcpy(lpData + nPos, byTemp, nReal);
            nPos += nReal;
            ::GlobalUnlock(hUnZip);
            if(nLast > 0)
            {
                if((::GlobalSize(hUnZip) - nPos) < nMax)
                {
                    hUnZip = ::GlobalReAlloc(hUnZip, nPos + nMax, 0);
                }
            }
            else
            {
                hUnZip = ::GlobalReAlloc(hUnZip, nPos, 0);
            }
            lpData = (LPBYTE)::GlobalLock(hUnZip);

        }
        ::GlobalUnlock(hUnZip);
    }
    TRACE(_T("PL_LZ77_UnZip--Time:%d, %d-->%d.\n"),
        GetTickCount() - _n, GlobalSize(hZip), GlobalSize(hUnZip));
    return hUnZip;
}
```

- LZW 压缩（队友负责）

```
HGLOBAL WINAPI PL_LZW_Zip(HGLOBAL hUnZip)
{
    int _n = ::GetTickCount();
    HGLOBAL hZip = NULL;

    LPBYTE lpData = (LPBYTE)::GlobalLock(hUnZip);
    if(lpData != NULL)
    {
        int nLen = ::GlobalSize(hUnZip);
        C_LZW lz;
        hZip = lz.Encode((char *)lpData, nLen);
        ::GlobalUnlock(hUnZip);
    }
    TRACE(_T("PL_LZW_Zip--Time:%d, %d-->%d.\n"),
        GetTickCount() - _n, GlobalSize(hUnZip), GlobalSize(hZip));

    return hZip;
}
```

- LZW 解压缩（队友负责）

```
HGLOBAL WINAPI PL_LZW_UnZip(HGLOBAL hZip)
{
    int _n = ::GetTickCount();

    HGLOBAL hUnZip = NULL;
    LPBYTE lpData = (LPBYTE)::GlobalLock(hZip);
    if(lpData != NULL)
    {
        int nLen = ::GlobalSize(hZip);
        C_LZW lz;
        hUnZip = lz.Decode((char *)lpData, nLen);
        ::GlobalUnlock(hZip);
    }

    TRACE(_T("PL_LZW_UnZip--Time:%d, %d-->%d.\n"),
        GetTickCount() - _n, GlobalSize(hZip), GlobalSize(hUnZip));

    return hUnZip;
}
```

五、结果测试与分析

- 系统主界面

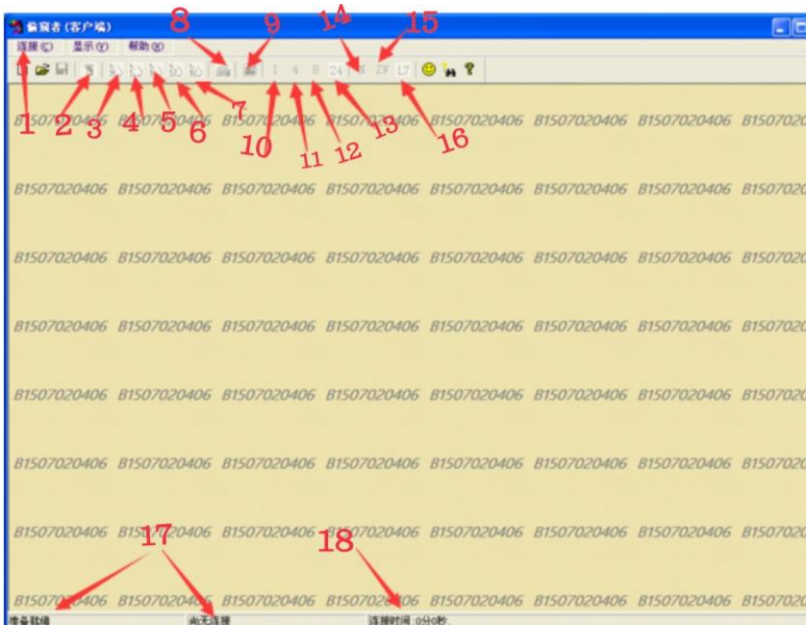

图 11 系统主界面

系统主界面按键解释如表 3 所示：

表 3 标号解释

标号	解释
1	菜单栏。包括“连接”、“显示”和“帮助”。
2	“发送鼠标左键消息”开关
3	发送鼠标右键消息
4	发送鼠标左键双击消息
5	发送鼠标右键双击消息
6	发送键盘锁定消息

7	发送键盘解锁消息
8	刷新屏幕
9	全屏显示
10	接收单色图像
11	接收 16 色图像
12	接收 256 色图像
13	接收真彩图像
14	接收不压缩图像
15	接收 LZW 压缩图像
16	接收 LZ77 压缩图像
17	连接状态显示
18	连接时长显示

- 点击“连接”按钮，执行后如图 12 所示

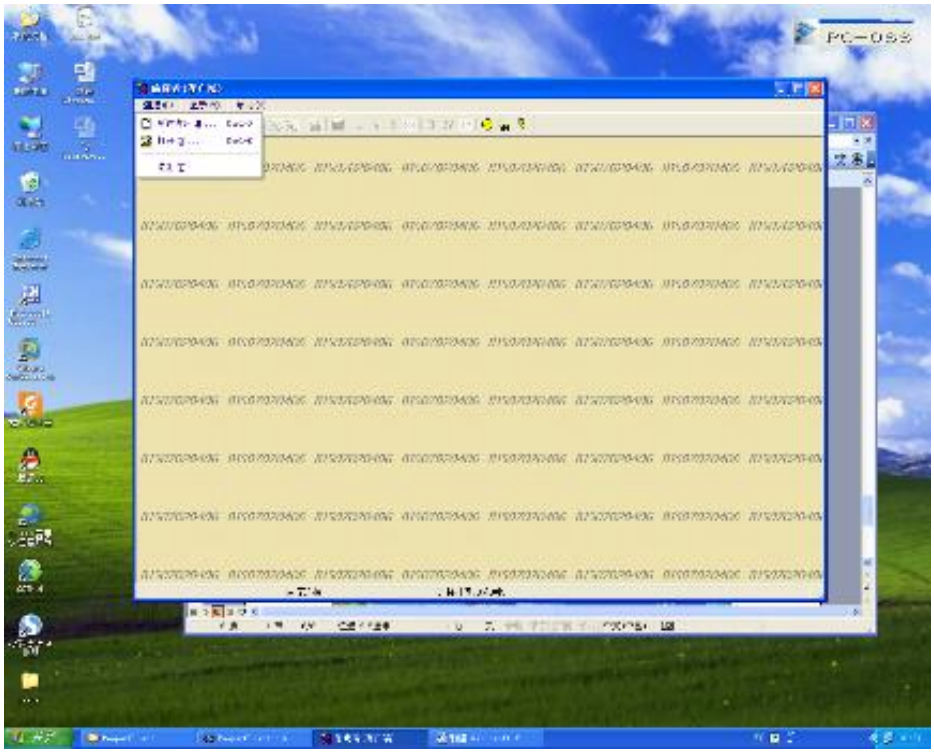

图 12 “连接”菜单

- 连接服务端：

服务器地址（自动获取本机，手动输入被攻击机器）：10.20.148.167

服务器端口（自动获取）：5180

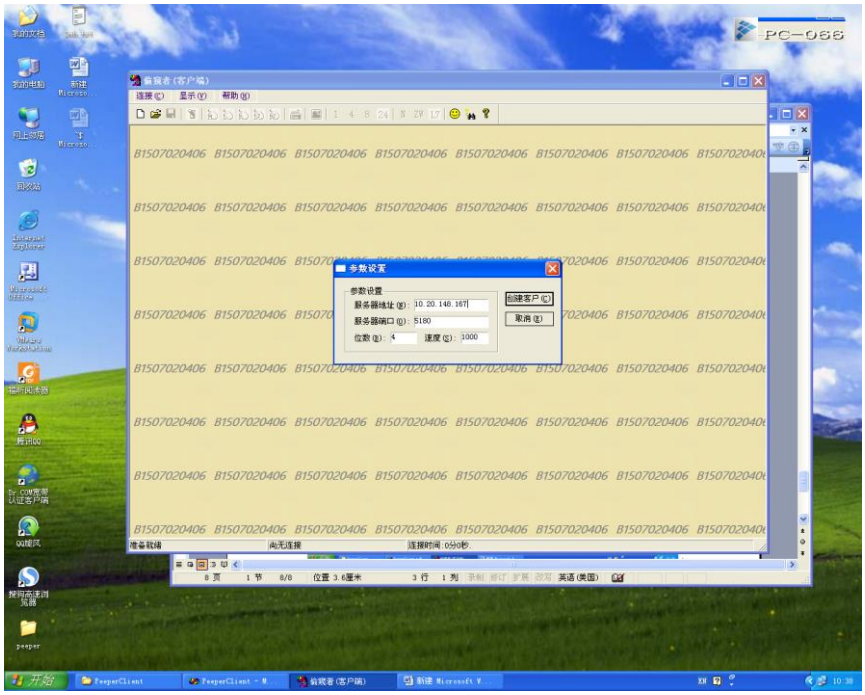

图 13 连接服务器

- 点击“1”按钮，接收单色图像，如图 14 所示：

图 14 单色图象

- 点击“4”按钮，接收 16 色图像，如图 15 所示

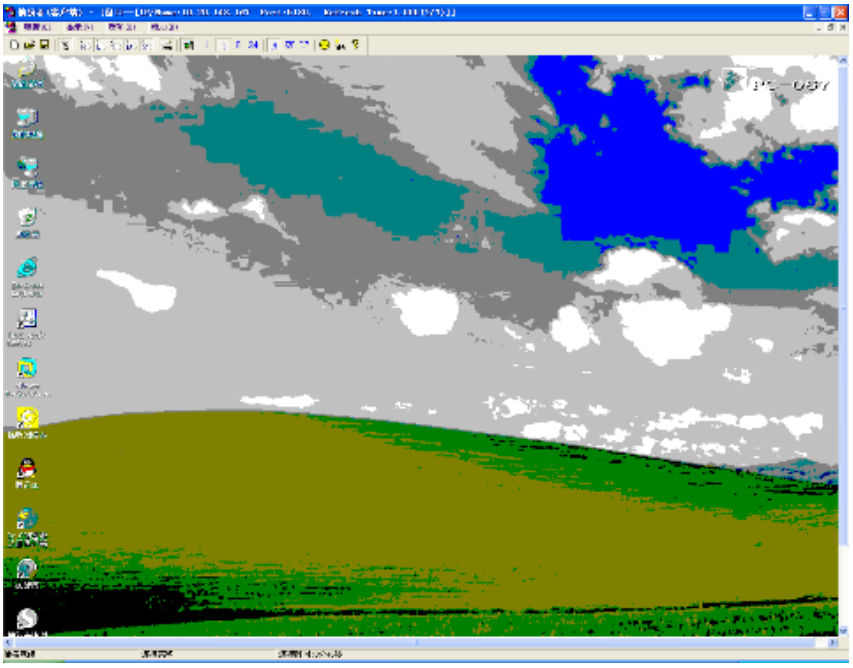

图 15 16 色图像

- 点击“8”按钮，接收 256 色图像，如图 16 所示：

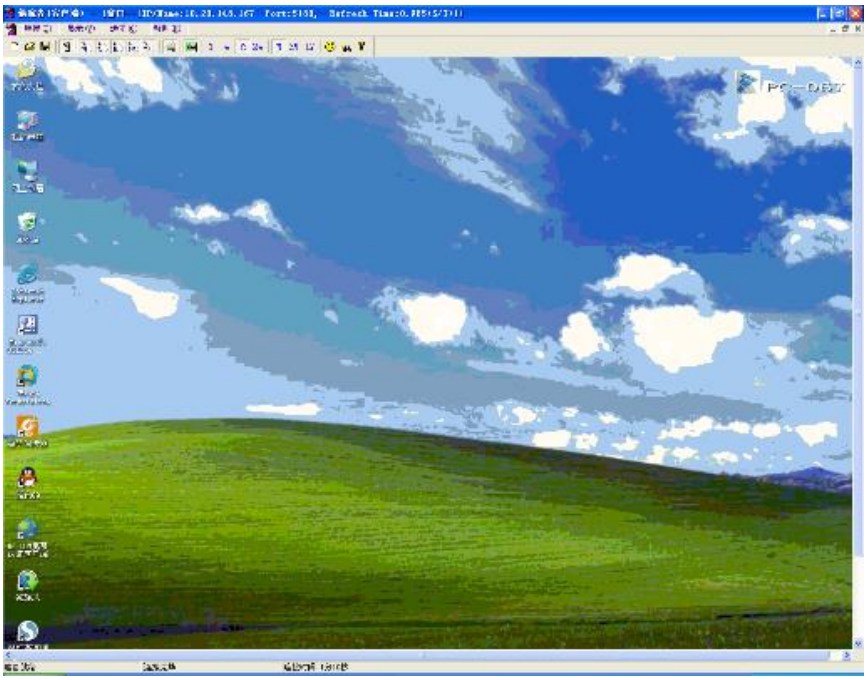

图 16 256 色图像

- 点击“24”按钮，接收真彩图像，如图 17 所示

图 17 真彩图像

- 接收不压缩图像，传输时间约为 1 秒。

图 18 不压缩图像

- 接收 LZW 压缩图像，传输时间约为 5 秒

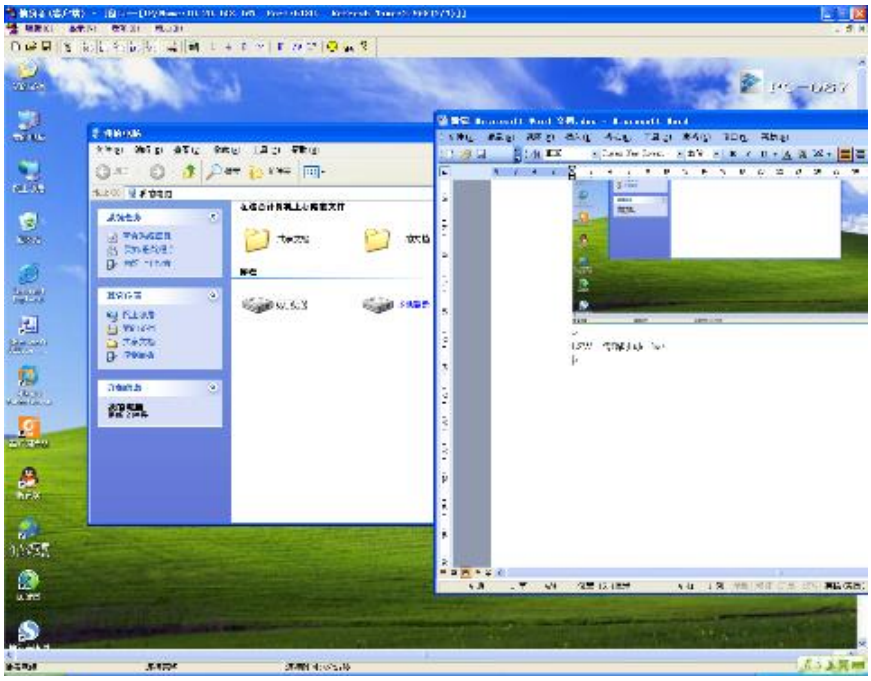

图 19 LZW 压缩图像

- 接收 LZ77 压缩图像，传输时间约为 3 秒。

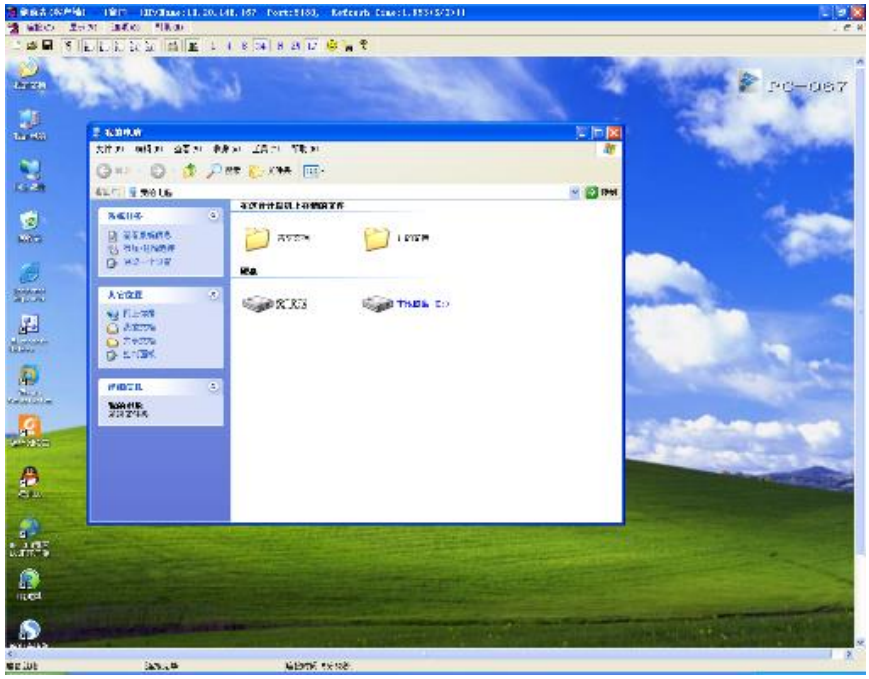

图 20 LZ77 压缩图像

- 点击 “显示”按钮

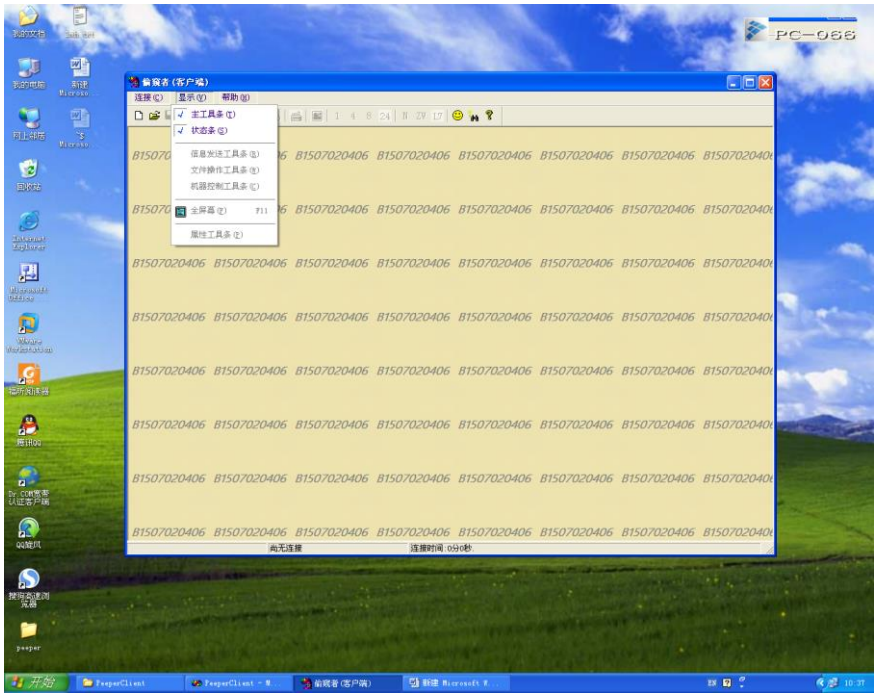

图 21 “显示”菜单栏

- 点击“帮助”按钮：

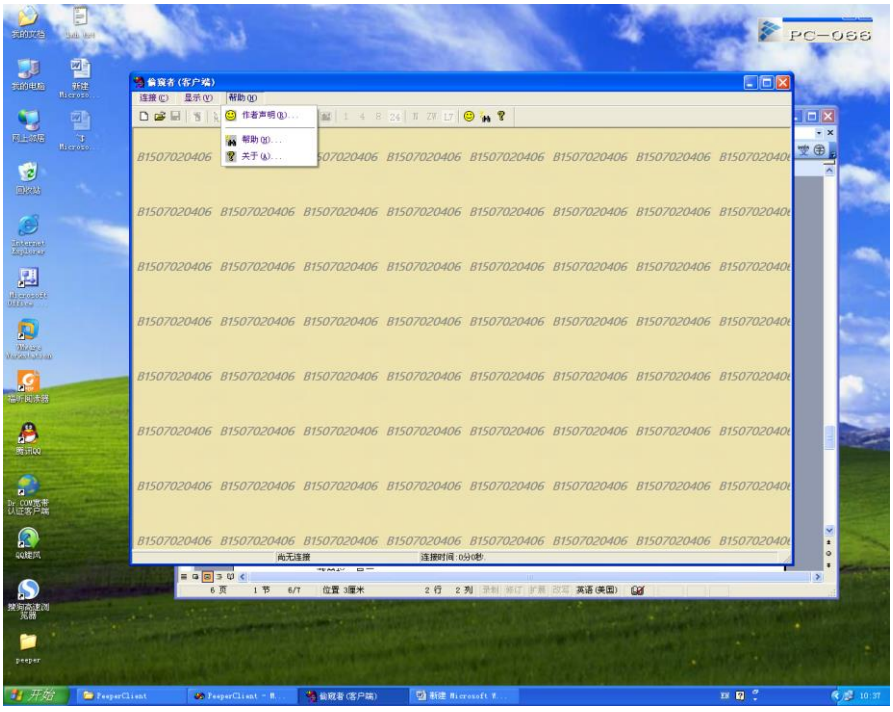

图 22 “帮助”菜单栏

- 发送消息

客户端发送消息：I am your client；

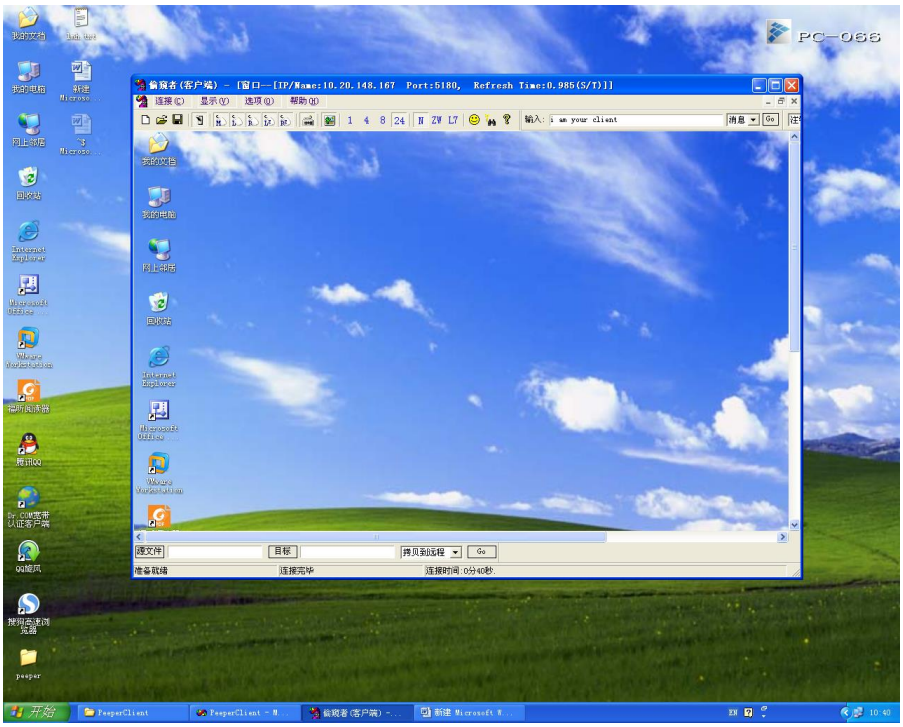

图 23 客户端发送消息窗口

服务端收到消息：

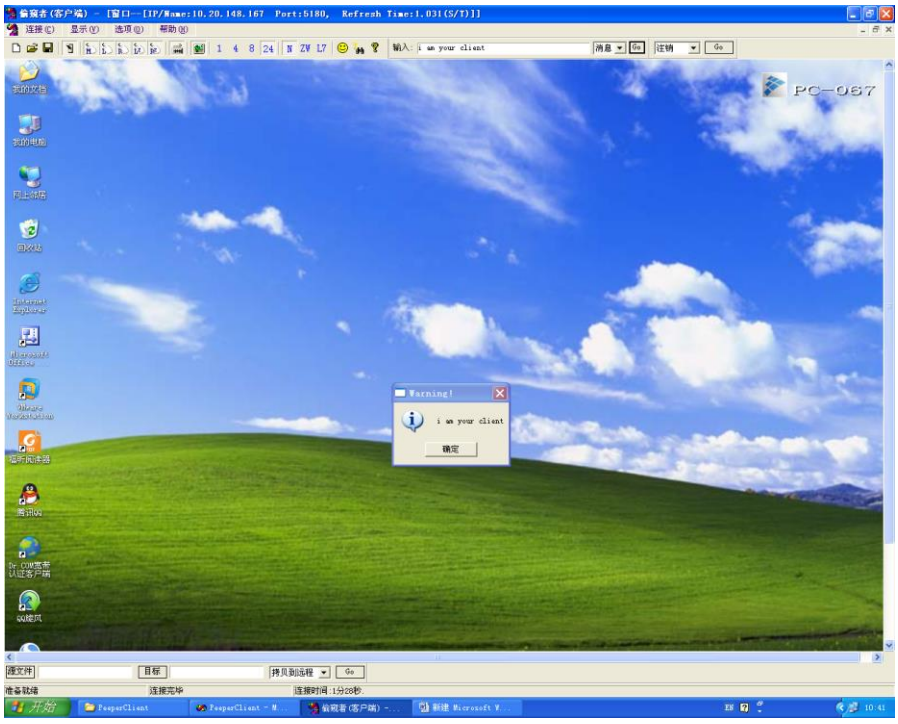

图 24 服务端接收窗口

- 发送命令

客户端发送命令：ipconfig

服务端接收命令；

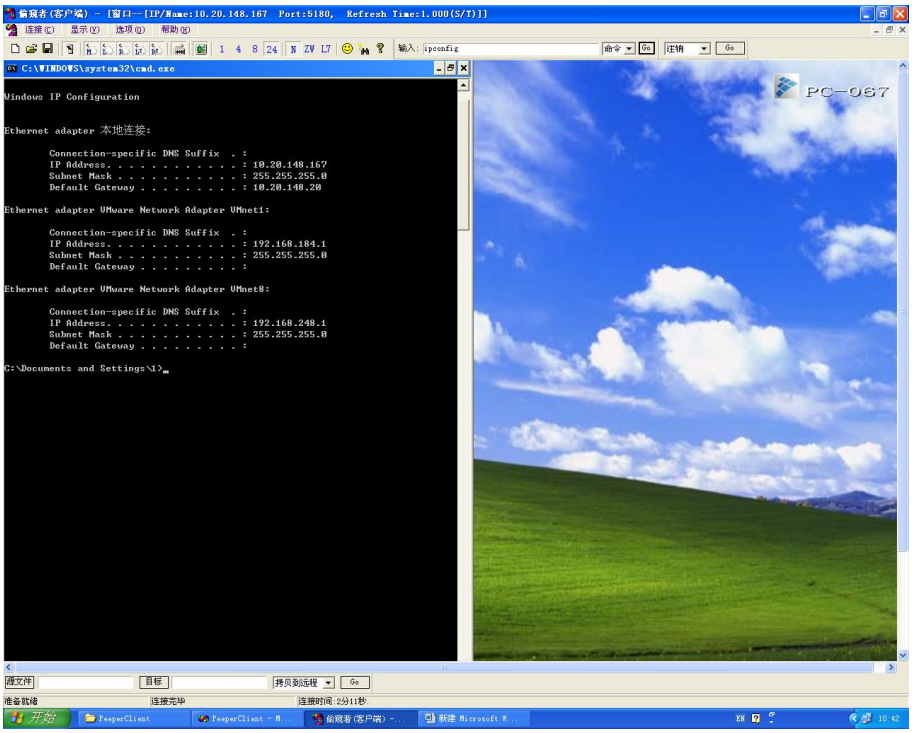

图 25 发送命令窗口

- 发送关机命令，服务端关机

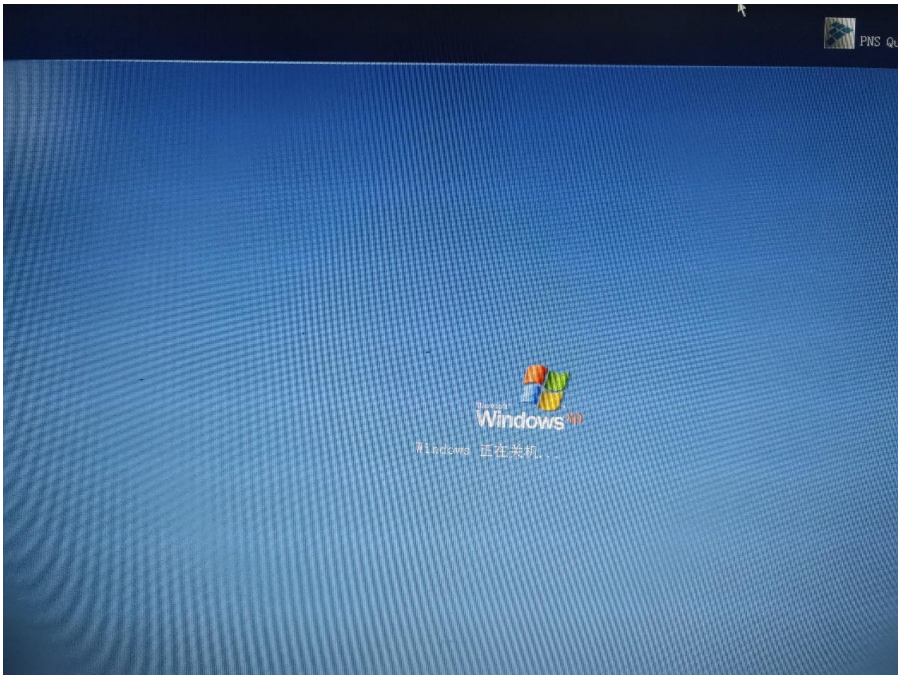

图 26 服务端关机图

- 发送注销命令，服务端注销

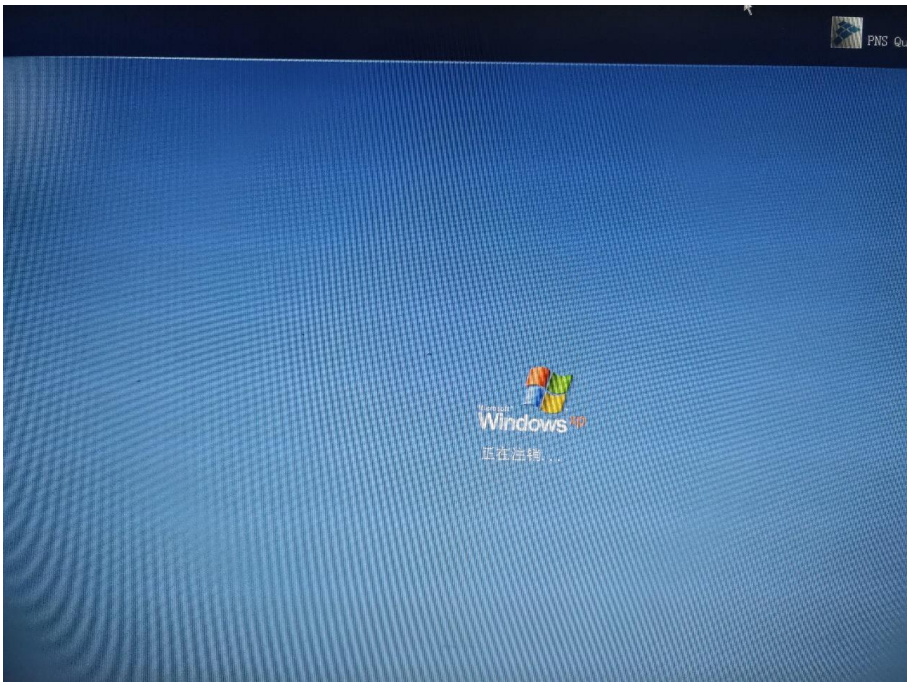

图 27 服务端注销图

- 对服务端可以实现任意操控，比如建立文件：

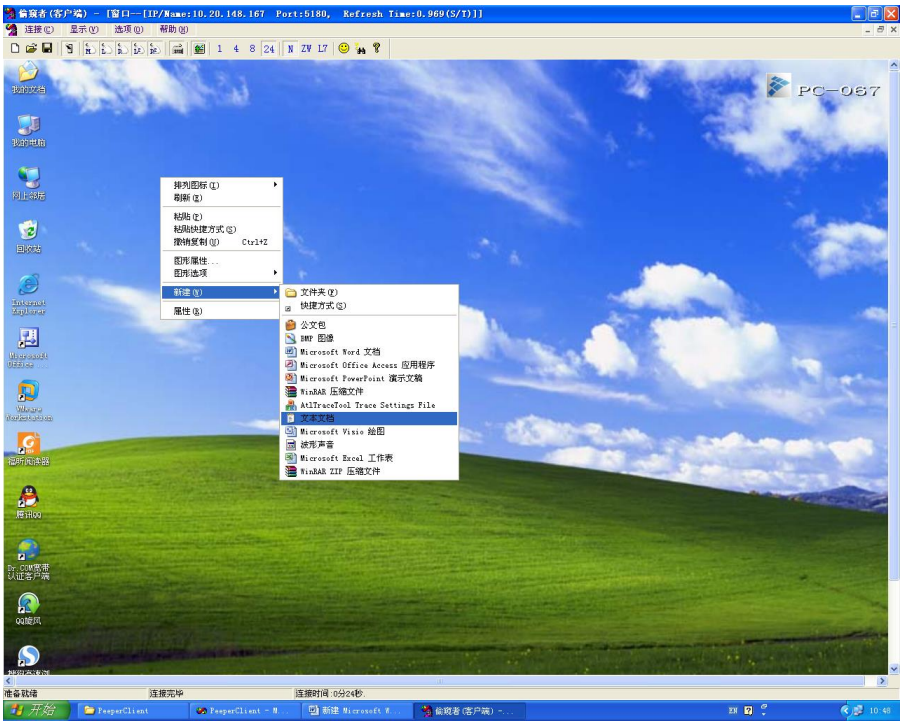

图 28 建立文件

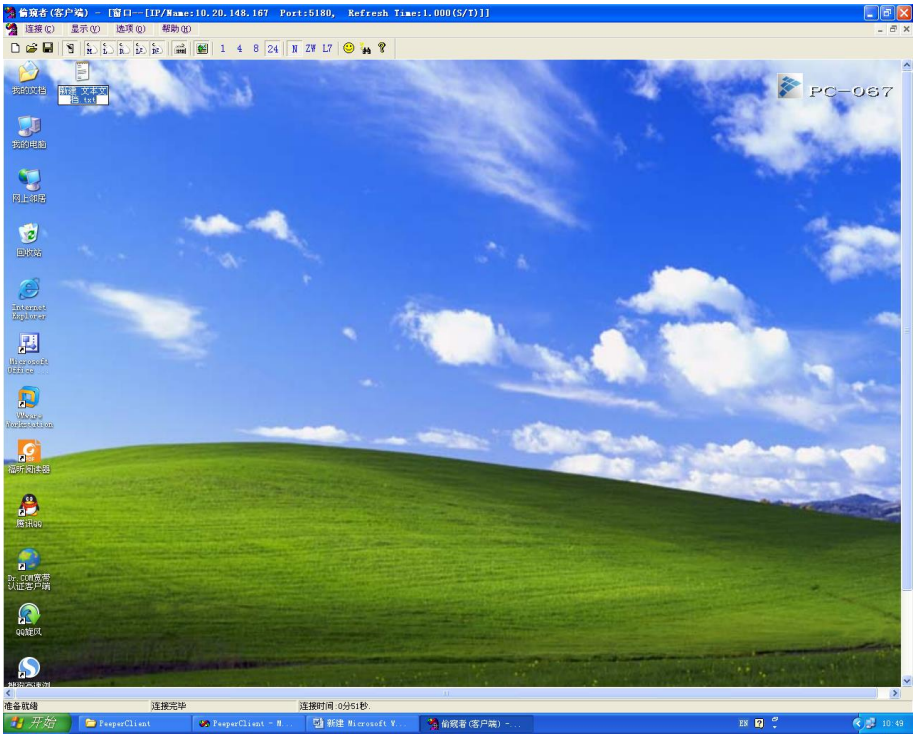

图 29 文件重命名

- 编辑文件

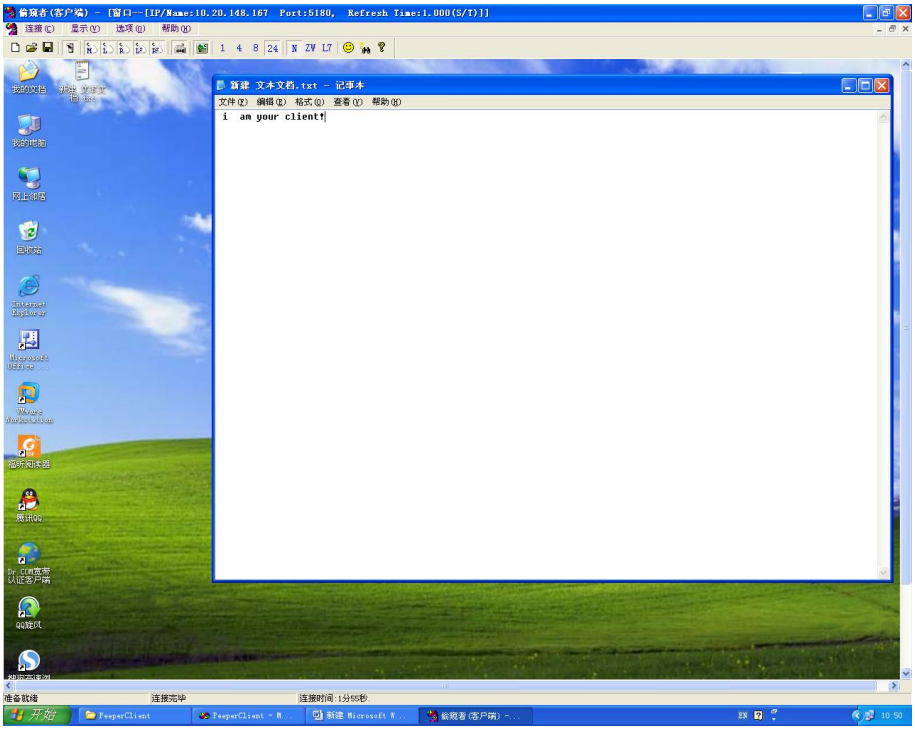

图 30 编辑文件

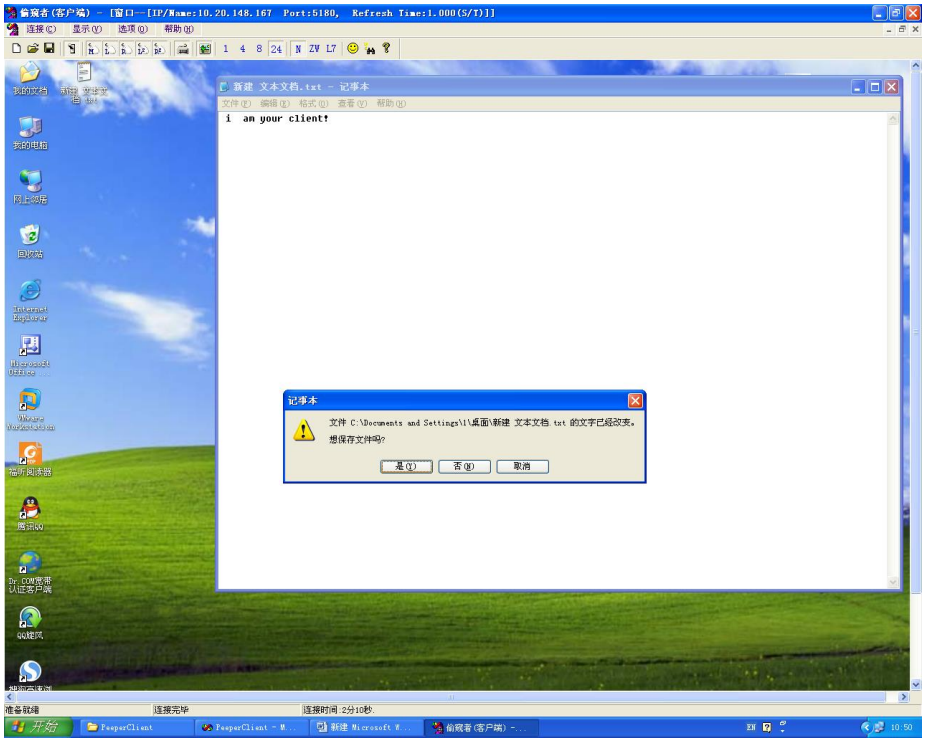

图 31 保存文件

结论：本次测试按照既定的测试方案对系统的功能进行了测试，并跟踪取得了结果，对于给出的测试用例，得到的结果基本都达到预期的要求，也发现了一些细小的问题，再对程序经过修改后已经排除，本次测试成功的完成了既定目标，本系统到达了预定的目标。

六、 存在的问题

1、压缩模块算法选择的问题：在编写图形压缩模块时，我们查找到了不同的方法，因此萌生出提供不同的压缩方法供用户选择的想法。在编写的过程中，我首先研究了 LZ77 算法的原理和步骤，其是通过输出真实字符解决了在窗口中出现没有匹配串的问题，但我们实验发现这个解决方案包含有冗余信息并且还会延误传输的缺点，降低了用户体验。为了进一步解决这个问题，我们又查找到了 LZSS 算法，它的思想是如果匹配串的长度比指针本身的长度长就输出指针，否则就输出真实字符。由于输出的压缩数据流中包含有指针和字符本身，为了区分它们就需要有额外的标志位。最终，我们将不压缩、应用 LZ77 算法压缩和应用 LZSS 算法压缩三种方法进行了对比，发现在人眼的识别能力下，图像传输的质量大致相似，但是时间区别巨大，因此若用户追求画面流畅程度，则可选择不压缩图像；若用户对画面质量要求较高，则可选择 LZSS 算法压缩。

2、实现了木马的“伪隐藏”方法，但没有实现“真隐藏”方法：伪隐藏，就是指程序的进程仍然存在，只不过是让他消失在进程列表里。真隐藏则是让程序彻底的消失，不以一个进程或者服务的方式工作。伪隐藏的方法，是比较容易实现的，只要把木马服务器

端的程序注册为一个服务就可以了，这样，程序就会从任务列表中消失了，因为系统不认为他是一个进程，当按下 Ctrl+Alt+Delete 的时候，也就看不到这个程序。当进程为真隐藏的时候，那么这个木马的服务器部分程序运行之后，就不应该具备一般进程，也不应该具备服务的，也就是说，完全的溶进了系统的内核。我们可以不把他做成一个应用程序，而把他做为一个线程，一个其他应用程序的线程，把自身注入其他应用程序的地址空间。而这个应用程序对于系统来说，是一个绝对安全的程序，这样，就达到了彻底隐藏的效果，这样的结果，导致了查杀黑客程序难度的增加。

3、自启动功能没能完美实现：本程序最遗憾的地方在于没有实现木马程序的自启动功能。我尝试了删除注册表中的键值，这是很多 Windows 程序都采用的方法，也是木马最常用的自启动方法。使用非常方便，但也容易被人发现。但是我发现使用 Windows 自带的程序：msconfig 或注册表编辑器都可以将它轻易的删除，总是出现“写入失败的错误”。对于这个问题，我认为可以在木马程序中加一个时间控件，以便实时监视注册表中自身的启动键值是否存在，一旦发现被删除，则立即重新写入，以保证下次 Windows 启动时自己能被运行。这样木马程序和注册表中的启动键值之间形成了一种互相保护的状态。木马程序未中止，启动键值就无法删除（手工删除后，木马程序又自动添加上了），相反的，不删除启动键值，下次启动 Windows 还会启动木马。我们还查阅了破解方法：首先，以安全模式启动 Windows，这时，Windows 不会加载注册表中的项目，因此木马不会被启动，相互保护的状况也就不攻自破了；然后，就可以删除注册表中的键值和相应的木马程序了。

七、总结

本实验除了题目要求中木马的基本功能以外，还能够同时控制不同的主机、通过发消息或命令恶意干扰远程电脑，并采了不同的位图压缩方法以及提供了多种图像展示方法供用户选择。除此之外，为了增强隐蔽性，木马的服务端是一个无窗口程序。程序采用 C/S 架构，分为两部分：客户端 peerclient.exe 和服务端 peerserver.exe。整个程序在 VC++ 6.0 下编译通过，主要用到了 WindowsSocket 编程的相关知识和一些 Windows API 函数的应用。总体来看，本程序灵活地使用系统提供的控件等工具，编程快捷，实现了对远程计算机的控制操作，达到了良好的操作以及视觉效果。

在进行实验之前，我调研了大量相关木马的原理和功能，比如著名的“冰河木马”、“特洛伊木马”等等，验证性实验是我最初接触木马的实践，当时也碰到了诸多问题，比如如何防御木马的攻击；实验过后，将调研结果与验证性实验结合思考，便对木马有了一定的了解。

在编程的实践中，我也遇到了诸多困难，从语言的回顾到进程编程的学习，再到运用MFC 框架实现界面与功能函数参数的传输等等。 同时，在编写的探索中，我学会了不同的压缩方法，更加印证了学习一个知识，就会发现自己有更多知识不会这句话；在实现基础功能的同时也在激励自己的思考，这个功能的缺陷是什么？是否人性化？有没有更高级的方法？如何改进？由一个基础功能延伸得到的种种想法我认为是最宝贵的，即使因为时间关系没有一一尝试。

实验过后，我反思得到我虽然学会了编制木马，但是学习的目的不是“生产”木马，更重要的是了解原理后如何更好的、科学的防御，不做木马的“文盲”。我还查阅了木马的清除和善后的相关资料以及木马的发展趋势，做到“知己知彼，百战不殆”。