南京邮电大学

课程设计报告

（ 17 /18 学年 第 2 学期）

题 目： 局域网内主机监控系统的设计与实现

专 业	网络工程
学 生 姓 名	刘美含
班 级 学 号	B15070204
指 导 教 师	李养群
指 导 单 位	物联网学院
日 期	2018.06.27

	评分项	成绩
评分细则	遵守机房规章制度（5 分）	
	上机时的表现（5 分）	
	学习态度（5 分）	
	实验准备情况（5 分）	
	实验动手能力（10 分）	
	团队合作精神（5 分）	
	代码分析完整性情况（15 分）	
	代码分析详细程度（10 分）	
	报告书写认真程度（5 分）	
	实验过程详实程度（15 分）	
	文字表达熟练程度（10 分）	
	回答问题准确度（10 分）	
简短评语	教师签名：_____ _____年____月____日	
评分		
备注	评分等级有五种：优秀、良好、中等、及格、不及格	

局域网内主机监控系统

1. 引言

1.1 编写目的

本文的编写目的意在给读者阐述本系统的功能与设计，详细地阐释了项目背景、需求说明、系统架构、系统概要设计、系统详细设计系统实现以及系统的运行和测试。使读者可以充分了解系统的实现原理和方法，用户了解系统的使用方法。

1.2 项目背景

软件名称：监控者

项目提出者：本开发小组　　**项目开发者：**本开发小组

项目管理者：刘美含　　**最终用户：**企业的管理人员

随着计算机技术和信息化建设的快速发展"局域网在各个领域内的应用也日益普及，但局域网的应用发展也带来了网络管理的问题。如：公司职员利用上班时间上网、聊天等，学生在机房实习时玩游戏等等。如能做到“坐在电脑前，就知局域网事”将有效地实现对局域网的监控。

实际上，现有的一些软件如各种木马程序都可以实现对局域网的监控。但由于这些程序功能过于强大，一旦被别人掌握，很容易造成公司机密被盗、信息被毁等恶性事件。因此，开发一个既可以实现局域网的远程监控，又没有破坏力的局域网监控软件是必要的。

我们意在开发一个局域网内远程控制系统，含有监控多个桌面、远程控制、控制多台设备的外部设备等功能。为了方便用户使用，改善用户的使用体验，我们基于 MFC 框架开发了简单实用的用户界面。

1.3 课题内容和要求

- 采用 C/S 模式；
- 同时监控多台主机的桌面，实时抓取桌面。
- 控制多台主机的鼠标、键盘、USB 接口等外部设备；
- 可远程操作目标主机。

1.4 名词解释

C/S 模式：C/S 模式一般是网络应用程序的基本开发模式[1]，其系统结构是能把一个大型的计算机应用系统拆封为多个相互独立的子系统，相互协作来工作实现完整的应用。

MFC：微软基础类库（Microsoft Foundation Classes）是微软公司提供的一个类库，以 C++类的形式封装了 Windows API，并且包含一个应用程序框架，以减少应用程序开发人员的工作量。其中包含大量 Windows 句柄封装类和很多 Windows 的内建控件和组件的封装类[2]。

远程控制：远程控制是在网络上由客户端去远程控制服务器端的技术[3]，所谓的远程是通过网络远程操控远端的电脑，一般都是指在局域网中进行远端操作。

Socket 技术：Socket 套接字为 TCP/IP 协议提供通信服务[4]，例如分别为 TCP 或 UDP 他提供相应的网络接口服务，同时可以为 TCP/IP 模型高层中的标准通信协议或用户自定义的数据通信格式提供网络通信服务。

多线程：多线程是指从软件或者硬件上实现多个线程并发执行的技术[5]。具有多线程能力的计算机因有硬件支持而能够在同一时间执行多于一个线程，进而提升整体处理性能。

TCP/IP 协议： TCP 为两台主机提供可靠的数据通信[6]。它所做的工作包括把应用程序交给它的数据分成合适的小块再交给下面的网络层，确认接收到的分组，设置发送最后确认分组的超时时钟等。由于传输层提供了高可靠性的端到端的通信，因此应用层可以忽略所有这些细节。

UDP：UDP 为应用层提供一种非常简单的服务。它只是把数据报的分组从一台主机发送到另一台主机，但并不保证该数据报一定达到另一端。任何必须的可靠性必须由应用层来提供。

参考文献

[1] 张正伟. 基于 C/S 的远程控制系统的开发[J]. 中国科技纵横, 2009(12):144-144.

[2] 徐璇,姜明新,黄静,徐晶,李敏.基于 MFC 的工程软件界面设计[J].电子设计工程,2011,19(21):11-13.

[3] 刘科. 基于 C/S 的局域网内远程控制系统的 设计与实现[D]. 电子科技大学, 2015.

[4] 赵军伟, 刘勋, 董浩. 基于 TCP 协议的远程监控系统的实现[J]. 中国测试, 2010, 36(1):78-81.

[5] 刘爽,史国友,张远强.基于 TCP/IP 协议和多线程的通信软件的设计与实现[J].计算机工程与设计,2010,31(07):1417-1420+1522.

[6] 王清著.O day 安全：软件漏洞分析技术(第 2 版).电子工业出版社.2011.

2. 需求分析

2.1 现有系统概述

在此次开发之前，我们拥有一个原型系统，仅可实现远程监控功能，无图形界面。这会给用户带来几个问题：①只监控会不方便用户使用，无法实现控制功能也未满足远程控制系统的需求；②监控是通过将对方电脑屏幕截屏然后进行图像传输实现的，图像不进行压缩导致传输过程慢、卡顿明显，用户体验不佳；③原先并没有考虑用户实际的机器性能问题，全部以 256 色图像传输，这在用户机器性能不佳的情况下更加重了传输复旦；④一次只能监控一个电脑，但这并不符合实际的用户需求，我们的目标客户为企业的管理者，需要一次监控多台电脑以达到高校的办公需求；⑤无信息发送与文件传输功能，这导致监控端与被监控端交互困难，不利于提高企业工作效率；⑥无图形界面设计，给用户使用带来不便。因此现有系统将针对以上几点，采用**“原型法”**的开发方法进行改善。

2.2 对新系统的要求

根据 2.1 节对现有系统的描述，我们可以清晰的知道现有系统目前的缺陷。因此我们针对以上缺陷，采用“原型法”的开发方法对新系统增加如下功能：①实现远程控制功能，包括被监控端的鼠标、键盘以及外部设备接口等；②采用不同的压缩算法，供用户根据实际情况选择；③提供 1、4、8、24 位色彩显示方案，供用户根据实际情况以及个人喜好选择；④可同时监控、控制多台电脑；⑤增加消息传输、命令传输以及文件传输功能；⑥增加友好的图形界面，改善客户体验、方便客户使用。

2.3 系统要求

使用远程控制软件进行远程控制，首先要使客户端和服务器端都处在网络中，网络可以是局域网。其次要使被控端和控制端都处在同一种通信协议之下，一般 TCP/IP 协议是多数远程控制软件首选的通讯协议。通过远程控制软件控制的双方电脑需要拥有合法的 IP 地址，而且客户端必须要明确知道服务器端的 IP 地址。为企业设计一个远程控制系统需要满足以下条件：

稳定性：监控系统在企业现代化的管理过程中占据了重要的地位，需要具有一定的稳定性以实现便捷的管理；

易维护性：一个系统的易维护性是系统的一项非常重要的指标，因此系统应易维护以降低企业的运营成本；

安全性：企业的信息数据属于企业机密，因此安全、保密工作是必须第一位考虑。然而数据在网络中传输的过程，因此为防止信息被窃听或截获，必须将数据加密。

易操作性：见解美观易操作的界面对提高企业员工的工作效率至关重要。

2.4 对功能的规定

本系统主要实现的功能有：

- 监控、控制多台主机的桌面；
- 实现远程控制功能，包括被监控端的鼠标、键盘以及外部设备接口等；
- 提供 1、4、8、24 位色彩显示方案，供用户根据实际情况以及个人喜好选择；
- 采用不同的压缩算法实现对图像的压缩，供用户根据实际情况选择；
- 消息传输、命令传输以及文件传输功能，实现监控端与被监控端的交互；
- 图形界面开发，改善客户体验、提高工作效率。

2.5 对性能的规定

用户界面需要简单，友好，方便操作。为了保证主控端对被控端的可靠控制，主控端和被控端采用有连接的 TCP 协议连接。为了到达快速有效地控制被控端，主控端发送给被控端的命令包含的字节数应该尽量的少。其次被控端要正确的解析被控端发送的命令同时也要快速发回主控端需要的信息。

2.6 运行环境规定

2.6.1 硬件配置

- CPU：2.40GHz
- USB：可用

2.6.2 软件配置

- 操作系统：Windows7 标准版或以上版本 64 位操作系统
- 开发工具：Visual C++ 6.0
- 开发框架：Microsoft Foundation Classes

3. 系统架构

在本软件设计中，采用典型的 C/S 结构，由客户端与服务端两部分构成。非对等作用是客户/服务器模式的最显著的特点，即不平等地位的体现是客户相对于服务器的集中体现，服务端提供服务，客户端提供请求。

3.1 系统拓扑结构

使用远程控制软件进行远程控制，首先要使客户端和服务器端都处在网络中。其次要使被控端和控制端都处在同一种通信协议之下，一般 TCP/IP 协议是多数远程控制软件首选的通讯协议。客户端与服务端均位于同一局域网内，其拓扑结构图如图 1 所示：

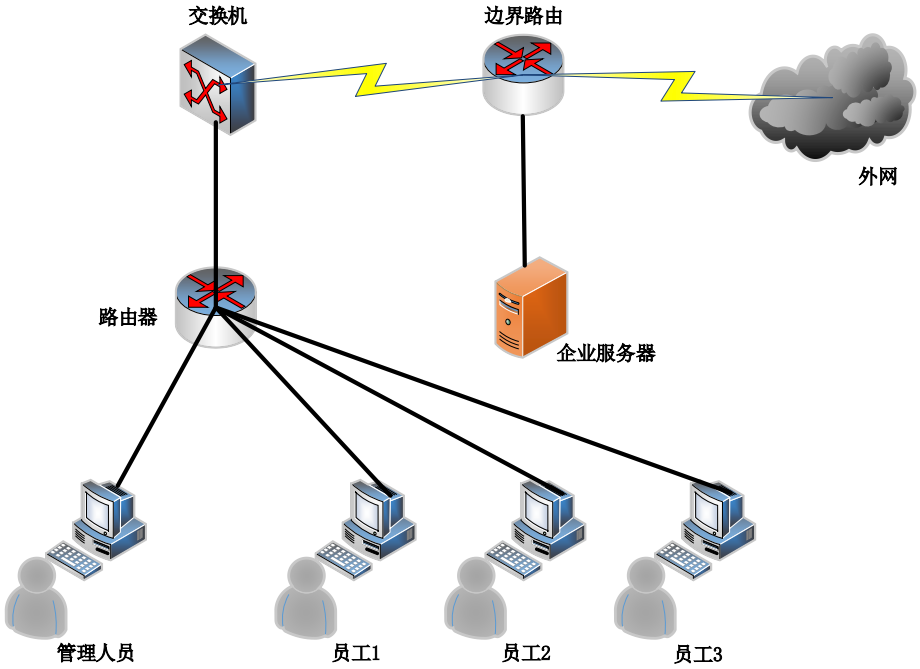


图 1　拓扑结构图

3.2 系统工作过程

客户端提供了极其方便的可视化操作界面，点击不同的按钮和菜单选项可方便快捷的向服务器，只要在对话框 IP 控件中输入被控端的 IP 地址再单击确定按钮就会连接服务器。如点击远程控制菜单选项会弹出远程控制对话框，对话框中分别有不同的控制按钮，点击不同的按钮会产生不同命令发送给服务器，服务器便执行相应操作。服务器运行程序后就等待控制端的连接，一旦客户端连接成功后，服务端一直等待并执行客户端发送命令，实现相关控制。系统工作过程见图 2。

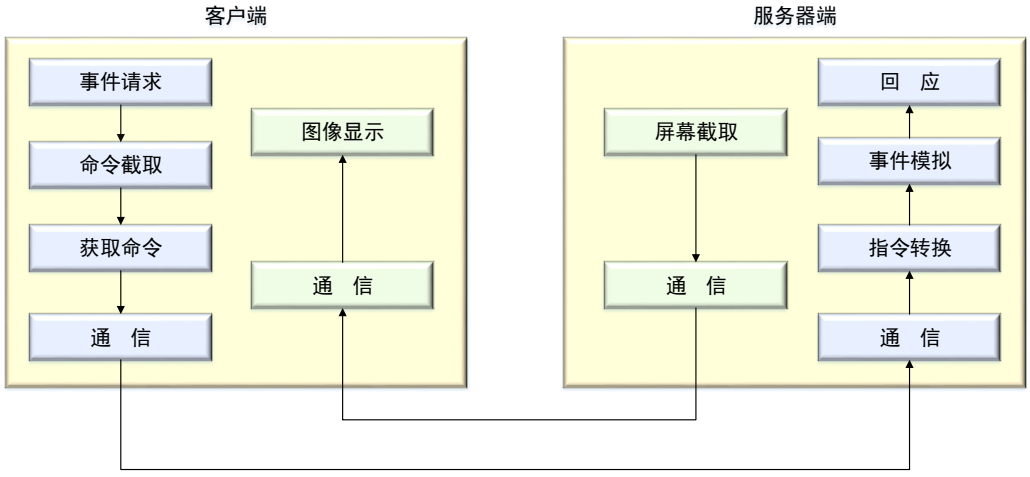


图 2　系统工作过程示意图

其中，系统的消息传递形式可简化为图 3 形式.。首先远程控制端发出控制命令，被控端接受后做出相应的反应并将做出反应后的信息再反馈给控制端。

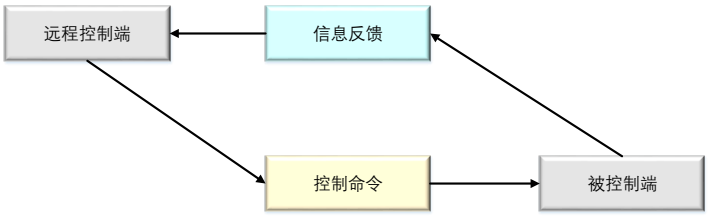


图 3 系统数据流图

3.3 系统框架

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 4 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 5 所示。

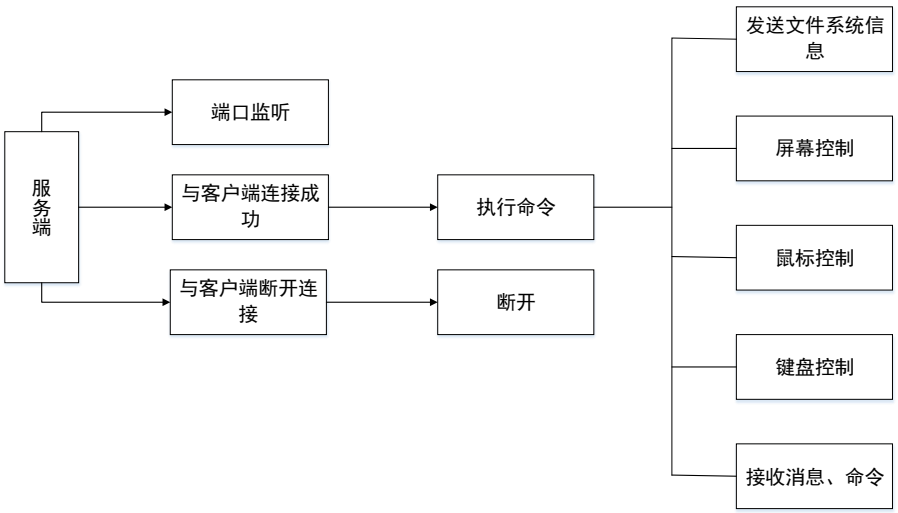


图 4 服务端框架

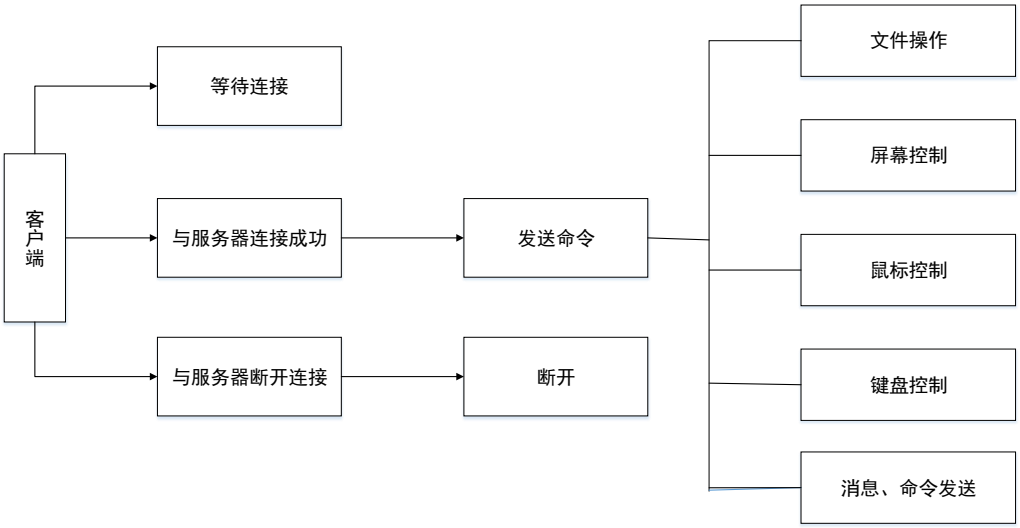


图 5 客户端框架

4. 系统概要设计

4.1 系统概貌

本系统主要由客户端和服务器两部分组成。客户端为用户提供远程控制的操作平台，负责发送命令给被服务器接受服务器发送的信息。服务器是用户想远程控制的目标，服务器在接受到客户端发送的命令以后，服务器会解析命令然后调用相应的函数执行命令。客户端和服务器是通过 Socket 建立地基于 TCP 协议的 P2P 通信。客户端主要功能有：连接多台服务器、获取服务器信息、控制服务器桌面、发送消息和命令、控制服务器设备、文件操作和断开连接；服务器主要功能有启动服务、不断监听、接受连接、捕捉桌面信息、发送桌面信息、接受消息和命令以及关闭服务。系统功能如图 6 所示：

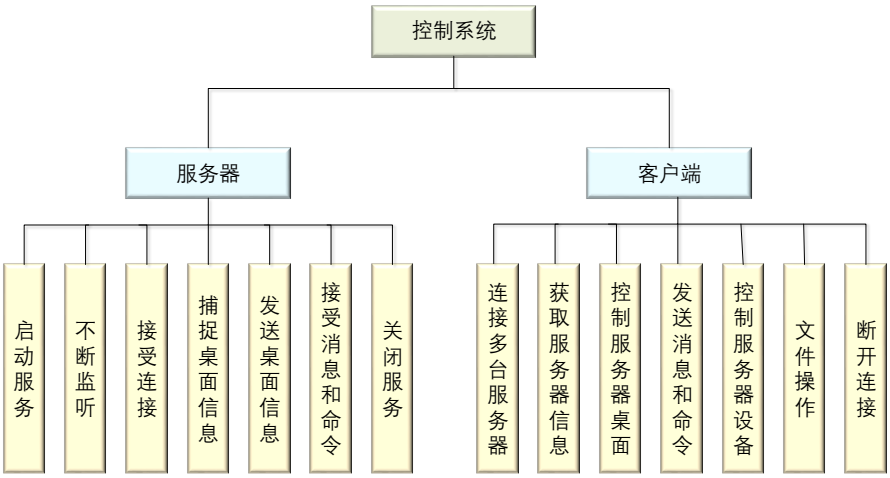


图 6 系统功能图

根据以上系统功能，我们可得用例图如图 7 所示：

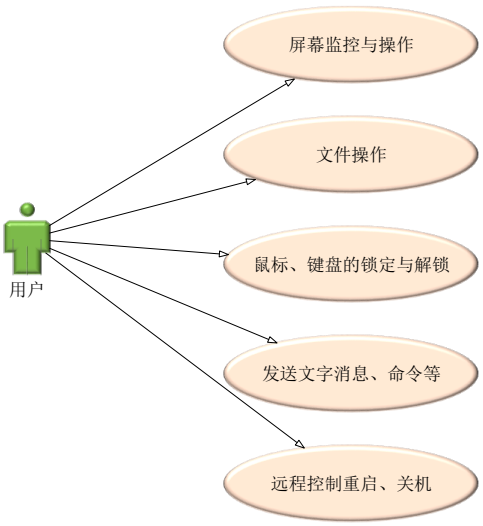


图 7 用例图

4.2 主要功能模块描述

本系统是基于 C/S 架构的远程控制软件，其主要任务是要从系统角度去设计远程控制模式，并开发出相应要求的软件。 该课程设计的远程控制系统需要实现的主要功能有以下几点：

- 实现最基本的远程连接；
- 实现桌面实时监控，获得图像可选择图像色彩与压缩方法；
- 实现远程控屏，包括发送鼠标消息和键盘消息；
- 远程控制对方机器关机、注销、重启；
- 远程禁止对方使用鼠标、键盘以及外部设备接口等；
- 发送消息和命令；
- 进行文件管理、实现文件传输。

根据以上功能，可以将系统功能划分为五个模块，分别为连接建立模块、桌面监控模块、桌面控制模块、消息发送模块和文件管理模块。系统模块功能如图 8 所示：

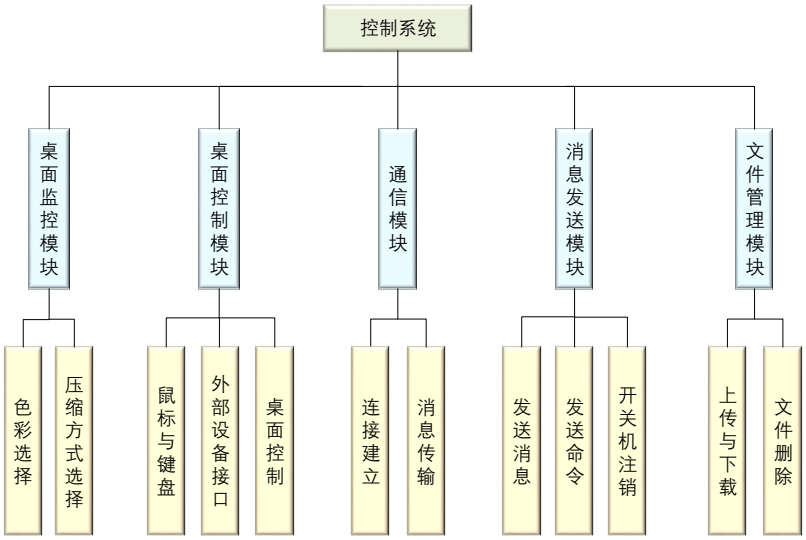


图 8 系统功能模块示意图

4.3 基本设计概念和处理流程

首先，系统初始化，内容包括绑定端口以及设置连接初始值；其次将录入连接信息；然后根据用户的功能需求调节接收画面质量或与服务器交互。系统处理流程图如图 9 所示：

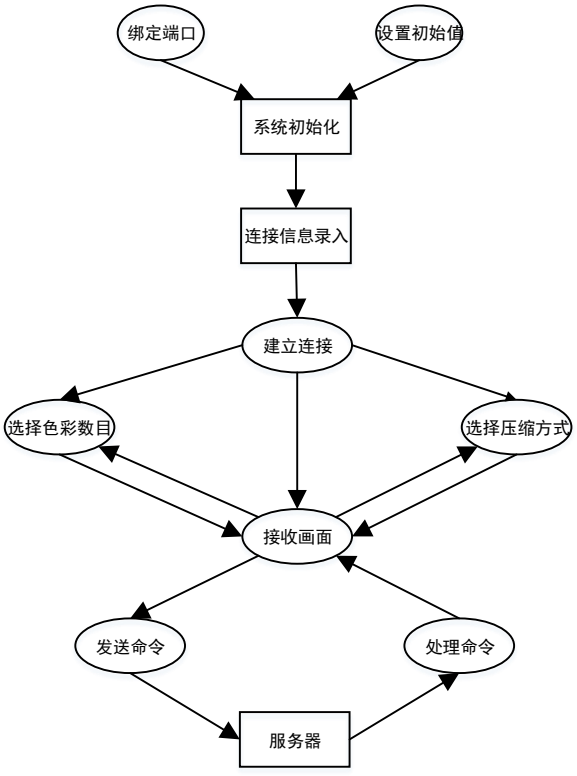


图 9 系统流程图

4.4 系统活动流程描述

用户先新建连接，若连接成功则接受画面，否则重新连接；若仍不成功，则再次重新连接，知道连接成功为止；若一段时间内尚未连接成功，则放弃连接。

用户可随时取消连接。系统活动图如图 10 所示。

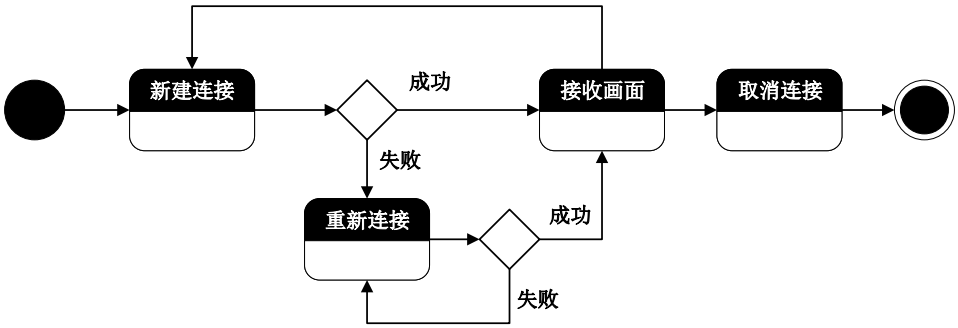


图 10 系统活动图

图 11 为系统时序图，反应了系统活动过程当中用户、界面与服务器的交互关系。用户打开软件进入主界面，点击“新建连接”则进入子窗口，并且尝试与服务器建立连接；连接结果在子窗口展示；用户可以取消连接，则子窗口关闭回到主窗口

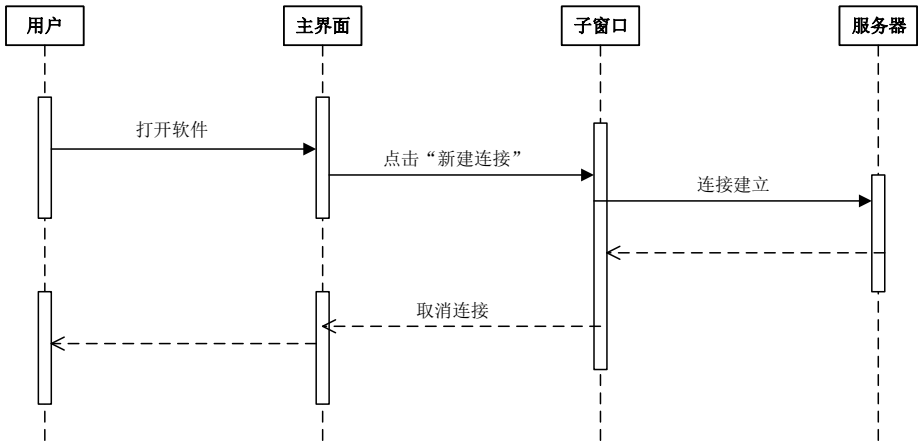


图 11 系统时序图

4.4 主要模块概要设计

系统主要分为通信模块、桌面监控模块、桌面控制模块、消息发送模块、文件管理模块以及图形界面设计。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。客户端要通过发送不同的命令达到控制被控端的目的。当服务器只要成功连接到客户端后，根据客户端界面的不同功能按钮和菜单选项操作，

客户端可以向服务器发送不同的命令。

在功能方面，我主要负责**通信模块**，主要实现客户端与多个服务器的连接与消息传输；此模块为系统的**基础与核心**，若没有此模块，则任何功能都无法实现。在此过程中，我使用了 **Socket** 与**多线程**编程。

4.4.1 Socket 概要设计

系统通信采用**有连接的流方式**。在该通信系统模型中，客户机向服务器程序请求服务，并通过一套协议保证服务能够被提供或者接受，当然，该协议需要客户机和服务器都认可。由于服务器要发送桌面信息、客户端要发送命令消息，因此该协议是**对称的**。服务器通常在一个特定的地址进行监听，等待客户对服务发送请求。当有客户对这个服务器地址提出连接请求时，服务器被“惊醒”，对客户的请求做出适当的反应。下面分别介绍客户端的 Socket 设计与服务器的 Socket 设计。本系统需要客户端和服务器端同时运行程序才能实现。本软件要实现的基本步骤可以简化如下：

步骤 1：被控制端计算机启动服务器程序，打开端口使其处于监听控制端请求的状态，当期监测到控制端发起的连接请求时就对其发起响应，建立连接。

步骤 2：本地计算机运行客户端程序，该程序根据指定的 IP 地址和端口去查找被控制端的服务器程序，点击连接之后，对被控制端程序发送连接请求，等待其相应。

步骤 3：通过 TCP 连接成功之后客户端就可以对服务器发送指令对其进行相关操作，然后服务器端对客户端操作进行响应，如此就可以实现远程控制。

客户端与服务器分别的连接建立步骤如图 12 所示：

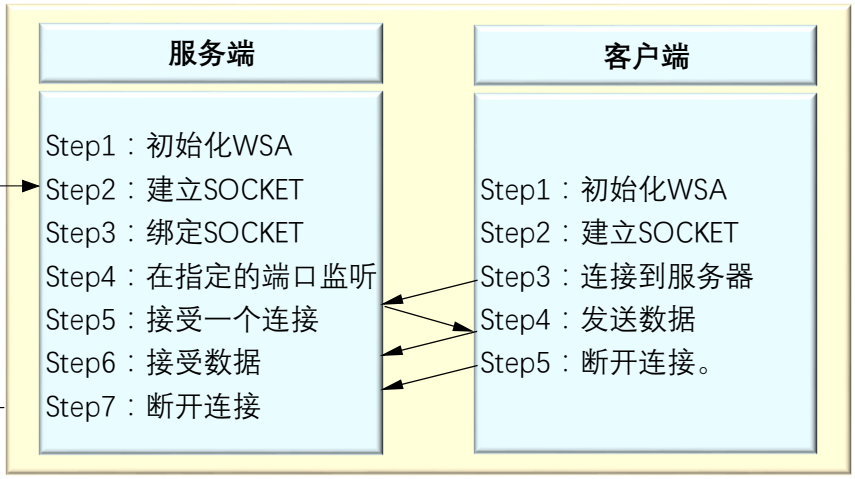


图 12 连接建立步骤

（1）客户端 Socket 设计

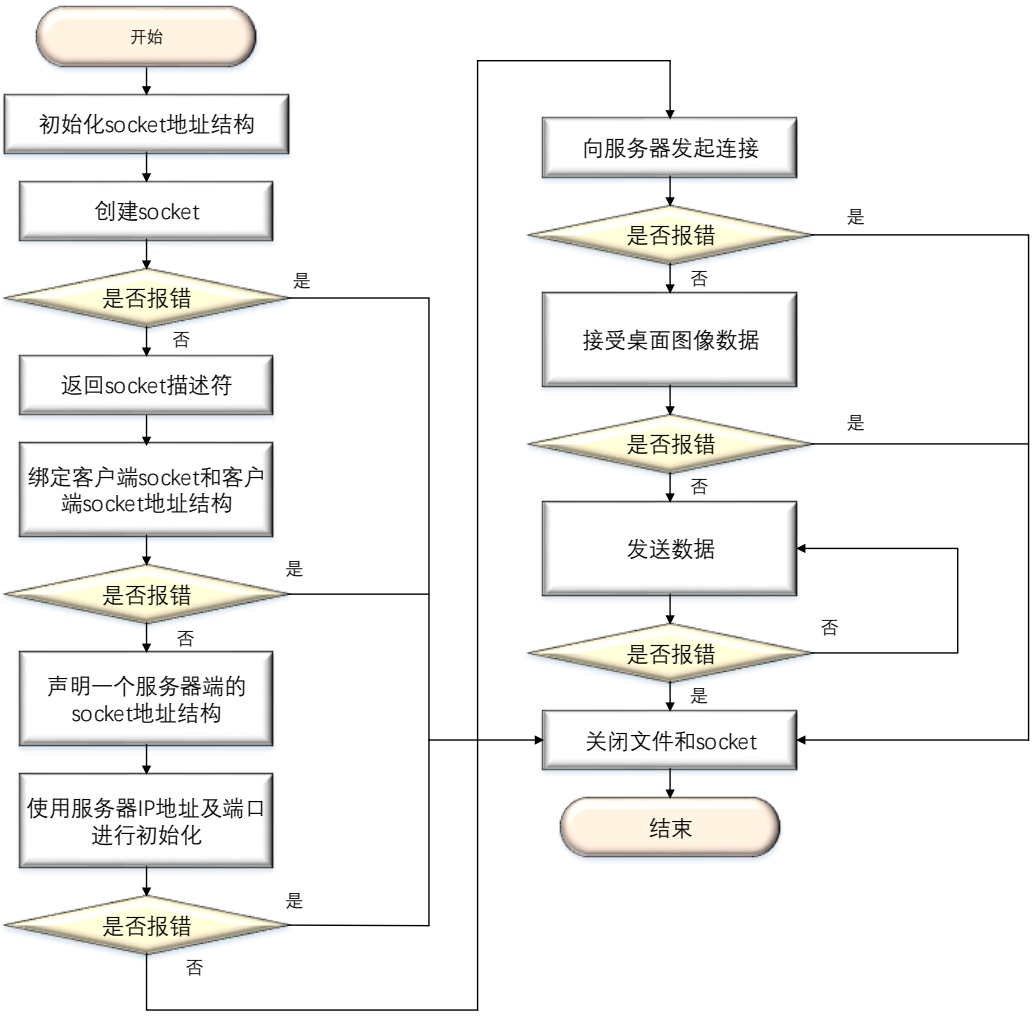


图 8 客户端 Socket 设计

客户端模块执行流程如图 13 所示，具体步骤如下：

步骤 1：初始化 socket 地址结构；

步骤 2：创建 socket；若成功，返回 socket 描述符；否则报错，转步骤 8；

步骤 3：绑定客户端的 socket 和距屋企的 socket 地址结构；若成功，转步骤 4；若失败，转步骤 8；

步骤 4：声明一个服务器端的 socket 地址结构，并用服务器的 IP 地址及端口对其进行初始化；若成功，转步骤 5；若失败，转步骤 8；

步骤 5：向服务器发起连接；若成功，转步骤 6；若失败，转步骤 8；

步骤 6：开始接受桌面图像；若成功，转步骤 7；若失败，转步骤 8；

步骤 7：发送控制命令；若发送失败，则继续尝试发送；若发送成功，则可转步骤 8；

步骤 8：关闭文件和 socket，程序结束。

（2）服务器 Socket 设计

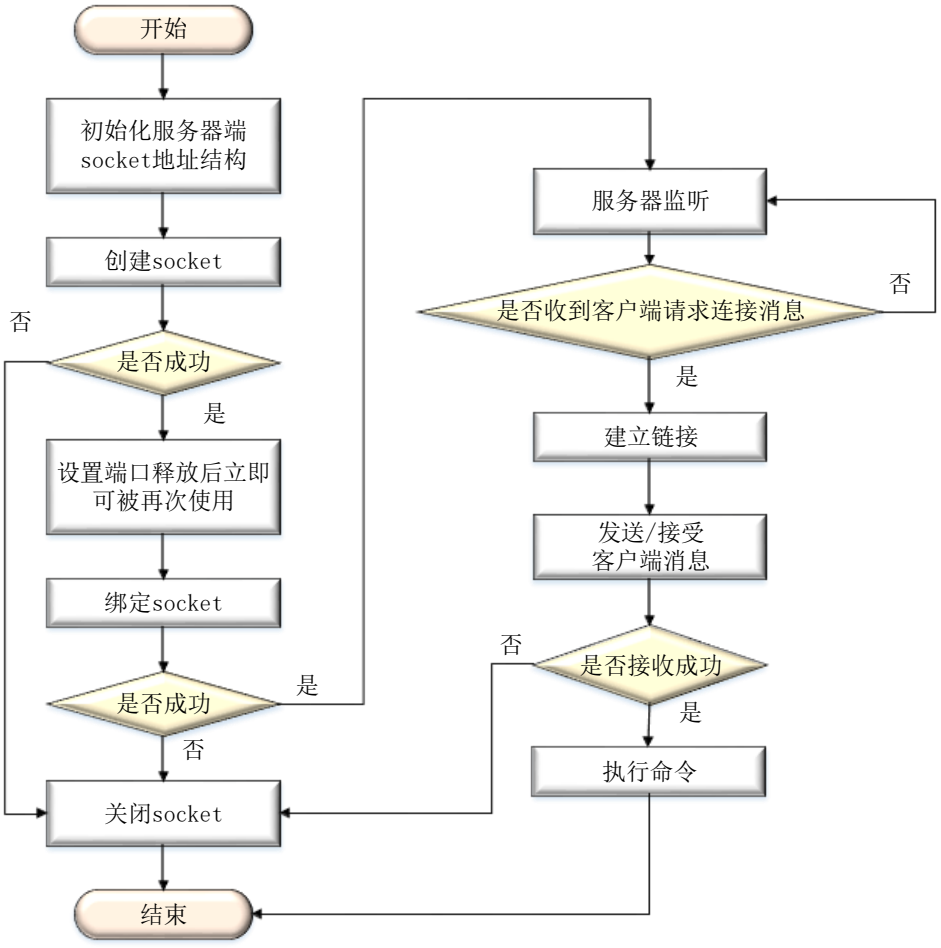


图 9 服务器 Socket 建立

服务器端模块执行流程如图 14 所示，步骤如下：

步骤 1：初始化服务器端的 socket 地址结构；

步骤 2：创建 socket；若成功，设置端口释放后立即就可以被再次使用转步骤 3；若失败，转步骤 6；

步骤 3：绑定 socket；若成功，转步骤 4；若失败，转步骤 6；

步骤 4：服务器开始监听；若收到客户端请求连接消息，则建立链接，转步骤 5；否则，一直监听；

步骤 5：接收客户端消息；若接收成功，则发送桌面数据或者准备接受消息；否则，转步骤 6；

步骤 6：关闭文件和 socket。

4.4.2 多线程概要设计

由于系统要实现同时对多台电脑进行监控和操作，并且线程开销较小，是轻量级的进程，因此需要建立多线程来实现此功能。线程间不需要实现通信与同步。

程序启动后立即建立一主进程，然后客户端向服务器发起连接；若连接成功则建议一个进程负责客户端与此服务器的通信，并将该线程加入主进程。若用户想停止监控，则该线程销毁。线程建立流程图如图 15 所示。

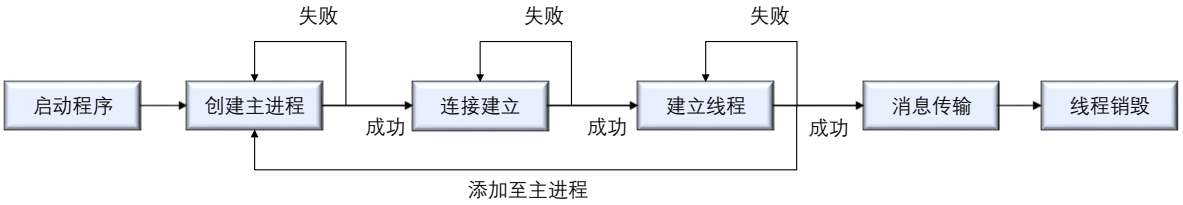


图 10 线程建立流程

4.5 图形界面设计

图形界面设计是一种结合计算机科学、美学、心理学、行为学，及各商业领域需求分析的人机系统工程，强调人—机—环境三者作为一个系统进行总体设计。友好的图形界面可以帮助客户快速找到各种功能，从而提高效率。

主窗口包括状态栏、工具栏和菜单栏。

状态栏为用户显示连接的基本状态，默认打开状态；包括当前状态是否可以建立连接、连接建立情况与连接建立时间。

工具栏为用户展示常用的功能，默认打开状态；包括新建连接、鼠标控制、键盘控制、压缩选择、色彩展示、帮助等。

菜单栏则包含所用功能，分为连接、显示和帮助。连接包括连接的新建、打

开和退出；显示可以决定工具栏、状态栏、文件控制栏、消息发送栏的显示与否，以及软件窗口是否全屏；帮助包括作者声明、软件使用帮助和关于软件的信息。

系统图形界面设计框架如图 16 所示：

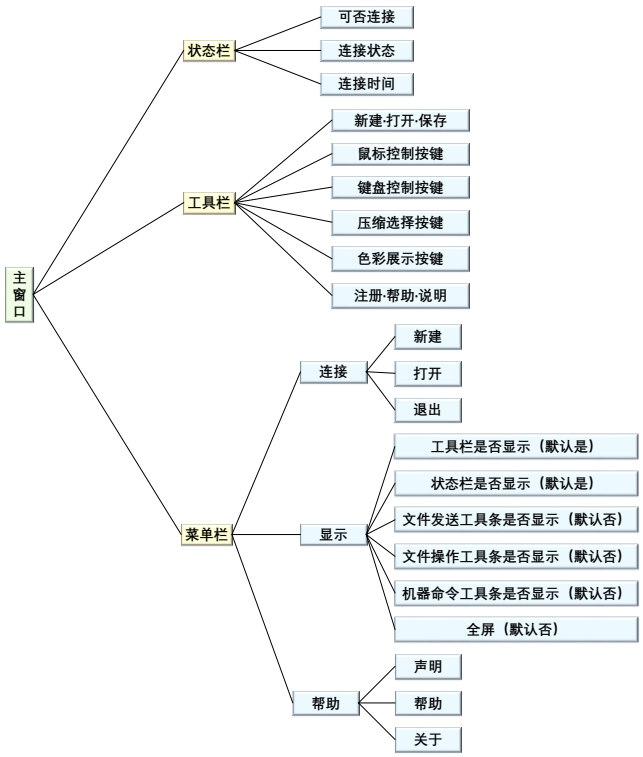


图 11 图形界面设计

文档的一般建立过程如图 17 所示：

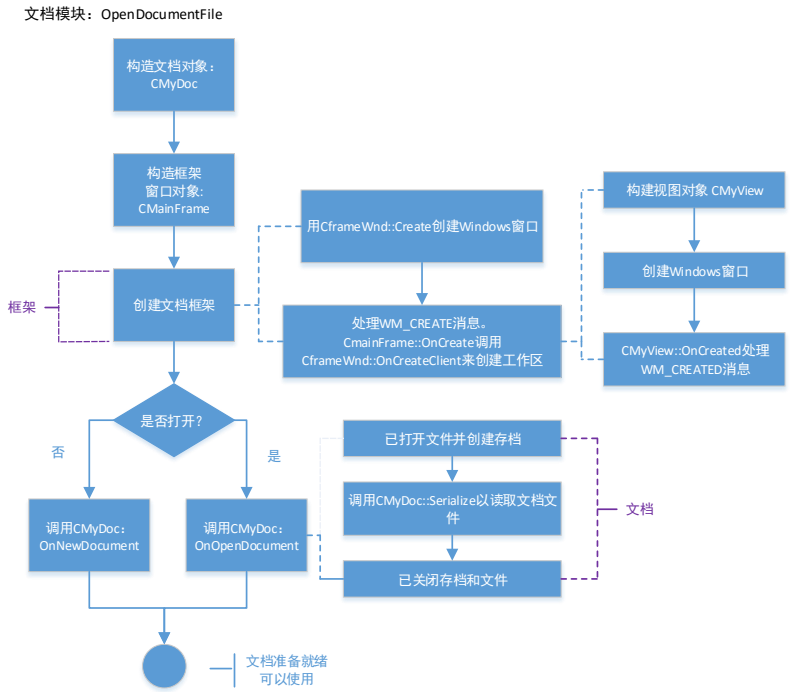


图 12 文档的创建流程

首先构建文档对象，然后创建文档框架并构建视图对象；若打开文档，则调用 CMyDoc::OnOpenDocument(),此时是文档准备就绪可以使用。

4.6 系统出错处理设计

4.3.1 出错信息

若用户输入 IP 地址后连接失败将跳出“连接失败，正在重新连接”的信息。

4.3.2 补救措施

首先检查用户输入的 IP 地址是否合理；若合理且连接失败则再次尝试连接，若尝试 16 次后仍未成功，则停止尝试连接。

5. 系统详细设计

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。

我负责 **Socket 设计、多线程建立**以及**图形界面设计与实现**。 此部分将针对

我负责的部分展开详细描述。

5.1 主要常量定义

由于系统工程量大，因此需要定义常量以便团队开发。主要常量定义如表 2 所示（由于定义量较多，因此以 Resources-String Table 下“字串表”的截图形式给出）：

表格 1 主要常量定义

ID	值	标题
IDP_SOCKETS_INIT_FAILE	104	Windows sockets initialization failed.
CG_ID_VIEW_DLGBAR	105	Show or hide 运行窗口\nToggle 运行窗口
ID_STATUS_CONN_INFO	106	尚无连接
ID_STATUS_TIME	107	连接时间:%d分%d秒.
IDR_MAINFRAME	128	监控者(客户端)
IDR_PEEPERTYPE	129	\n监控者\n监控者\n选择配置文件(*.ppe)\n.ppe\nPEEPERf.ID\npeeperdoc
IDR_TOOLBAR_DIBVIEW	139	显示工具条
ID_HELP_HELP	32783	帮助信息\n帮助
ID_HELP_REGISTER	32784	注册\n注册
ID_FILE_CLOSESERVER	32787	关闭远程服务程序\n关闭服务
ID_VIEW_SEND	32788	是否显示发送信息/命令的工具条\n发送信息/命令工具条
ID_FILE_MANAGE	32789	是否显示文件操作工具条\n文件操作工具条
ID_VIEW_COMPUTER	32790	是否显示机器控制的工具条\n机器控制工具条
ID_OPTION_MOUSE_MOV	32794	是否发送鼠标移动的消息\n发送/禁止鼠标移动消息
ID_OPTION_MOUSE_LBU	32795	是否发送鼠标左键消息\n发送/禁止鼠标左键消息
ID_OPTION_MOUSE_RBU	32796	是否发送鼠标右键消息\n发送/禁止鼠标右键消息
ID_OPTION_KEY	32797	是否发送按键消息\n发送/禁止按键消息
ID_OPTION_MOUSE_LDBI	32798	是否发送鼠标左键双击消息\n发送/禁止鼠标左键双击消息
ID_OPTION_MOUSE_RDB	32799	是否发送鼠标右键双击消息\n发送/禁止鼠标右键双击消息
ID_OPTION_DIB	32800	是否更新桌面图像\n更新/禁止桌面图像
ID_TOOL_DIBVIEW_FULL	32808	全屏幕显示\n全屏幕
ID_TOOL_PROPERTY	32809	是否显示属性工具条\n属性工具条
ID_OPTION_1BITS	32812	获取单色图像\n单色图像
ID_OPTION_4BITS	32813	获取16色图像\n16色图像
ID_OPTION_8BITS	32814	获取256色图像\n256色图像
ID_OPTION_24BITS	32815	获取真彩色图像\n真彩色图像
ID_OPTION_NOZIP	32817	不压缩图像\n不压缩
ID_OPTION_LZ77	32818	使用LZ77压缩算法进行压缩\nLZ77算法压缩
ID_OPTION_LZW	32819	使用LZW算法进行压缩\nLZW算法压缩(推荐)
ID_OPTION_JPEG	32820	转换成JPEG图像\nJPEG图像
ID_OPTION_LZSS	32821	使用LZSS算法进行压缩\nLZSS算法压缩
ID_OPTION_ARI	32822	使用ARI算法进行压缩\nARI算法压缩
ID_OPTION_PAUSE	32826	暂停发送消息\n暂停/恢复
AFX_IDS_APP_TITLE	57344	监控系统(客户端)
AFX_IDS_IDLEMESSAGE	57345	准备就绪
ID_FILE_NEW	57600	创建一个新的客户连接\n创建客户
ID_FILE_OPEN	57601	打开一个保存的设置文件\n打开文件
ID_FILE_CLOSE	57602	关闭当前的客户窗口\n关闭客户
ID_FILE_SAVE	57603	保存当前的设置\n保存设置
ID_FILE_SAVE_AS	57604	另存当前的设置\n另存设置
ID_APP_ABOUT	57664	关于\n关于
ID_APP_EXIT	57665	退出\n退出
ID_VIEW_TOOLBAR	59392	是否显示主工具条\n主工具条
ID_VIEW_STATUS_BAR	59393	是否显示状态条\n状态条
IDS_LISTEN	59394	声明:\r\n　　(1) 作者拥有本软件的版权。\r\n　　(2) 本软件仅供大家网络管理使用，不可非法使用，用户个人

5.2 主要模块实现

5.2.1 连接前的准备工作

(1)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;//记录主机的信息指针
    ZeroMemory(chTemp, 256);
    int nRet = ::gethostname(chTemp, 256);// 接收缓冲区name,其长度必须为len字节或是更长,存获得的主机名，返回本地主机的标准主机名
    if(nRet == 0)//若接受成功
    {
        if(AfxIsValidAddress(chName, strlen(chTemp)))// AfxIsValidAddress判断指针地址是否有效
        {
            strcpy(chName, chTemp);
            bRet = TRUE;
        }
        if(AfxIsValidAddress(chIP, 16))
        {
            pEnt = ::gethostbyname(chTemp);//用域名或主机名获取IP地址
            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;
}
```

5.2.2 连接的建立与取消

(1)连接建立

```
HRESULT CPeeperThread::OnMsgConnect(WPARAM, LPARAM)//一般在消息函数中带这两个类型的参数，通常用来存储窗口消息的参数。
//LOWORD(lParam)是客户区的宽，HIWORD(lParam)是客户区的高
{
    BOOL bRet = FALSE;
    do
    {
        bRet = ::PL_InitSocket();//初始化SOCKET
        if(!bRet)//不成功持续获得
        {
            break ;
        }

        if(!m_pPeeperWnd)
        {
            bRet = FALSE;
            break ;
        }
        int nRet = 0;
        sockaddr_in addr;
        hostent* pEnt = NULL;

        m_pPeeperWnd->m_sckClient[0] = ::socket(AF_INET, SOCK_STREAM, 0);//生成一个TCP的socket
        m_pPeeperWnd->m_sckClient[1] = ::socket(AF_INET, SOCK_STREAM, 0);//生成一个TCP的socket
        TRACE(_T("Client Socket 0 and 1:%d, %d.\n"),
            m_pPeeperWnd->m_sckClient[0], m_pPeeperWnd->m_sckClient[1]);//TRACE调试web服务器连接的HTTP方式
        if(m_pPeeperWnd->m_sckClient[0] == INVALID_SOCKET ||
            m_pPeeperWnd->m_sckClient[1] == INVALID_SOCKET)
        {
            bRet = FALSE;
            break ;
        }

        pEnt = ::gethostbyname(m_pPeeperWnd->m_strIP);//得到主机名
```

```
        if(!pEnt)
        {
            bRet = FALSE;
            break ;
        }
        addr.sin_family = AF_INET;
        addr.sin_port = ::htons((u_short)(m_pPeeperWnd->m_uPort));//将整型变量从主机字节顺序转变成网络字节顺序
        //将用户输入字符串拆分IP地址
        addr.sin_addr.S_un.S_un_b.s_b1 = pEnt->h_addr_list[0][0];
        addr.sin_addr.S_un.S_un_b.s_b2 = pEnt->h_addr_list[0][1];
        addr.sin_addr.S_un.S_un_b.s_b3 = pEnt->h_addr_list[0][2];
        addr.sin_addr.S_un.S_un_b.s_b4 = pEnt->h_addr_list[0][3];

        bRet = FALSE; // default
        for(int i = 0; i < 3; i ++)
        {
            int nRet1 = ::connect(m_pPeeperWnd->m_sckClient[0], (sockaddr*)&addr, sizeof(addr));//连接
            if(nRet1 != SOCKET_ERROR)//连接成功
            {
                BYTE chData[5];
                ZeroMemory(chData, 5);
                ::PL_ReadSocketData(m_pPeeperWnd->m_sckClient[0], chData, 5, NULL);
            }
            else//连接失败，报错
            {
                TRACE(_T("Socket Error Code = %d.\n"), ::WSAGetLastError());
            }
            TRACE(_T("Client[0] is: %s.\n"), (nRet1 == 0)?_T("Ok"):_T("Failed."));
            int nRet2 = ::connect(m_pPeeperWnd->m_sckClient[1],
                (sockaddr*)&addr, sizeof(addr));
            if(nRet2 == SOCKET_ERROR)
            {
                TRACE( T("Socket Error Code = %d.\n"), ::WSAGetLastError());
            }

            TRACE(_T("Client[1] is: %s.\n"), (nRet2 == 0)?_T("Ok"):_T("Failed."));
            if(nRet1 == SOCKET_ERROR || nRet2 == SOCKET_ERROR)//若连接失败，则过一定时间再次连接
            {
                Sleep(2000);
                continue ;
            }

            char chConnectInfo[512];
            ZeroMemory(chConnectInfo, 512);
            int nRet = ::PL_ReadSocketData(m_pPeeperWnd->m_sckClient[0], (BYTE *)chConnectInfo, 512, NULL);//得到SOCKET中的数据
            m_pPeeperWnd->SetConnectInfo(chConnectInfo);

            bRet = TRUE; //连接成功
            break ;
        }
    }while(0);

    if(m_pPeeperWnd)
    {
        DWORD dwFlag = PL_PEEPER_NOTIFY_DISCONNECT;//初始化为连接不成功的标志
        if(bRet)
        {
            dwFlag = PL_PEEPER_NOTIFY_CONNECT;
        }
        ::SendMessage(m_pPeeperWnd->GetSafeHwnd(),
            PL_CONNECT_MESSAGE, (WPARAM)PL_CONNECT_MESSAGE, (LPARAM)dwFlag);//发送数据
    }

    return bRet ? S_OK : E_FAIL;//返回成功或者失败
}
```

(2)取消连接

```
BOOL CPeeperWnd::ExitConnect()
{
    if(m_sckClient[0] != INVALID_SOCKET)
    {
        ::PL_SendSocketData(m_sckClient[0], NULL, 0, PL_CLIENT_CLOSE);
        ::closesocket(m_sckClient[0]);
    }
    if(m_sckClient[1] != INVALID_SOCKET)
    {
        ::closesocket(m_sckClient[1]);
    }
    m_sckClient[0] = INVALID_SOCKET;
    m_sckClient[1] = INVALID_SOCKET;

    if(m_pNotifyWnd)
    {
        ::PostMessage(m_pNotifyWnd->GetSafeHwnd(), PL_PEEPER_NOTIFY,
            (WPARAM)PL_PEEPER_NOTIFY_DISCONNECT, (LPARAM)this);
    }

    return TRUE;
}
```

5.2.3 Socket 的设计与实现

(1)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;
}
```

(2)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;
}
```

(3)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;
}
```

（3）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;
}
```

5.2.4 多线程的建立与实现

(1)建立线程

```
int CPeeperWnd::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
    if (CScrollView::OnCreate(lpCreateStruct) == -1)
        return -1;

    m_hPeeperThread.m_pPeeperWnd = this;
    m_hPeeperThread.CreateThread();

    ModifyStyleEx(WS_EX_CLIENTEDGE, 0);

    return 0;
}
```

(2)销毁线程

```
void CPeeperWnd::OnDestroy()
{
    ExitConnect();
    if(::IsWindow(m_hWnd))
    {
        KillTimer(m_nTimerID);
    }
    if(m_hPeeperThread.m_hThread != NULL)
    {
        ::TerminateThread(m_hPeeperThread.m_hThread, 0xDB);
        ::WaitForSingleObject(m_hPeeperThread.m_hThread, INFINITE);
        m_hPeeperThread.m_hThread = NULL;
        TRACE(_T("Peeper Thread Exit.\n"));
    }
    if(m_pNotifyWnd)
    {
        ::PostMessage(m_pNotifyWnd->GetSafeHwnd(), PL_PEEPER_NOTIFY,
            (WPARAM)PL_PEEPER_NOTIFY_CLOSE, (LPARAM)this);
    }
    CScrollView::OnDestroy();
}
```

5.4 图形界面设计及实现

图形界面基于 MFC 框架实现，其中主界面包括一个菜单、工具栏以及状态栏。除此之外还要建立窗口、快捷键等。由于图形界面**代码量大且重复性强**，因此本节不进行截图展示。

5.4.1 窗口类 CWnd

CWnd 封装了 Windows 窗口句柄 HWND。CWnd 对象的创建和销毁，是由 CWnd 的构造函数和析构函数完成的。创建一个窗口需要两步：首先，调用 CWnd 的构造函数，构造一个 CWnd 对象，然后调用成员函数 Create，创建窗口。当用

户要关闭窗口时，可以销毁与窗口相关的 CWnd 对象，或者调用 CWnd 对象的成员函数 DestoryWindow，删除窗口以及其数据结构。创建静态控件时首先建立一个容器，然后再将空间控件添加入容器中，再修改控件参数。

此类中，我主要**重载**了以下成员函数：

表格 2 窗口类主要重载函数

函数名	解释
BOOL ModifyStyle(DWORD dwRemove, DWORD dwAdd, UNIT uFlags = 0)	修改窗口风格
void GetWindowRect(LPRECT lpRect)	得到窗口的矩形位置
UNIT SetTimer(UINT nIDEvent, UINT nElapse, void(CALLBACK EXPORT* lpfnTimer)(HWND, UINT, UINT, DWORD))	创建一个时钟
afx_msg void OnSize(LPCREARESRTUCT lpCreateStruct)	窗口创建时被调用
afx_msg void OnDestory()	窗口销毁时被调用
afx_msg void OnGetMinMaxInfo(MANMAXINFO FAR* lpMMI)	需要得到窗口尺寸时被调用

5.4.2 下拉层叠式菜单的实现

菜单的创建主要可以分为以下四个步骤：

步骤一：创建要拥有菜单的应用程序；

步骤二：为用户的工程创建一菜单资源；

步骤三：为用户的应用程序定制菜单资源；

步骤四：通过将用户的程序与创建的菜单项相联，为菜单增加处理函数。

当用户选择了一个有效的菜单项时，系统会向应用程序发送一个 WM_COMMAND 消息，在消息的参数中表明来源。MFC 对 WM_COMMAND 消息的映射方式有两种：ON_COMMAND 映射和 ON_UPDATE_COMMAND_UI 映射。

我用类 CMenu 用来处理和菜单有关的功能。要生成 CMenu 对象，需要调用 BOOL CMenu::LoadMenu(UINT nIDResource)函数从资源中装入菜单，然后对菜单进行动态修改。

5.4.3 工具栏的实现

创建工具栏分为五步：

步骤一：创建按钮映像位图；

步骤二：定义命令码数组；

步骤三：创建并初始化工具栏对象，再调用 CToolBar::Create 生成。

步骤四：将储存在位图资源中的按钮位图与程序的工具栏相连接。这可以通过调用函数 CToolBar::LoadBitmap 实现；

步骤五：关联按钮与命令 ID，通过调用 CToolBar::SetButtons()来实现。

在创建工具栏之后，还要可以将工具栏者隐藏或者漂浮。

(1) 显示或者隐藏工具栏

由于工具栏是一个窗口，它的显隐通过调用父类 CWnd 的成员函数实现。在改变工具栏状态之前，需要通过函数 CWnd::GetStyle 了解当前工具栏的状态。该函数不带参数，其原型为：SWORD GetStyle()const；通过对函数 CWnd::SetStyle 的调用可以改变某些窗口风格。

一旦在程序中改变了工具栏的显示、隐藏状态，就必须将这一改变通知程序窗口，并调用函数 CFrameWnd::RecalcLayout，再次计算控制条的位置。该函数不带参数，在程序中可以简单地进行调用。只要通过下列代码：RecalcLayout()；即可将该工具栏的变化通知到程序窗口。

(2) 泊位和漂浮工具栏

工具栏可以在泊位在框架窗口的任意一边（上、下、左、右），或者漂浮在屏幕上的任何地方。实现泊位的方法可以表示如下。

首先，框架窗口调用 CFrameWnd::EnableDocking 函数，使在框架窗口获得泊位控制条的许可，并且指明在框架窗口的哪边接收泊位。然后，工具栏调用 ControlBar::Enable Docking 使工具栏获得泊位许可。最后，框架窗口调用 CFrameWnd::DockControlBar 泊位工具栏。

(3)提示窗口

如果工具栏制定了 CBRS_TOOLTIPS 风格，当鼠标落在工具栏的某个按钮上时，就会在鼠标附近弹出一个文本框——Tooltip 提示窗口。

Tooltip 窗口也是 Windows 控制窗口。在一个线程的生存期间，最多只能拥有一个 Tooltip 窗口。我利用了 CWnd 本身自身支持的 tooptip 来实现，这种方法

适用给控件增加 tooltip，方法如下：

步骤一：在窗口中增加消息映射 ON_NOTIFY_EX(TTN_NEEDTEXT, 0, SetTipText)；

步骤二：EnableToolTips(TRUE)，使用这个方法调用这个函数是必不可少的，我在 CDialog::OnInitDialog 中调用。

步骤三：在窗口中增加一个函数用于动态提供显示内容，其原型为 BOOL SetTipText(UINT id, NMHDR *pTTTStruct, LRESULT *pResult)。

5.4.4 状态栏的实现

状态栏是能显示一段文本的控件，这些文本不断地通知用户应用程序的当前状态。与工具栏类相似，MFC 使用 CStatusBarCtrl 类对状态栏窗口进行封装，但是直接使用这个类不是很方便，因此 MFC 提供了 CStatusBar 来处理状态栏，如图 11.2 所示。CStatusBar 类同样派生自 CControlBar 类，这里就不再详细介绍 CControlBar 类。

状态栏实际上是个窗口，一般分为几个窗格，而且每个窗格显示？？是不同的。AppWizard 会为应用程序自动创建一个状态栏，该状态栏包括几个窗格，？？个窗格分别用来显示状态栏提示和 CAPSLOCK、NUMLOCK、SCROLL LOCK 键。在 MFC 中，状态栏的功能由 CStatusBar 类实现。创建一个状态栏需要以下三个步骤。

步骤一：构建一个 CStatusBar 对象。该对象隶属于主窗口。

步骤二：调用 CStatusBar::Create 创建状态栏窗口。状态栏窗口的创建在主框架窗口的 OnCreate 函数中完成。

步骤三：调用 CStatusBar::SetIndicators 函数分配窗格，并将状态栏的每一个窗格与一个字符串 ID 相关联。

6. 系统的运行和测试

6.1 运行环境说明

本次运行所采用的机器配置如表 4 所示：

表格 3 测试机器配置

	硬件环境		软件环境	
	CPU	内存	操作系统	开发工具

服务器（若干）	4G	256G	Windows XP	Visual C++ 6.0
客户端	4G	256G	Windows XP	Visual C++ 6.0

6.2 运行和测试

主控端界面采用 MFC 开发的，它拥有一个主界面和若干功能窗体。整个主控界面非常简 ，它把相关的功能呈现在用户面前，用户可非常容易的操作该软件就能实现相应的功能达到用户的目的。

测试部分遵循**“静态测试与动态测试相结合”**的原则，按照功能模块测试、系统测试的顺序“**自下而上**”的进行。

6.2.1 系统主界面展示

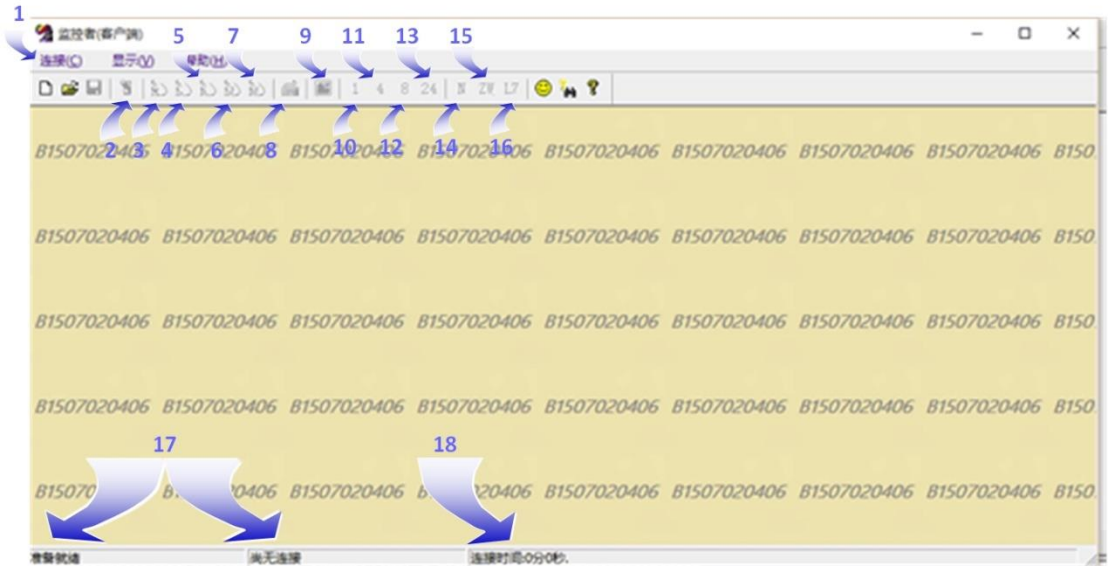


图 13 系统主界面

系统主界面按键解释如表 6 所示：

表格 4 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

6.2.2 连接菜单展示以及连接建立测试

点击“连接”按钮，执行后如图 18 所示：

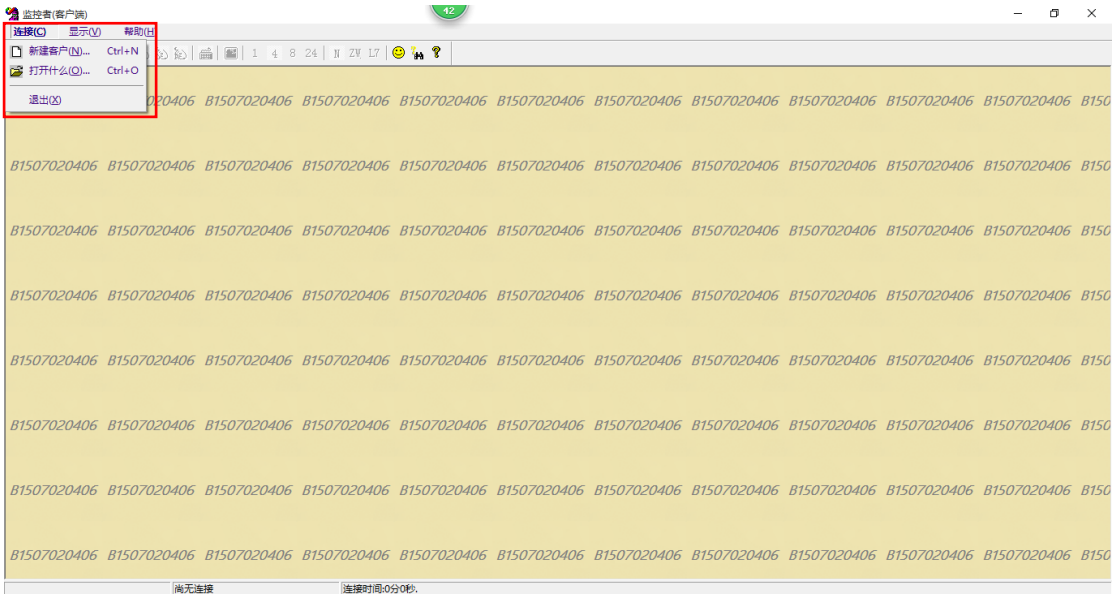


图 14 连接菜单

选择“新建客户”，跳出子窗口，如图 19 所示。可见服务器默认地址为 127.0.0.1（即环回地址），服务器默认端口为 5180，默认位数为 4，图像传输速度为 1000bit/s。

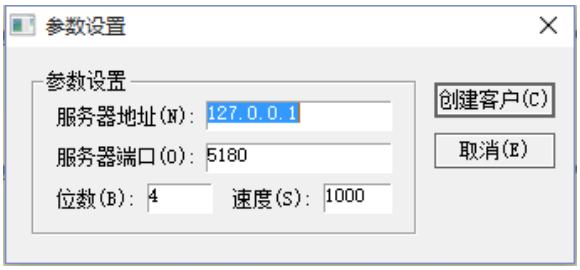


图 15 参数设置

服务器地址（自动获取本机，手动输入服务端 IP 地址）：10.20.148.167

服务器端口（自动获取）：5180

在客户端运行初始状态，在 IP 地址栏里面填写服务器端地址，在端口号里面

填写服务器端的端口号（本程序服务器端使用的端口号是 9600），完成此项操作之后就点击连接按钮对服务器端进行连接，当出现 Client 连接成功之后，就表示已经于服务器端连接上了，然后就可以对远端计算机进行远程操作了。

在连接之前我们可以看到状态栏相关信息，此时系统准备就绪、商务连接、连接建立时间为 0s.

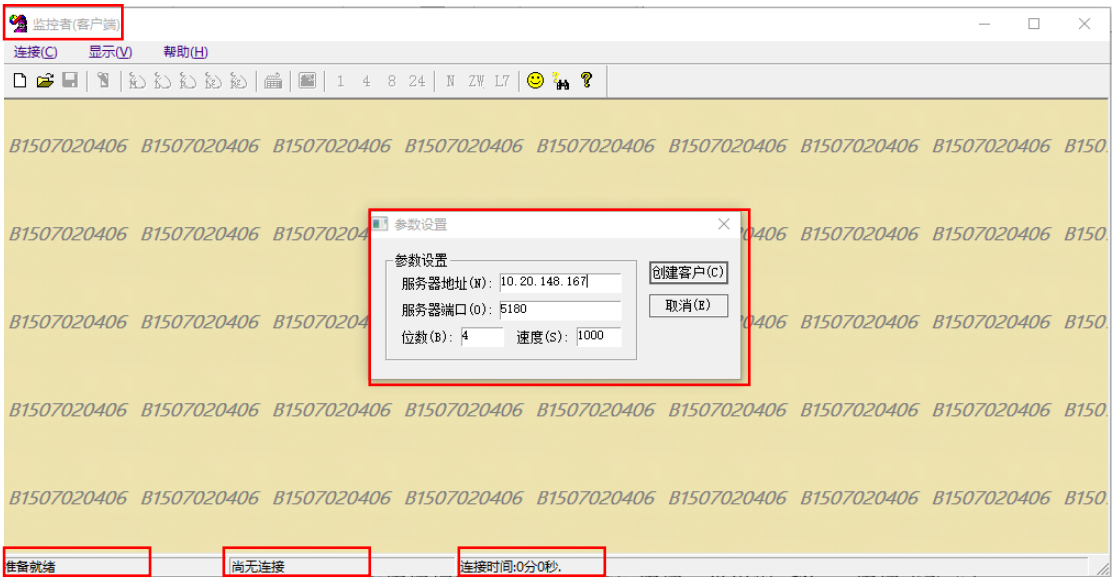


图 16 连接服务器

输入相关参数，点击“创建客户”。结果如图 20 所示。我们可以看到系统自动跳出子窗口，显示“啦啦啦开始连接啦啦啦”，并且也在用户栏中显示

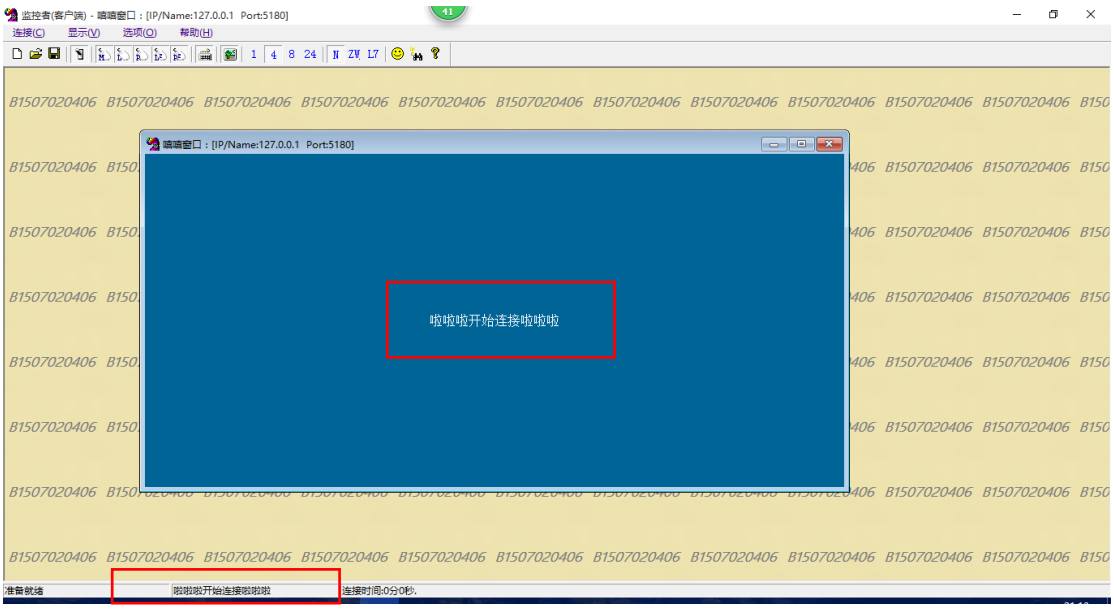


图 17 点击创建客户后

(1) 连接失败

若连接失败，则显示“连接失败啊啊啊啊啊，开始重试”。如图 21 所示：

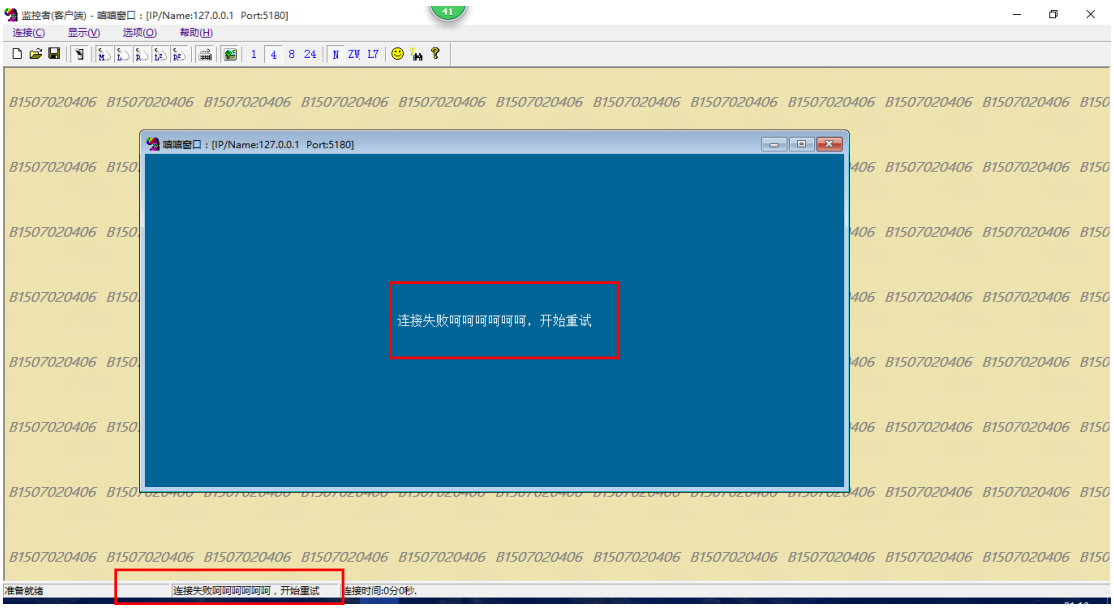


图 18 连接失败

（2）连接成功

若连接成功则继续以下测试内容。

6.2.3 色彩选择测试

(1)点击“1”按钮，接收单色图像，如图 22 所示：

图 19 单色图象

(2)点击“4”按钮，接收 16 色图像，如图 23 所示

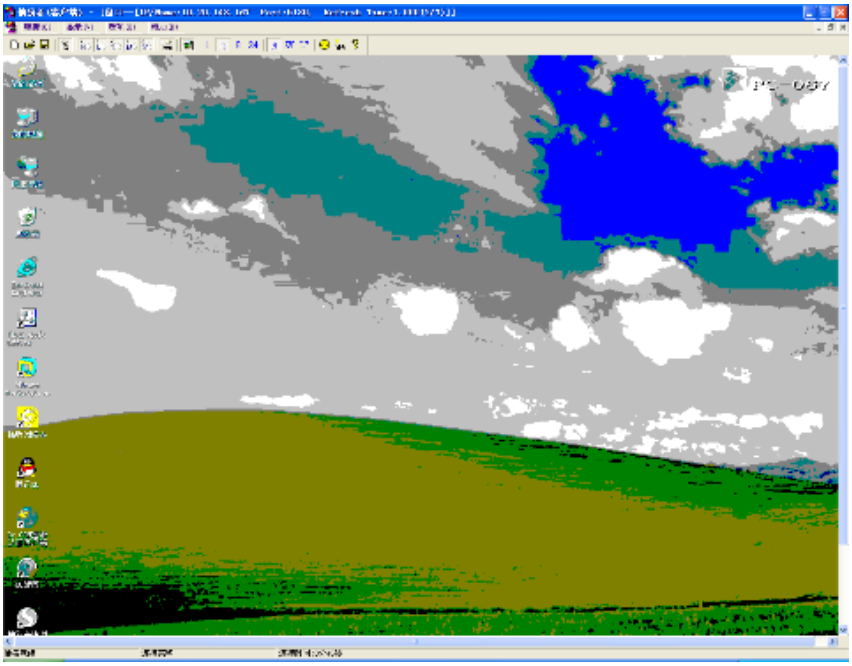


图 20 16 色图像

(3)点击“8”按钮，接收 256 色图像，如图 24 所示：

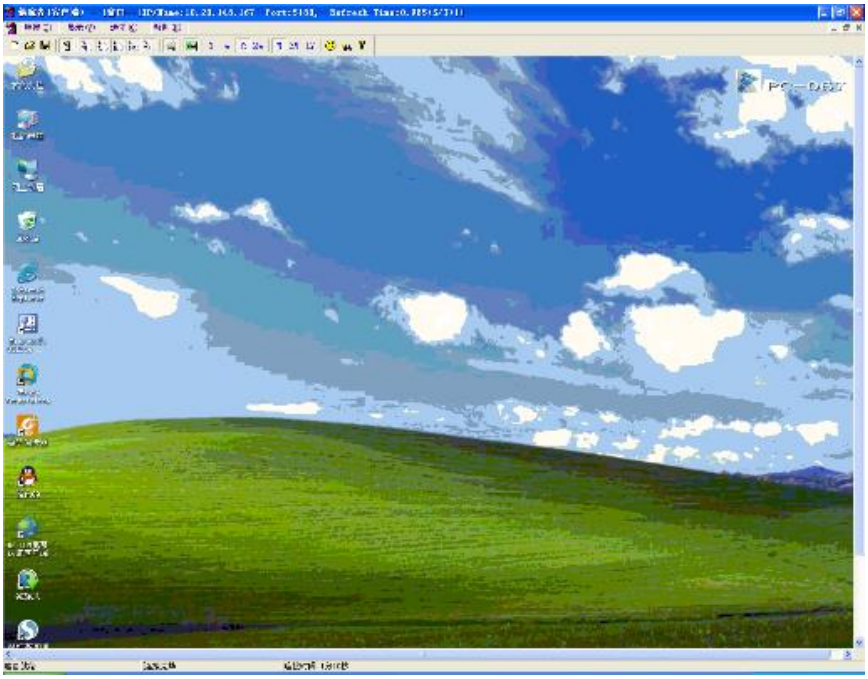


图 21 256 色图像

(4)点击“24”按钮，接收真彩图像，如图 25 所示

图 22 真彩图像

6.2.4　图像压缩选择测试

(1)接收不压缩图像，传输时间约为 **1** 秒。

图 23 不压缩图像

(2)接收 LZW 压缩图像，传输时间约为 **5** 秒

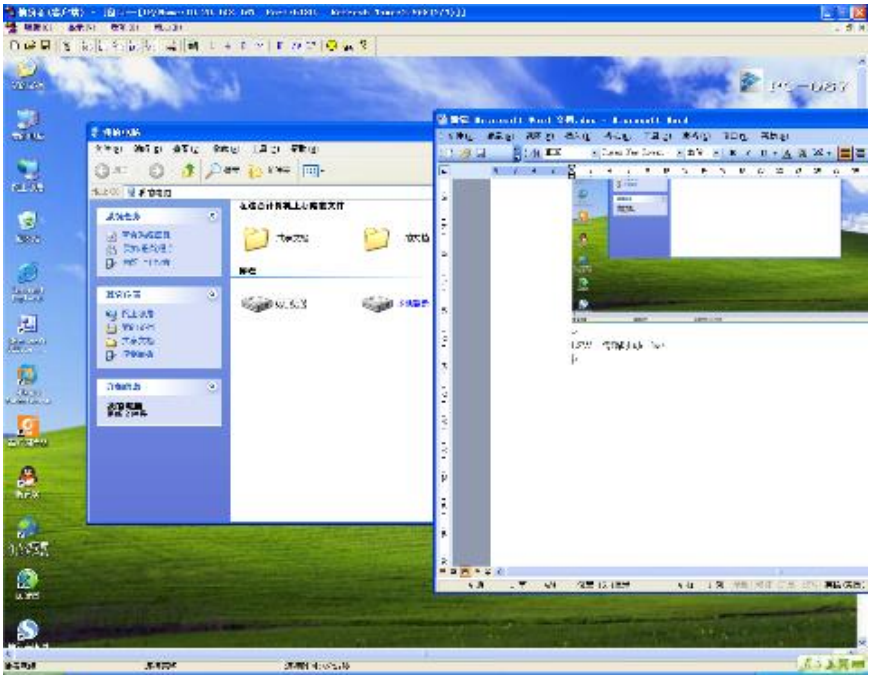


图 24　LZW 压缩图像

(3)接收 LZ77 压缩图像，传输时间约为 **3** 秒。

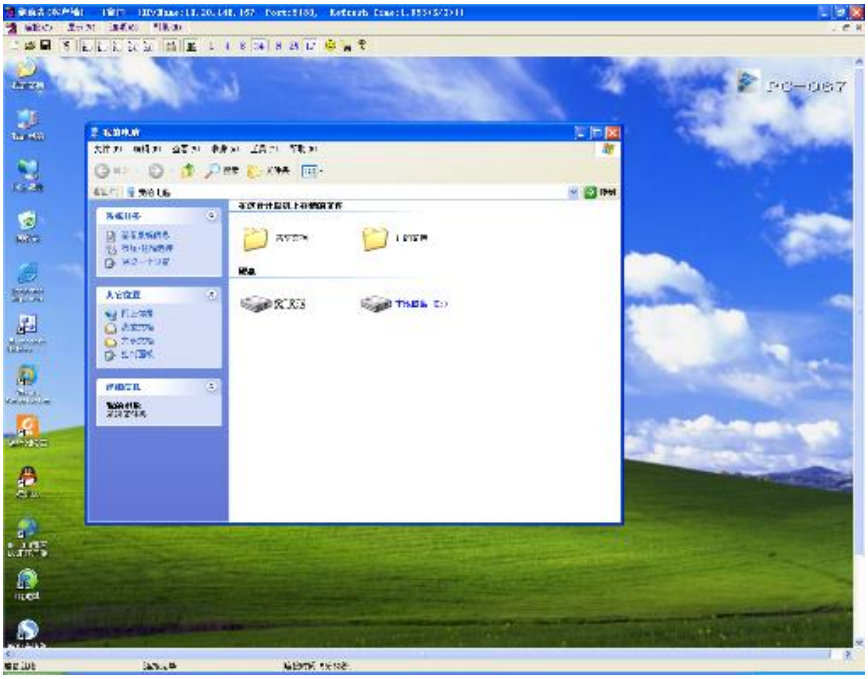


图 25 LZ77 压缩图像

6.2.5 控制对方桌面测试

客户端可对服务器进行控制，此处以文件操作为例；

(1)对服务端可以实现任意操控，比如建立文件：

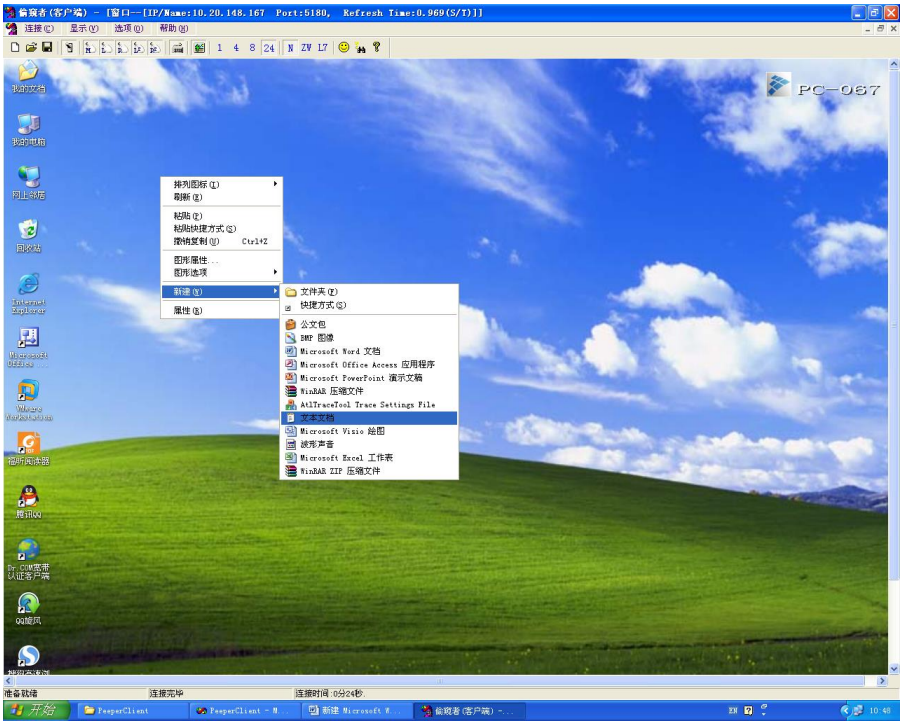


图 26 建立文件

(2)文件重命名

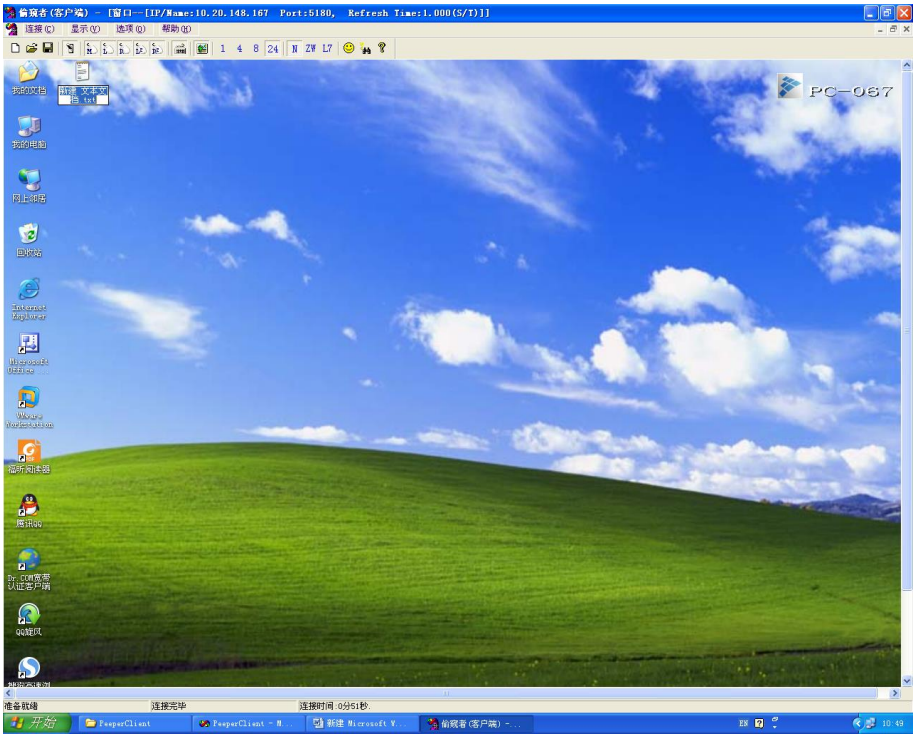


图 27 文件重命名

(3)编辑文件

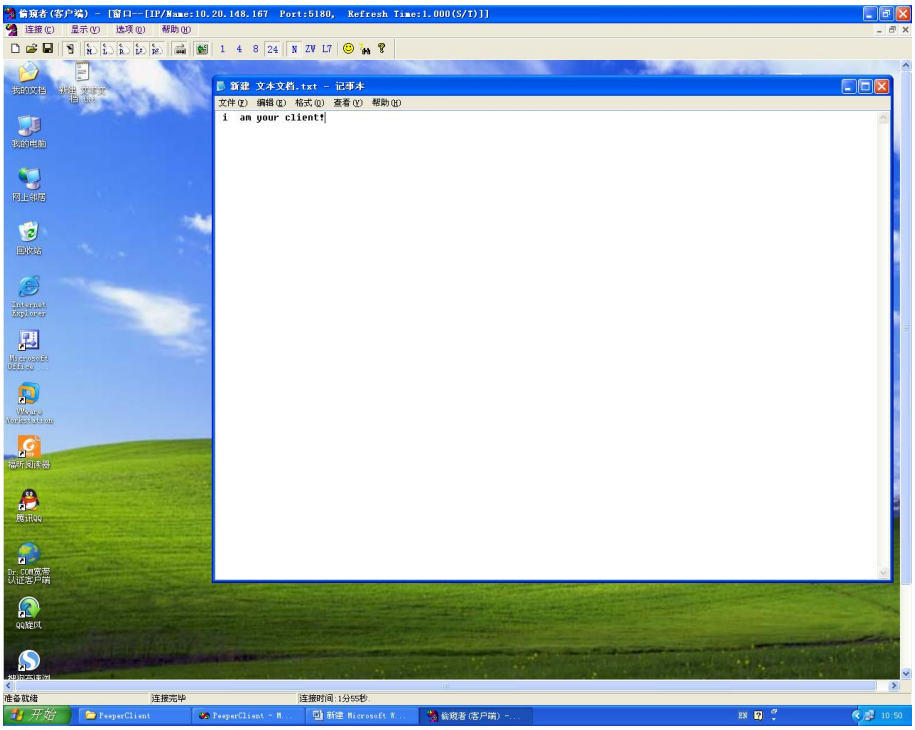


图 28 编辑文件

(4)保存文件

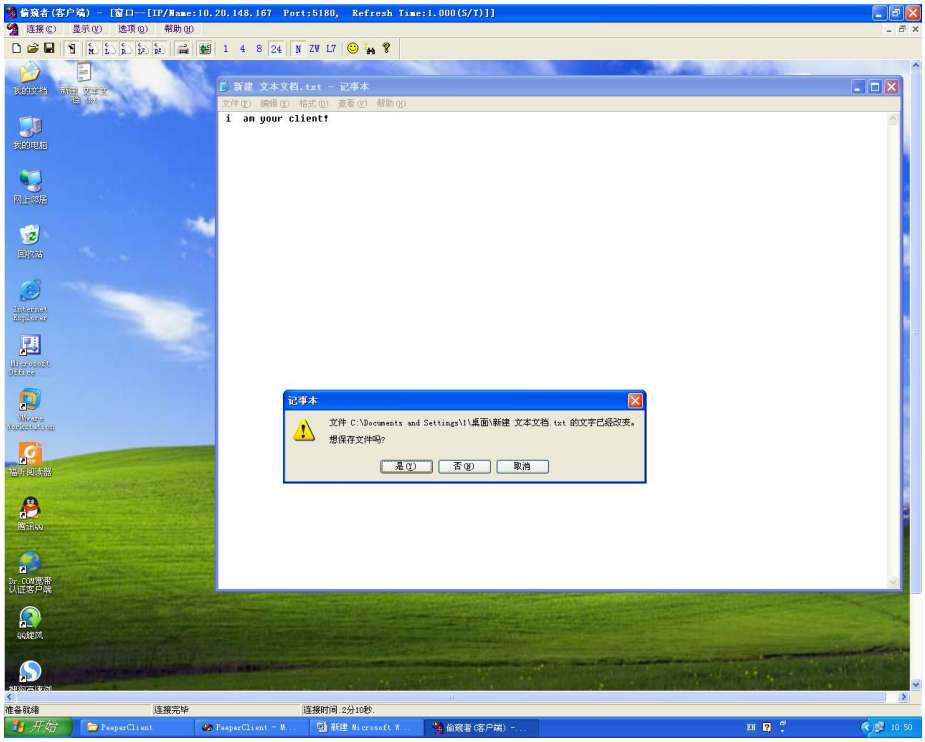


图 29 保存文件

6.2.6 “显示”菜单展示以及包含功能测试

点击“显示”，可以看到“主工具条”与“状态条”默认为“显示”，当无连接建立时，其他按键不可选。

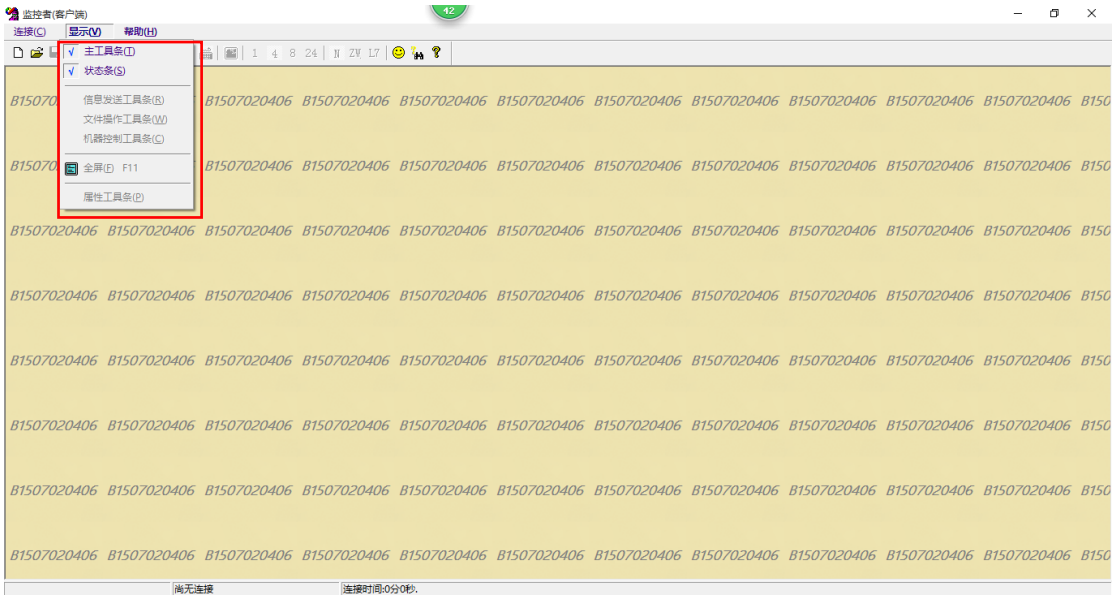


图 30 “显示”菜单栏

连接成功后，分别点击“信息发送工具条”、“文件操作工具条”和“机器控制工具条”，按照功能进行测试。

(1) 发送消息功能测试

客户端发送消息：I am your client；

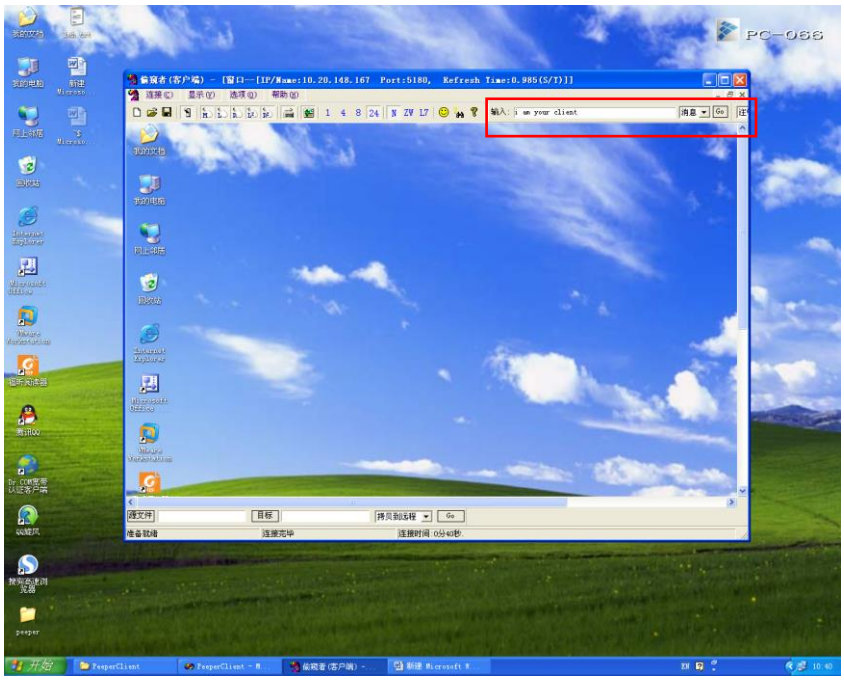


图　31 客户端发送消息窗口

服务端收到消息：

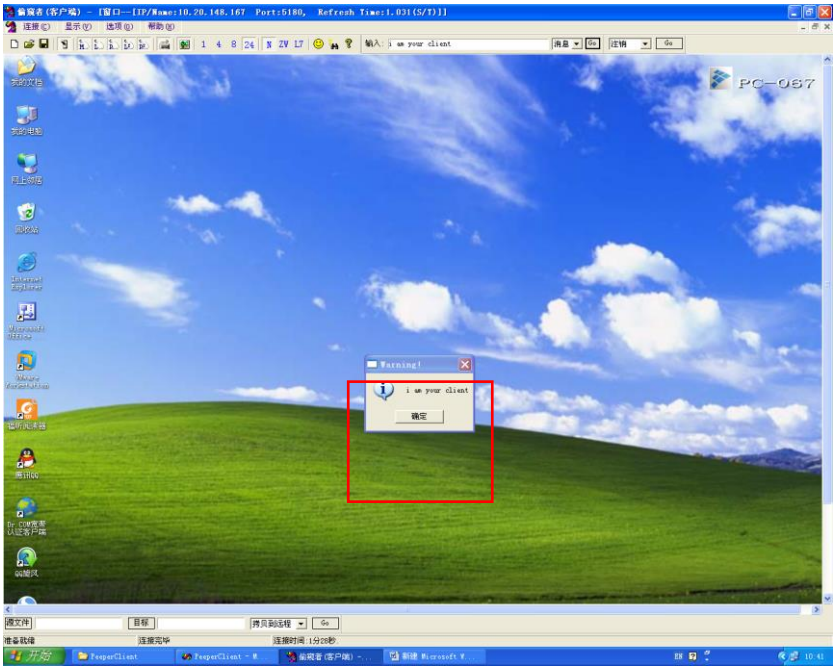


图 32 服务端接收窗口

(2) 发送命令功能测试

客户端发送命令：ipconfig

服务端接收命令；

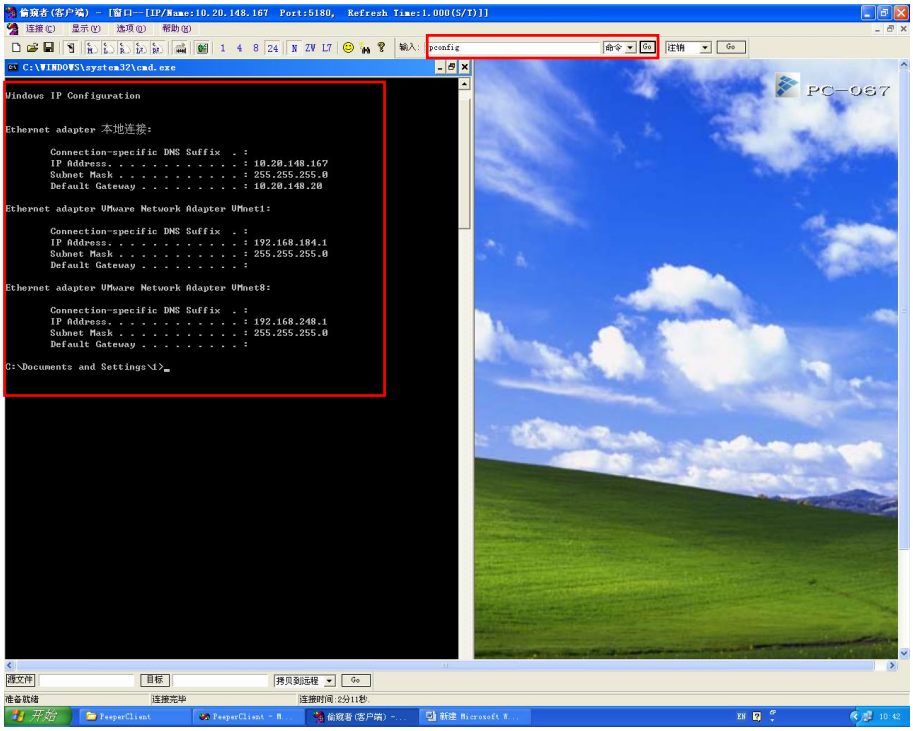


图 33 发送命令窗口

(3) 发送关机命令，服务端关机

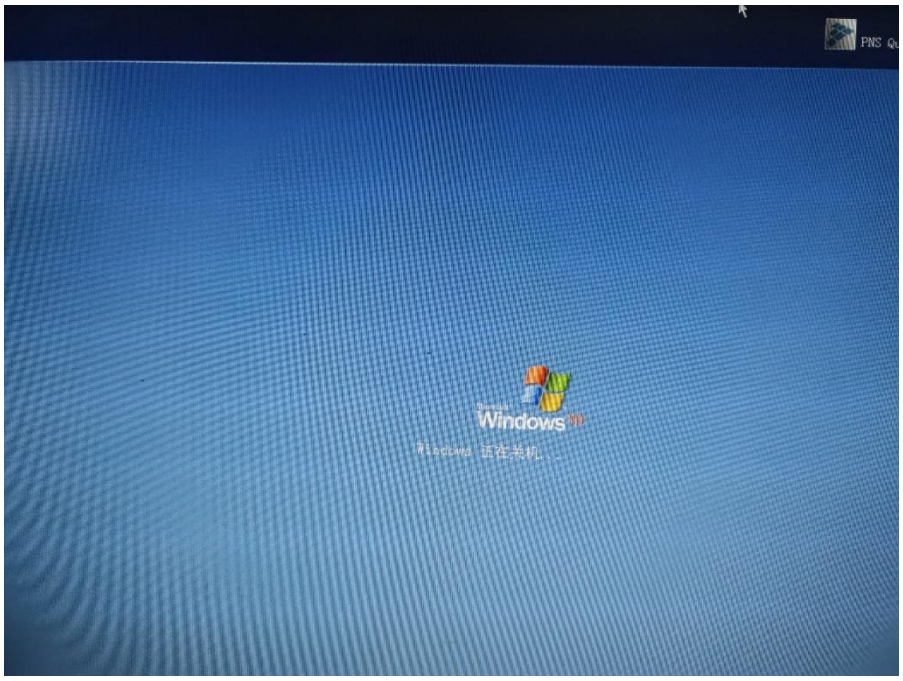


图 34 服务端关机图

(4)发送注销命令，服务端注销

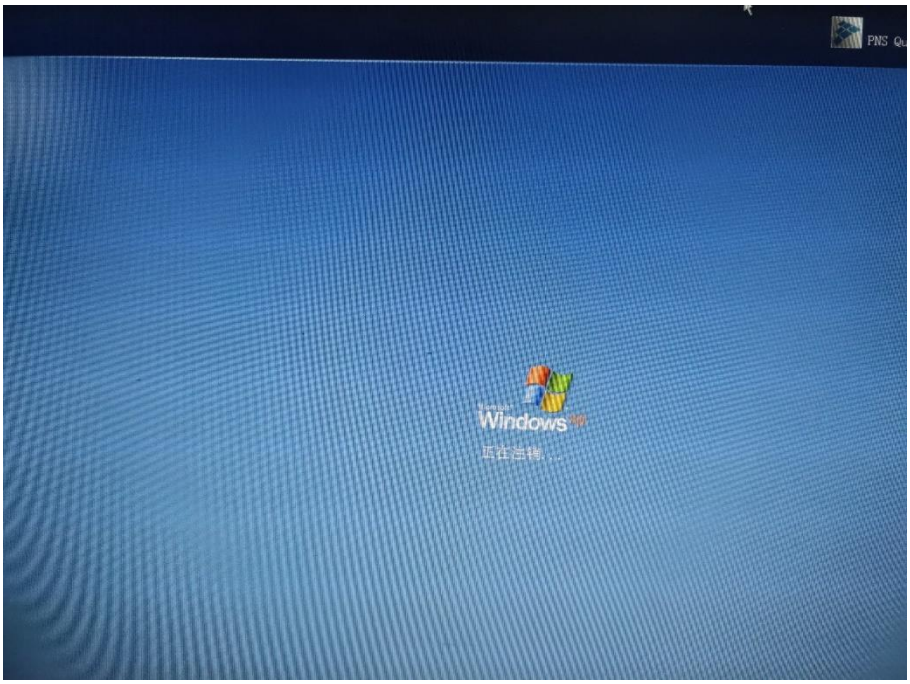


图 35 服务端注销图

6.2.7 “帮助”菜单展示

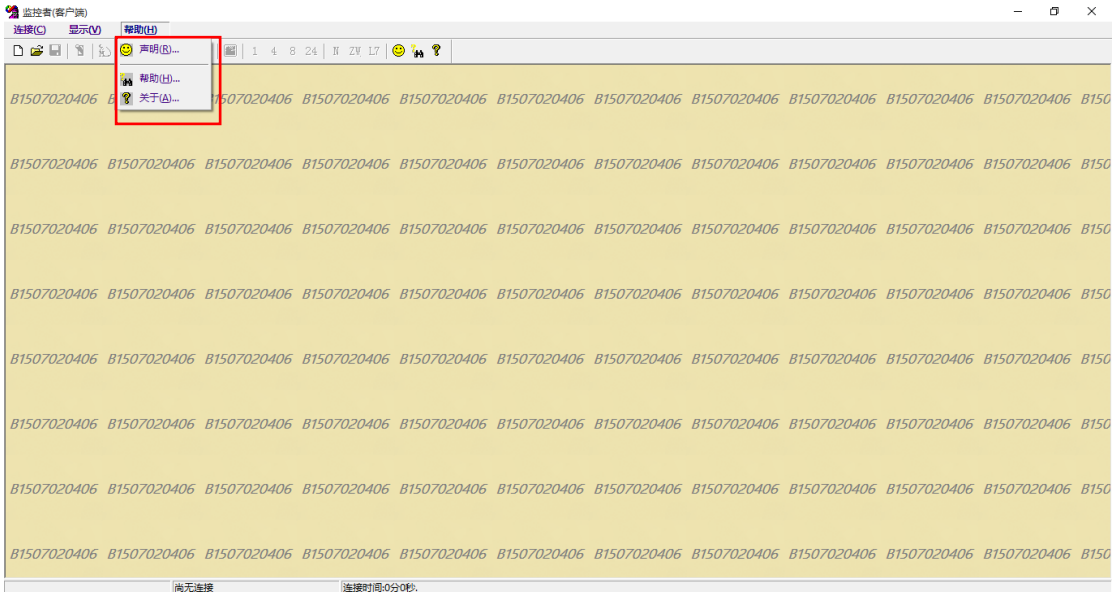


图 36 “帮助”菜单栏

(1)点击“声明”

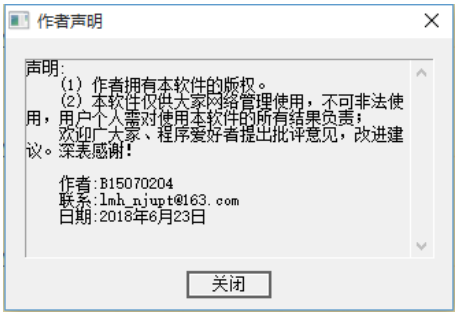


图 37 作者声明

(2)点击“关于”

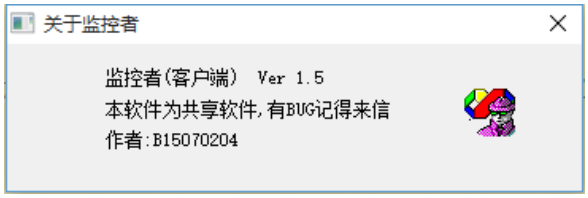


图 38 关于监控着

6.3 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，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。

7. 调试过程中的问题

开发过程中，遇到诸多问题，大部分已经解决。罗列过程中的几个问题如下：

7.1 Socket 创建失败

问题描述：在 CChatApp 的 InitInstance 函数调用 AfxSocketInit 函数以加载套接字库，然后在 CChatDlg 的自定义的函数 InitSocket 里进行创建套接字等操作，其它一些操作都实现后，我调试运行，却提示“套接字创建失败”。

原因及对策：AfxSocketInit 函数的调用放在了 DoModal()的后面了，使得 AfxSocetInit 函数没有被执行到，因为 m_socketf 未被初始化。代码应该如下：C/C++ code // 应该放在 DOModa() 之前调用 if (!AfxSocketInit()) { AfxMessageBox(L" error "); return FALSE; }

7.2 内存溢出

问题描述：程序输入 IP 地址后，出现错误提示如图 39 所示。

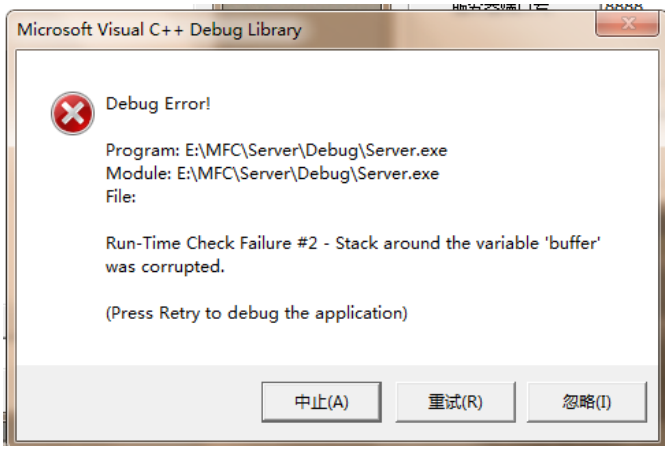


图 39 错误提示

原因及对策：在弹出的对话框按相应按钮进入调试，按 Alt+7 键查看 Call Stack 即“调用堆栈”里面从上到下列出的对应从里层到外层的函数调用历史。双击某一行可将光标定位到此次调用的源代码或汇编指令处，最终发现是数组定义错误，出现溢出。

7.3 Socket 数据传输错误

问题描述：传输过来的数据为乱码。最终将问题定位为客服端是用的多字节字符集，而服务器端是 unicode 字符集。二者之间很多数据的转换有问题。

原因及对策：最终将问题定位为客服端是用的多字节字符集，而服务器端是 unicode 字符集。二者之间很多数据的转换有问题。最终将客户端也在 unicode 字符环境下开发，就可以正常的传输和接收数据。

7.4 多线程弹出对话框错误

问题描述：计划的是在一个线程中通过 Create 和 ShowWindow 弹出一个对话框，但是偶尔会出错。跟踪发现问题是发生在 Create 函数中。

原因及对策：查阅资料得 C++中在线程间传递对象是不安全的。原因有:

(1)MFC 的大多数类不是线程安全的，调用传入对象的成员函数可能不会报错，但是未必能达到程序预定的功能.

(2)MFC 与界面有关的类，其大多数成员方法都是通过 sendmessage 实现的，如果消息处理函数本身不是线程安全的，你从工作线程中调用这些方法迟早会同你界面线程的用户消息响应发生冲突；

(3)对于 CWnd 相关的类，即使传入窗口句柄，有时操作也会引起异常（ASSERT 异常）：通过句柄获取窗口对象并且调用其成员函数或者成员变量！因为该对象是临时对象，访问其成员变量没有意义，访问其成员函数可能会抛出异常！

最后是通过自定义消息响应函数的机制来实现的。

在对话框类的头文件里加上：

```
#define WM_MY_MESSAGE WM_USER+100
```

在它的 AFX_MSG 消息响应函数模块中加上：

```
afx_msg void OnMyMessage(WParam wParam,LParam lParam);
```

在 cpp 文件中添加消息映射：

```
On_Message(WM_MY_MESSAGE,OnMyMessage)
```

在线程函数方面就这么写了：

```
void CTestDlg::OnButton1(){
hThread=CreateThread(NULL,0,ThreadFunc,m_hWnd,0,&ThreadID);
CloseHandle(hThread);
  }
DWORD WINAPI ThreadFunc(LPVOID LpParameter){
      ……
      HWND hwnd=(HWND)LpParameter;
      CWnd *pWnd=CWnd::FromHandle(hwnd);
      pWnd->SendMessage(WM_My_Message,(WPARAM)(&str));
}
```

7.5 位图显示错误

问题描述：使用了 GetDIBits 函数来显示位图，但是图是倒过来的。

原因及策略：有两个用来得到位图图像数据的 API，分别是 GetBitmapBits 和 GetDIBits；前者是用来得到设备独立位图的 BITS，后者是得到兼容位图的 BITS，所以在调用该函数的时候，第一个主要的区别是：GetDIBits 需要提供一个设备内容，同时需要将位图的 HANDLE 选进这个设备内容（DC）才能能够得到位图的信息。我想上面的区别大家可能都知道，其实它还隐藏着另一个区别：

就是对于同一个位图，得到的 BITS 内容的 BUFFER 不一样！大家都知道 BMP 文件存储数据是倒叙的，也就是从图像的右下角开始存储，文件的最后是图像的左上角（这个来历可以看：WINDOWS 编程中介绍）；使用 GetBitmapBits 取得的 BUFFER，位图的右下角的内容为第一个字节，实际上和真正的图像字节应该是一样的，而 GetDIBits 刚好相反，其 BUFFER 的顺序符合 BMP 文件中的顺序，如果按照正常的坐标，其存储顺序应该是倒叙。

7.6 子窗口建立崩溃

问题描述：源代码为：

```
void CMDIChildWnd::AssertValid() const{
    CFrameWnd::AssertValid();
    ASSERT(m_hMenuShared == NULL || ::IsMenu(m_hMenuShared));
}
```

原因及对策：Attach()就是 m_hMenuShared 句柄与 m_hMenu 相连接。用 m_hMenu 连接后就以 m_hMenu 来替代 m_hMenuShared 实例对象操作。

将 ASSERT(m_hMenuShared == NULL || ::IsMenu(m_hMenuShared));

换成：ASSERT(m_hMenu== NULL || ::IsMenu(m_hMenu));

7.7 未解决的问题：

问题描述：锁定桌面未实现，代码如下：

```
void WINAPI PL_LockDesktop(BOOL bLock,POINT point)//锁屏  //
{
    ::ShowCursor(!bLock);
    //
    if(!bLock){
        ::SetCursorPos(0, 0);//??
    }else{
        ::SetCursorPos(point.x, point.y);
    }
    //
    /*
    ShowCursor函数功能：该函数显示或隐藏光标。
    函数原型：int ShowCursor（BOOL bShow）；
    参数：
    bShow：确定内部的显示计数器是增加还是减少，如果bShow为TRUE，则显示计数器增加1，如果bShow为FALSE，则计数器减1。
    返回值：返回值规定新的显示计数器。
    备注：该函数设置了一个内部显示计数器以确定光标是否显示，仅当显示计数器的值大于或等于0时，光标才显示，
    如果安装了鼠标，则显示计数的初始值为0。如果没有安装鼠标，显示计数是-1。
    这里面遇到的问题是我在ShowCursor（false）里面隐藏鼠标的时候用的是一个循环在里面，结果发现要显示鼠标的时候根本出不来，
    之后过了一会发现鼠标出来了，之后以为是有延迟的问题，最后发现竟然是这个东西是靠计数器在里面加加或减减的，
    所以就会出现在循环的时候，不断减，要显示鼠标的时候又必须加回这么多次减的数才行。
    */
    //::SystemParametersInfo(SPI_SETFASTTASKSWITCH, (int)(!bLock), NULL, 0);
    //::SystemParametersInfo(SPI_SCREENSAVERRUNNING, (int)bLock, NULL, 0);
    ::SystemParametersInfo(SPI_SETFASTTASKSWITCH, !bLock, NULL, 0);
    ::SystemParametersInfo(SPI_SCREENSAVERRUNNING, !bLock, NULL, 0);
    ::SystemParametersInfo(SPI_GETKEYBOARDPREF, !bLock, NULL, 0);
    ::SystemParametersInfo(SPI_GETMOUSEHOVERWIDTH, int(bLock), NULL, 0);
```

```
    //SystemParametersInfo(Info中是i大写)函数查询或设置系统级参数。参数参考网页：https://baike.so.com/doc/7631000-7905095.html
    /*
    SPI_GETKEYBOARDPREF:它确定用户是否依赖键盘而非鼠标，是否要求应用程序显示键盘接口，以免隐藏。
    pvParam参数必须指向一个BOOL类型变量，如果用户依赖键盘，那么该变量取值为TRUE，否则为FALSE。
    SPI_GETMOUSEHOVERWIDTH:获得在TrackMouseEvent事件中，为产生WM_MOUSEOVER消息而鼠
    标指针必须停留的矩形框的宽度,以像素为单位。参数pvParam必须指向一个UINT变量以接收这个宽度值。
    */
    ::EnableWindow(::GetDesktopWindow(), !bLock);
    /*
    if(!block){
        ::SetCursorPos(0, 0);
    }*/
    /*
    该函数返回桌面窗口的句柄。桌面窗口覆盖整个屏幕。桌面窗口是一个要在其上绘制所有的图标和其他窗口的区域。
    函数原型:HWND GetDesktopWindow(VOID)
    参数:无。
    返回值:函数返回桌面窗口的句柄。
    */

}
```

ShowCursor 函数设置了一个内部显示计数器以确定光标是否显示，仅当显示计数器的值大于或等于 0 时，光标才显示，如果安装了鼠标，则显示计数的初始值为 0。如果没有安装鼠标，显示计数是-1。

这里面遇到的问题是我在 ShowCursor（false）里面隐藏鼠标的时候用的是一个循环在里面，结果发现要显示鼠标的时候根本出不来，之后过了一会发现鼠标出来了，之后以为是有延迟的问题，最后发现竟然是这个东西是靠计数器在里面加加或减减的，所以就会出现在循环的时候，不断减，要显示鼠标的时候又必须加回这么多次减的数才行。

8. 课程设计总结

8.1 总结

本系统采用“原型法”的开发方法，较原系统已改善很多，但是仍存在以下几个问题需要进一步开发优化：①图形界面不够美观，工具栏不能自助添加或取消；②几种压缩算法最终得到的图像效果人眼识别不出，只能感受到时间上的差异，测试证明，不压缩图像已经达到较好效果；③控制鼠标、键盘以及外部设备功能未完成。

在此过程中，我养成了一个极其重要的习惯：**纠错前先思考**。如果你一头扎进问题中，你可能只解决了当前出现问题的代码，但如果你先思考这个错误，这个错误是怎么引入的？你通常发现和纠正一个更高层次的问题，进而改进了系统设计，防止了更多错误的出现。我认识到这种编程思考模式非常的重要。有些人痴迷于一行行的、使用各种工具来调试所有的东西。但我现在相信，思考——不看代码的思考——是最好的调试途径，因为它能让你开发出更好的软件。

本系统的改进花费了很长时间，但是也积累了很多宝贵的经验。程序设计也遇到了诸多问题，这些问题不仅仅训练我调试错误的能力，更多反映出来编程时

的逻辑思路或者细节欠缺的地方，只有意识到了这一点，才能在今后避免同类问题，做一个“合格”的程序员。

8.2 展望

本系统是一个远程控制软件，其中由于时间紧促加上本人技术限制，只实现了其中的部分功能。但是这个软件完全还没有达到用户的完全满意，因此需要进一步完善。 需要完善的需求如下：①开发服务端图形界面，以便企业员工使用；②开发远程语音与视频功能，方便管理者与员工的沟通；③开发最小化到任务栏中的功能，以便实现隐藏；④开发注册登录功能，增强系统安全性，识别登录用户；⑤开发对外连接接口，以便企业信息系统集成，更高效的进行管理。

致 谢

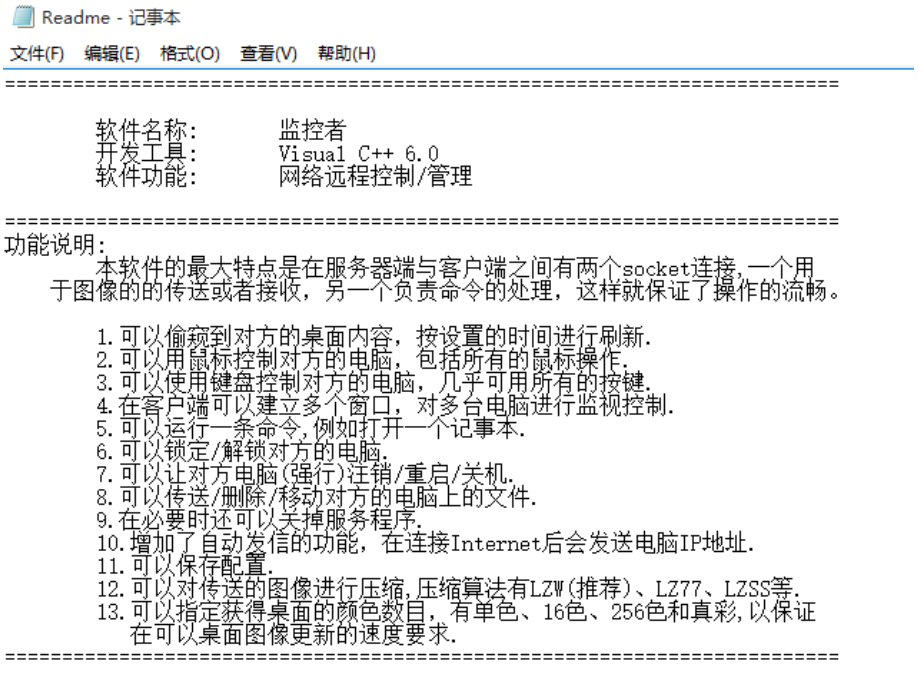
感谢开发团队的配合与付出，感谢李养群老师在过程中的指导。

附 录

Ⅰ 参考书目和网页

a) 《精通 MFC 程序设计》——人民邮电出版社
b) https://blog.csdn.net/ddjj_1980/article/details/7286184
c) https://blog.csdn.net/zhangyinze123/article/details/5289268
d) https://www.cnblogs.com/zfluo/p/5131929.html
e) https://blog.csdn.net/stonesharp/article/details/7709674
f) https://blog.csdn.net/chy19911123/article/details/49928959
g) https://bbs.csdn.net/topics/40086117
h) http://blog.csdn.net/L_yangliu/article/details/11553713
i) http://bbs.csdn.net/topics/390411637?page=1
j) http://blog.csdn.net/ljianhui/article/details/10875883
k) https://www.cnblogs.com/railgunman/archive/2010/11/06/1870867.html
l) https://www.cnblogs.com/likeyiyy/p/3670213.html
m) https://www.cnblogs.com/panfeng412/archive/2011/11/06/2237857.html
n) https://www.cnblogs.com/rosesmall/archive/2012/04/13/2445527.html
o) http://blog.csdn.net/u011068702/article/details/53931648
p) https://www.cnblogs.com/TianFang/archive/2013/01/20/2868889.html
q) http://blog.csdn.net/piaojun_pj/article/details/6098438
r) http://bbs.csdn.net/topics/280060121/
s) http://blog.csdn.net/u011068616/article/details/41819787
t) http://blog.csdn.net/bytxl/article/details/2823047
u) http://bbs.csdn.net/topics/391998869

Ⅱ ReadMe 文件展示



Readme - 记事本

文件(F)　编辑(E)　格式(O)　查看(V)　帮助(H)

```
==========================================================================

        软件名称:       监控者
        开发工具:       Visual C++ 6.0
        软件功能:       网络远程控制/管理

==========================================================================
功能说明:
        本软件的最大特点是在服务器端与客户端之间有两个socket连接,一个用
    于图像的的传送或者接收，另一个负责命令的处理，这样就保证了操作的流畅。

        1.可以偷窥到对方的桌面内容，按设置的时间进行刷新.
        2.可以用鼠标控制对方的电脑，包括所有的鼠标操作.
        3.可以使用键盘控制对方的电脑，几乎可用所有的按键.
        4.在客户端可以建立多个窗口，对多台电脑进行监视控制.
        5.可以运行一条命令,例如打开一个记事本.
        6.可以锁定/解锁对方的电脑.
        7.可以让对方电脑(强行)注销/重启/关机.
        8.可以传送/删除/移动对方的电脑上的文件.
        9.在必要时还可以关掉服务程序.
        10.增加了自动发信的功能，在连接Internet后会发送电脑IP地址.
        11.可以保存配置.
        12.可以对传送的图像进行压缩,压缩算法有LZW(推荐)、LZ77、LZSS等.
        13.可以指定获得桌面的颜色数目，有单色、16色、256色和真彩,以保证
           在可以桌面图像更新的速度要求.
==========================================================================
```