

网络空间安全学院（密码学院）

2022-2023 学年度第 2 学期

J00716 密码测评课程设计课程论文

论文题目： 网络监控与管理软件

学生学号及姓名： **20213006839 甄五四**

学生所在班级： 信息安全(密码学方向)理科实验  
班

论文提交时间： **2024 年 6 月 11 日**

授课教师： 郭祯

教师评阅：

成绩： \_\_\_\_\_

教师签名： \_\_\_\_\_

# 目 录

1	系统设计.....	4
1.1	远程监控的背景知识.....	4
1.2	远程监控软件的原理.....	4
1.3	模块设计.....	5
2	系统实现.....	6
2.1	屏幕截图和编码.....	7
2.2	远程控制.....	7
2.3	内网穿透.....	8
2.4	远程通信.....	9
2.5	可视化界面.....	9
3	关键技术.....	10
3.1	控制端.....	10
3.2	被控制端.....	19
4	测试和运行.....	23
4.1	测试环境.....	23
4.2	运行效果.....	23
5	总结.....	28
	参考文献.....	29
	附录.....	29

## 摘 要

随着计算机应用技术的迅猛发展，图形用户界面已经成为各类计算设备不可或缺的特性。在这一背景下，基于图形化用户界面的远程控制技术应运而生，并广泛应用于远程办公、远程教学、产品演示、远程配置以及高风险环境下的远程作业等场景。本课程设计报告详细描述了一个网络监控和管理系统的设计与实现，该系统旨在实现对远程主机的全面监控和操作能力。系统能够实时监视目标主机的屏幕活动，允许远程操作包括文件管理、进程控制和系统配置等功能。此外，系统还具备监控和记录被监控机与监控机之间的通信内容，以及上传和下载文件的功能。该系统为远程桌面控制的实现，适用于 Windows、Linux 等多种平台。本文详细介绍了系统的设计原理、实现方法以及开发过程中的总结和反思，为未来的改进和升级提供了参考指导。

# 1 系统设计

## 1.1 远程监控的背景知识

远程监控技术是通过网络实现远距离监控和操作计算机的技术。在这种技术下，一台主控端电脑（通常称为远程端）可以远程连接到另一台被监控端电脑（通常称为主机端），无论它们之间的物理距离如何。远程监控技术最常见的应用场景包括局域网内的远程管理和技术支持，但也可以通过互联网进行全球范围的监控和控制。当操作者使用主控端电脑监控被监控端电脑时，他们可以像直接操作本地电脑一样，启动应用程序、访问文件和执行各种操作。然而，实际上，主控端只是将键盘和鼠标输入传输到远程电脑，并将远程电脑的屏幕回传到主控端，所有的处理和存储操作都在被监控端电脑上进行。这种技术使得用户能够在远程实现本地化的操作体验，无论是打开文件、浏览网页还是下载内容。远程监控技术起源可以追溯到 DOS 时代，尽管当时由于技术和网络条件的限制，这些技术并未受到广泛关注。随着网络技术的进步和对计算机管理和支持需求的增加，远程监控技术逐渐成为 IT 管理中不可或缺的一部分。它支持多种网络方式，包括局域网（LAN）、广域网（WAN）、拨号连接以及互联网方式。传统的远程控制软件通常使用 NETBEUI、NETBIOS、IPX/SPX、TCP/IP 等协议实现远程控制，本课程设计采用客户服务器模式基于 Python 实现提供跨平台的支持，适用于不同操作系统和设备，能够对所监控的系统进行必要的控制。远程监控技术不仅提高了效率和便利性，还在信息安全和操作保密性上做出了相应的保障。它在企业、教育和医疗等领域的应用广泛，成为管理和技术支持的重要工具，同时也推动了远程工作和远程服务的发展。

## 1.2 远程监控软件的原理

本课程设计的远程监控和控制软件的工作原理基于客户端-服务器模型，这种软件允许用户在主控端电脑上通过网络连接到另一台被监控端电脑。客户端程序通常安装在主控端，它与被监控端电脑上运行的服务器端程序进行通信。通常情况下，客户端通过安全的网络连接（如加密的 TCP/IP 连接）与服务器端建立

通道，确保数据传输的安全性和可靠性。在建立连接后，客户端可以通过用户界面实时查看被监控端电脑的屏幕活动，获取实时的屏幕截图，并对被监控端电脑进行各种操作。这些操作包括但不限于远程文件管理（浏览、上传、下载文件）、远程命令执行（启动、关闭应用程序）、系统进程管理以及其他管理任务。客户端发送操作命令到服务器端，服务器端收到命令后执行相应操作，并将结果反馈给客户端，通常是操作状态、执行结果或者实时屏幕图像。

### 1.3 模块设计

本课程设计实现了一个基于 Python 的网络监控与管理软件，通过 NOVPN 代理允许客户端和服务端通过网络进行连接，基于 Python 中多个函数库，主要包括 tkinter, cv2, PIL, PyAutoGUI, numpy, PIL, PyAutoGUI, mouse 和 paramiko, 实现网络监控与管理软件。整个系统设计涵盖了远程计算机的远程访问和控制，监控机和被监控机之间的通信，以及主控机和被控机之间的文件传输等关键功能。

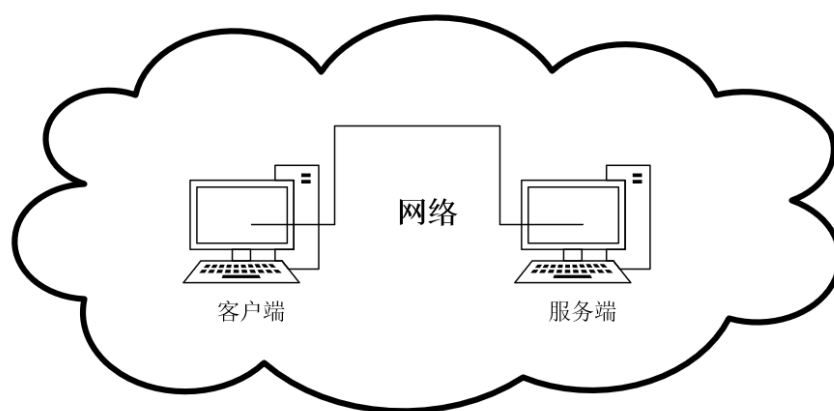


图 1 系统运行环境

具体分为四个主要部分：

1. 远程主机端(被监控机)：在远程主机上部署基于 Python 的服务端程序。通过 SSH 协议确保通信的安全性。服务端等待客户端的连接请求，并响应来自本地控制端的指令。
2. 本地控制端（监控机）：在本地计算机上运行客户端程序，通过网络连接到远程主机端。客户端程序提供用户界面，允许操作者远程监控和控制远程主机。

3. 图像处理模块：在本地控制端实现图像处理功能，利用 numpy 和 PIL 库处理远程主机屏幕的截屏图像。处理后的图像数据通过 PyAutoGUI 和 mouse 模块模拟鼠标和键盘事件。模拟的事件发送至远程主机端，实现对远程主机的实时远程控制和操作。
4. 聊天应用模块：基于客户服务器模式建立通信套接字，实现监控机和被监控机之间的通信，在建立通信连接后，监控机和被监控机可以实现远程通信，还可以根据需要进行上传，实现监控机能够上传文件到被监控机指定的地址，还可以能够从被监控机下载特定的文件。

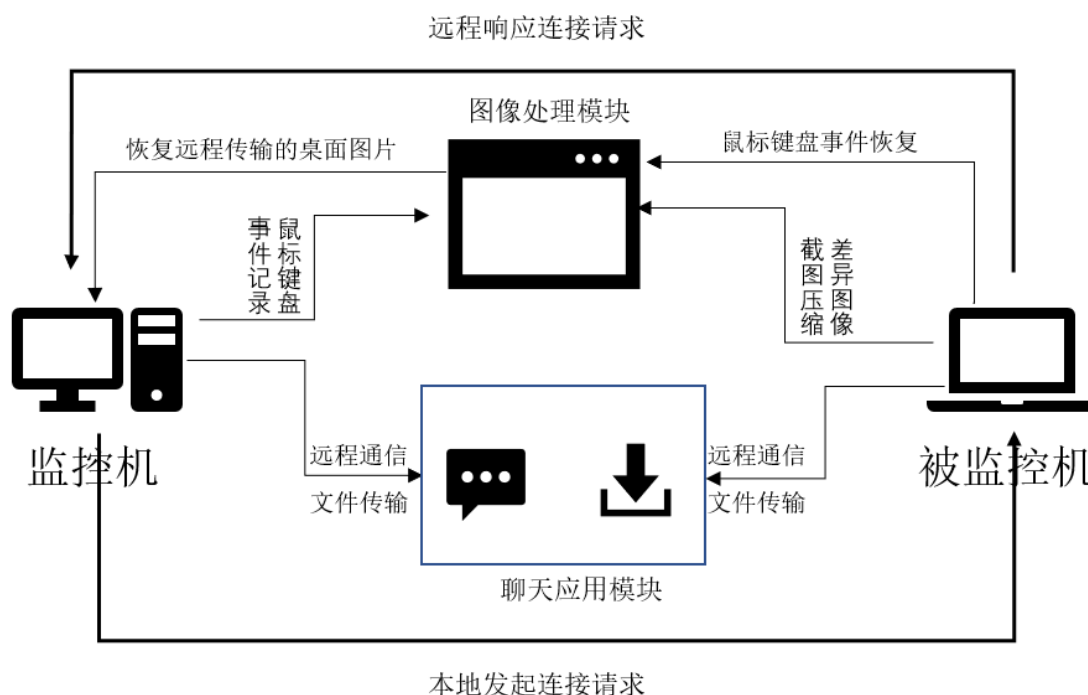


图 2 系统总体设计

## 2 系统实现

系统采用 Python 作为开发语言，并结合多个 Python 库和第三方工具实现各项功能：如 PIL 库（Python Imaging Library）用于屏幕截图的获取和处理，Socket 库实现主控端和被控端之间的数据通信和屏幕数据传输，PyAutoGUI 库用于捕获和模拟鼠标键盘事件，Paramiko 库用于 SSH 协议的实现，支持文件的上传和下载，Tkinter 库创建图形化用户界面，方便用户操作和控制。自定义 `_keyboard.py` 用于 `from _keyboard import getKeyboardMapping` 导入键盘映

射获取函数。

## 2.1 屏幕截图和编码

屏幕截图和编码是远程控制系统中关键的两个步骤，用于实现屏幕监控和图像传输。屏幕截图是从计算机屏幕上捕获当前显示内容的过程。在 Python 中，我们使用 Pillow 或者其前身 PIL 库的 ImageGrab 模块来实现屏幕截图。这个模块可以捕获整个屏幕或者指定区域的图像数据，并将其转换为 PIL Image 对象或者 NumPy 数组。屏幕截图的目的是获取当前屏幕上的视觉内容，以便后续对其进行处理和传输。一旦获得了屏幕截图的数据，接下来的步骤是将其编码为适合传输的格式。常见的编码格式包括 JPEG、PNG 等，具体选择取决于带宽和传输效率的要求。编码的目的是将图像数据转换为字节流的形式，以便通过网络传输到远程控制的客户端。编码过程通常涉及使用图像处理库（如 Pillow 或 OpenCV）将 PIL Image 对象或 NumPy 数组转换为特定格式的字节流，以便在传输过程中保持图像的完整性和质量。在实际的远程控制系统中，屏幕截图可能会根据需求定期或事件驱动地进行。编码过程不仅仅是将图像数据打包为字节流，还可以包括压缩和优化图像以减少传输延迟和带宽占用。本课设使用 JPEG 格式编码并调整质量参数来权衡图像质量和传输效率，编码后的新图像字节流转换为 NumPy 数组，之后使用 OpenCV 解码新的图像字节流为 RGB 颜色空间的图像，计算图像差异，即当前图像与上一帧图像的异或操作，如果有变化更新全局变量中的编码后的图像字节流为新截图的字节流，更新全局变量中的图像为新截图的图像，使用 OpenCV 将图像差异编码为 PNG 格式的图像字节流，之后计算原编码图像和差异化图像的大小，如果原编码图像的大小大于差异化图像的大小，构造消息长度信息，加入前导 0 表示差异化图像，发送差异化图像的字节流，如果小于则构造消息长度信息，加入前导 1 表示编码图像发送编码后的图像字节流。通过计算仅传输发生变化的部分，以减少数据传输量和网络带宽消耗。

## 2.2 远程控制

首先控制端通过套接字与服务器通信，本课设默认为 80 端口通信，使用 `SetSocket()` 函数设置套接字连接并发送平台信息（PLAT），服务器端根据接收

到的平台信息来确定键盘映射。之后开启桌面共享，通过套接字接收图像数据，使用 `cv2.imdecode()` 将数据解码为图像格式，将 OpenCV 格式的图像转换为 tkinter 可用的格式，用于在 GUI 中显示实时显示。同时，通过事件绑定函数 `BindEvents` 处理用户输入（鼠标和键盘事件），将用户操作发送到远程服务器。使用 `BindEvents(canvas)` 函数将鼠标和键盘事件绑定到 tkinter 画布上。鼠标事件包括左键按下释放、右键按下释放和滚轮滚动事件。根据不同平台（Windows/macOS 和 Linux）使用不同的事件绑定方式。键盘事件用于处理键盘按键的按下和释放事件。鼠标移动事件用于每隔一定时间（IDLE）发送鼠标当前位置的信息。被控制端通过从连接的套接字对象 `conn` 中接收控制数据，并在本地还原操作。根据存储键盘映射关系转为对应命令，根据接收到的控制命令执行对应的操作，包括鼠标移动、鼠标按键操作（左键、右键）、滚轮事件和键盘事件。控制端使用 `cv2.resize()` 调整图像大小，并将其转换为 tkinter 可用的格式进行显示，还可以随时捕获可能发生的异常，如套接字连接中断或图像解码错误，关闭显示窗口并重新启动显示功能。

## 2.3 内网穿透

内网穿透是一种技术，用于解决位于 NAT（网络地址转换）后面的设备或计算机无法直接被外部网络访问的问题。在企业和个人网络中，这种情况经常发生，限制了对内部服务或数据的远程访问。内网穿透通过一些技术手段，使得位于局域网内部的服务或数据可以通过互联网被访问到。主要应用于由于 NAT 或防火墙等网络设备的存在，内网中的计算机无法直接被公网 IP 访问。内网穿透允许用户或管理员远程访问公司内部的服务、设备或文件，无需直接连接到公司网络。可以将本地开发的服务暴露到公网，方便客户或用户访问，便于在团队或个人之间共享文件、资源或应用程序，即使他们不在同一个物理位置。内网穿透技术虽然便利，但也存在一些安全隐患，如未经身份验证的访问可能导致未授权的访问、传输数据可能会被篡改或窃取、不当配置可能会导致网络服务受到拒绝服务攻击等。通过 VPN（虚拟专用网络）技术来实现内网穿透是一种安全且方便的方式，本课设基于一款成熟的代理服务 Novpn 来实现内网穿透，Novpn 的 VPN 服务可以加密网络连接，保护通信免受窃听、干扰和数据泄漏的威胁。VPN 可以隐藏用



户的真实 IP 地址，防止他人跟踪在线活动，增强个人隐私保护。并且用户只需注册代理编码并下载客户端和服务端，即可快速安全地连接到 VPN 服务，操作简单高效。

## 2.4 远程通信

当我们想要实现监控机和被监控机之间的通信时，首先控制端开启通信，此时会打开聊天应用窗口，并发送开始通信请求到被控制端，被控制端接收到后则开启通信，同时也打开聊天应用窗口。双方通过建立新的套接字开启通信线程，本课设默认聊天端口为 12345，可以避免干扰远程桌面的传输，造成端口的占用。双方通过 UDP 协议发送消息，使用的套接字是 `socket.socket(socket.AF_INET, socket.SOCK_DGRAM)`，这种选择基于 UDP 的无连接特性，适合实时性要求较高的场景。UDP 的特点是快速且轻量，适合短消息的快速发送，但不保证可靠性和顺序性。接收消息同样使用 UDP 协议进行消息接收，绑定在所有地址的特定端口上，通过 `sock.bind(('0.0.0.0', 12345))` 监听来自任何发送者的消息。接收到消息后，解码并显示发送者的地址及消息内容。建立通信后还可以进行远程文件的下载与传输，为了不干扰通信线程，本课设对于文件传输建立新的套接字，端口默认为 54321。文件传输的实现基于 TCP 协议，TCP 提供了可靠的、面向连接的数据流传输，适合大文件传输和要求完整性的场景。发送文件使用 `socket.socket(socket.AF_INET, socket.SOCK_STREAM)` 创建 TCP 套接字，连接到指定的控制 IP 和端口。通过循环读取文件内容并使用 `sock.sendall(data)` 发送，保证文件内容完整传输。接收文件创建 TCP 监听套接字并绑定到本地地址和指定端口，使用 `sock_file.listen(1)` 监听连接请求。接受到连接后，询问用户保存文件的路径，然后循环接收数据直至接收完整文件内容。对于文件传输通过 `filedialog.askopenfilename()` 实现对文件地址的选择，从而可以下载特定的文件，对于文件保存 `filedialog.asksaveasfilename()` 实现对文件保存路径的选择，完成上传文件到指定的地址。当然也可以事先约定好文件传输地址，本课设为了实现更便捷、可以自由选择文件传输和保存，未事先确定具体地址。

## 2.5 可视化界面

可视化界面在远程通信和文件传输系统中扮演着至关重要的角色，它不仅仅是用户与系统交互的窗口，更是提升用户体验、简化操作流程的关键。Tkinter 是一个强大且易于使用的库，用于创建 Python 图形用户界面 (GUI) 应用程序。它基于 Tk GUI 工具包，可以在 Windows、Mac OSX 和 Unix 系统上使用。本课程设计在控制端主要实现主控端界面、聊天应用界面、远程桌面界面，聊天应用界面和远程桌面界面根据主控端界面的选择进行打开。被控制端只有聊天应用界面，根据主控端是否进行聊天通信决定是否打开。



图 3 主控端界面

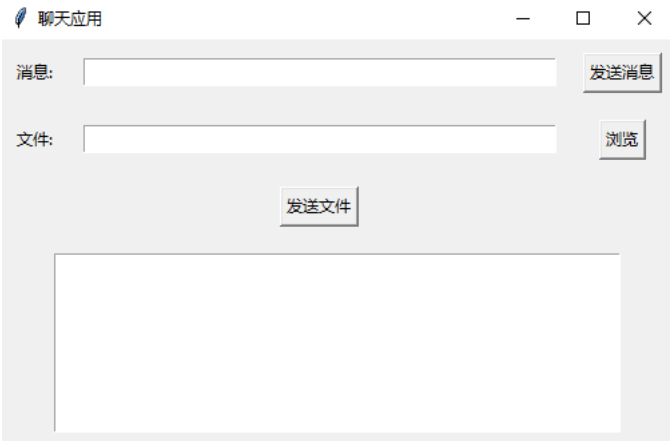


图 4 聊天应用界面

### 3 关键技术

#### 3.1 控制端

初始化 socket。对主机 ip 进行打包，变成长度为 4 的数据。开始一个新的线程。

```

global soc, host_en # 声明全局变量 soc 和 host_en

def byip4(ip, port):
    return struct.pack(">BBBBBBBH", 5, 1, 0, 1, ip[0], ip[1], ip[2], ip[3], port)

def byhost(host, port):
    d = struct.pack(">BBBB", 5, 1, 0, 3) # SOCKS5 协议版本号和命令类型
    blen = len(host)
    d += struct.pack(">B", blen) # 域名长度
    d += host.encode() # 域名编码
    d += struct.pack(">H", port) # 端口号
    return d

host = host_en.get()

```

创建了一个基于 Canvas 的框，鼠标事件和键盘事件都是在这个框内执行。首先是将服务端发来的数据（图片）长度解包，判断解包的长度与缓冲区的大小。如果缓冲区大小更大，说明还可以再接受一个图片。后将图片用 unit8 进行编码，计算图片的长和宽用于构建框。

```

def run():
    global wscale, fixh, fixw, soc, showcan

    SetSocket()

    soc.sendall(PLAT)

    lenb = soc.recv(5)

    imtype, le = struct.unpack(">BI", lenb)

    imb = b""

    while le > bufsize:
        t = soc.recv(bufsize)

        imb += t

        le -= len(t)

    while le > 0:
        t = soc.recv(le)

```

```

        imb += t

        le -= len(t)

# 解码图像数据为 NumPy 数组

data = np.frombuffer(imb, dtype=np.uint8)

img = cv2.imdecode(data, cv2.IMREAD_COLOR)

# 获取图像高度和宽度

h, w, _ = img.shape

fixh, fixw = h, w

# 将图像从 OpenCV 格式转换为 tkinter 可用的格式

imsh = cv2.cvtColor(img, cv2.COLOR_RGB2RGBA)

imi = Image.fromarray(imsh)

imgTK = ImageTk.PhotoImage(image=imi)

# 创建 tkinter 画布并绑定事件处理函数

cv = tkinter.Canvas(showcan, width=w, height=h, bg="white")

cv.focus_set()

BindEvents(cv)

cv.pack()

cv.create_image(0, 0, anchor=tkinter.NW, image=imgTK)

h = int(h * scale)

w = int(w * scale)

# 循环接收并显示图像

while True:

    if wscale:

        # 根据缩放比例调整画布大小

        h = int(fixh * scale)

        w = int(fixw * scale)

        cv.config(width=w, height=h)

        wscale = False

    try:

        # 接收图像长度信息

```

```

lenb = soc.recv(5)

imtype, le = struct.unpack(">BI", lenb)

# 接收图像数据

imb = b""

while le > bufsize:

    t = soc.recv(bufsize)

    imb += t

    le -= len(t)

while le > 0:

    t = soc.recv(le)

    imb += t

    le -= len(t)

# 解码图像数据为 NumPy 数组

data = np.frombuffer(imb, dtype=np.uint8)

ims = cv2.imdecode(data, cv2.IMREAD_COLOR)

if imtype == 1:

    # 全图传输

    img = ims

else:

    # 差异图像传输，进行异或操作

    img = img ^ ims

# 将图像调整大小并转换为 tkinter 可用的格式

imt = cv2.resize(img, (w, h))

imsh = cv2.cvtColor(imt, cv2.COLOR_RGB2RGBA)

imi = Image.fromarray(imsh)

imgTK.paste(imi)

except Exception as e:

    # 发生异常时，关闭显示窗口并重新启动显示功能

    showcan = None

```

```
ShowScreen()
```

```
return
```

鼠标操作与键盘操作。这里的操作是在上述的框中进行的。当客户端按鼠标左键时，根据上下这两部分代码，服务端也会产生一个按下鼠标左键的行为。同样，客户端按鼠标右键或滚动滚轮时，在服务端也会对应相应的操作。当客户端按键盘时，服务端也会有相应的操作。如若客户端按“a”键，服务端也会是一个按了“a”键的操作。这部分是客户端的代码，会将其对应服务发送给服务端，服务端对应代码进行接收并解析协议，执行操作。

```
def BindEvents(canvas):
```

```
    global soc, scale
```

```
    def EventDo(data):
```

```
        soc.sendall(data)
```

```
    # 鼠标左键按下事件处理函数
```

```
    def LeftDown(e):
```

```
        return EventDo(struct.pack('>BBHH', 1, 100, int(e.x / scale), int(e.y / scale)))
```

```
    # 鼠标左键释放事件处理函数
```

```
    def LeftUp(e):
```

```
        return EventDo(struct.pack('>BBHH', 1, 117, int(e.x / scale), int(e.y / scale)))
```

```
    canvas.bind(sequence="<1>", func=LeftDown) # 绑定鼠标左键按下事件
```

```
    canvas.bind(sequence="<ButtonRelease-1>", func=LeftUp) # 绑定鼠标左键释放事件
```

```
    # 鼠标右键按下事件处理函数
```

```
    def RightDown(e):
```

```
        return EventDo(struct.pack('>BBHH', 3, 100, int(e.x / scale), int(e.y / scale)))
```

```
    # 鼠标右键释放事件处理函数
```

```
    def RightUp(e):
```

```
        return EventDo(struct.pack('>BBHH', 3, 117, int(e.x / scale), int(e.y / scale)))
```

```
    canvas.bind(sequence="<3>", func=RightDown) # 绑定鼠标右键按下事件
```

```
    canvas.bind(sequence="<ButtonRelease-3>", func=RightUp) # 绑定鼠标右键释放事件
```

```
    # 鼠标滚轮事件处理函数
```

```
    if PLAT == b'win' or PLAT == 'osx':
```

```
        # windows/mac
```

```

def Wheel(e):

    if e.delta < 0:

        return EventDo(struct.pack('>BBHH', 2, 0, int(e.x / scale), int(e.y / scale)))

    else:

        return EventDo(struct.pack('>BBHH', 2, 1, int(e.x / scale), int(e.y / scale)))

    canvas.bind(sequence="<MouseWheel>", func=Wheel) # 绑定鼠标滚轮事件

elif PLAT == b'x11':

    # Linux

    def WheelDown(e):

        return EventDo(struct.pack('>BBHH', 2, 0, int(e.x / scale), int(e.y / scale)))

    def WheelUp(e):

        return EventDo(struct.pack('>BBHH', 2, 1, int(e.x / scale), int(e.y / scale)))

    canvas.bind(sequence="<Button-4>", func=WheelUp) # 绑定鼠标滚轮向上滚动事件

    canvas.bind(sequence="<Button-5>", func=WheelDown) # 绑定鼠标滚轮向下滚动事件

# 鼠标移动事件处理函数,每隔 100ms 发送一次

def Move(e): #处理鼠标移动事件

    global last_send

    cu = time.time()

    if cu - last_send > IDLE:

        last_send = cu

        sx, sy = int(e.x / scale), int(e.y / scale)

        return EventDo(struct.pack('>BBHH', 4, 0, sx, sy))

    canvas.bind(sequence="<Motion>", func=Move) # 绑定鼠标移动事件

# 键盘按键按下事件处理函数

def KeyDown(e):#处理键盘按键按下事件

    return EventDo(struct.pack('>BBHH', e.keycode, 100, int(e.x / scale), int(e.y / scale)))

# 键盘按键释放事件处理函数

def KeyUp(e):#处理键盘按键释放事件

    return EventDo(struct.pack('>BBHH', e.keycode, 117, int(e.x / scale), int(e.y / scale)))

```

```
canvas.bind(sequence="<KeyPress>", func=KeyDown) # 绑定键盘按键按下事件
```

```
canvas.bind(sequence="<KeyRelease>", func=KeyUp) # 绑定键盘按键释放事件
```

主控端界面使用 tkinter 进行窗口设置，root 为主窗口，使用 tkinter.Button 函数中 command 参数设置菜单的相关功能，其他界面类似

```
val = tkinter.StringVar()
```

```
host_lab = tkinter.Label(root, text="Host:")
```

```
host_en = tkinter.Entry(root, show=None, font=('Arial', 14), textvariable=val)
```

```
sca_lab = tkinter.Label(root, text="Scale:")
```

```
sca = tkinter.Scale(root, from_=10, to=100, orient=tkinter.HORIZONTAL, length=100,  
                    showvalue=100, resolution=0.1, tickinterval=50, command=SetScale)
```

```
proxy_btn = tkinter.Button(root, text="代理", command=ShowProxy)
```

```
show_btn = tkinter.Button(root, text="连接远程桌面", command=ShowScreen)
```

```
com_btn = tkinter.Button(root, text="开启通信", command=communication)
```

```
host_lab.grid(row=0, column=0, padx=10, pady=10, ipadx=0, ipady=0)
```

```
host_en.grid(row=0, column=1, padx=0, pady=0, ipadx=40, ipady=0)
```

```
sca_lab.grid(row=1, column=0, padx=10, pady=10, ipadx=0, ipady=0)
```

```
sca.grid(row=1, column=1, padx=0, pady=0, ipadx=100, ipady=0)
```

```
proxy_btn.grid(row=2, column=0, padx=0, pady=10, ipadx=30, ipady=0)
```

```
show_btn.grid(row=2, column=1, padx=0, pady=10, ipadx=30, ipady=0)
```

```
com_btn.grid(row=2, column=2, padx=0, pady=10, ipadx=30, ipady=0)
```

```
sca.set(100)
```

聊天功能通过 ChatApp 类来实现，通过 12345 端口建立套接字通信，通过 54321 端口实现文件传输，代码省略部分，具体查看源码。

```
class ChatApp:
```

```
    def __init__(self, root):
```

```
        ...
```

```
    def send_message(self):
```

```
        message = self.message_entry.get()
```

```
        self.message_entry.delete(0, 'end') # 清空输入框
```

```
    try:
```



```

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

dest_addr = (becontrolIP, 12345) # 目标IP 地址和端口号

sock.sendto(message.encode('utf-8'), dest_addr)

message = f"{IP}发送消息: {message}\n"

self.receive_text.insert('end', message)

self.receive_text.see('end') # 滚动到最新消息处

sock.close()

except Exception as e:

    messagebox.showerror("错误", f"发送消息失败: {str(e)}")

def browse_file(self):

    file_path = filedialog.askopenfilename()

    if file_path:

        self.file_entry.delete(0, 'end')

        self.file_entry.insert(0, file_path)

def send_file(self):

    file_path = self.file_entry.get()

    if not file_path:

        messagebox.showerror("错误", "请选择要发送的文件。")

        return

    try:

        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        dest_addr = (becontrolIP, 54321) # 目标IP 地址和端口号

        sock.connect(dest_addr)

        with open(file_path, 'rb') as f:

            while True:

                data = f.read(1024)

                if not data:

                    break

                sock.sendall(data)

        sock.close()

```

```

        messagebox.showinfo("成功", "文件发送成功。")

    except Exception as e:

        messagebox.showerror("错误", f"发送文件失败: {str(e)}")

def receive_file_thread(self):

    try:

        sock_file = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        sock_file.bind(('0.0.0.0', 54321))

        sock_file.listen(1)

        conn, addr = sock_file.accept()

        file_path = filedialog.asksaveasfilename(defaultextension=".txt",
initialdir="~/Desktop")

        if file_path:

            with open(file_path, 'wb') as f:

                while True:

                    data = conn.recv(1024)

                    if not data:

                        break

                    f.write(data)

                messagebox.showinfo("成功", "文件接收成功。")

            conn.close()

            sock_file.close()

    except Exception as e:

        messagebox.showerror("错误", f"接收文件失败: {str(e)}")

def receive_message_thread(self):

    try:

        sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

        listen_addr = ('0.0.0.0', 12345) # 监听所有地址的端口 12345

        sock.bind(listen_addr)

        while True:

            data, addr = sock.recvfrom(1024)

```

```

        message = f"来自 {addr} 的消息: {data.decode('utf-8')}\n"

        self.receive_text.insert('end', message)

        self.receive_text.see('end') # 滚动到最新消息处

    sock.close()

except Exception as e:

    messagebox.showerror("错误", f"接收消息失败: {str(e)}")

```

## 3.2 被控制端

对应于客户端的键盘操作。将键盘上所有字母，数字及操作对应的地址编码与操作或值相对应。比如当客户端按“backspace”键时，通过以下函数可使服务端完成一个按“backspace”键的操作。

```

keycodeMappingWin = {

    0x08: 'backspace', # VK_BACK

    0x5B: 'super', # VK_LWIN

    0x09: 'tab', # VK_TAB

    0x0c: 'clear', # VK_CLEAR
}

```

鼠标与键盘的执行。接收客户端发来的协议，对其进行解包，获得编码地址与 op，key 值。根据 op 值与 key 值判断进行的操作是鼠标左键按下或者弹起。或是鼠标右键按下或者弹起，或是滚轮的操作或是键盘输入，或是快捷方式。进行相应模块后，根据（x，y）值将鼠标移到相应位置并进行操作。如要进行关闭某个页面的操作，（x，y）可以定位到×的位置，op 与 key 可以判断这是一个鼠标左键按下的操作，就可以完成关闭这个页面的操作。键盘输入和快捷方式与上述代码相对应，可以知道具体为什么操作，判断出来后即可执行相应操作。

```

def ctrl(conn):

    keycodeMapping = {} # 初始化键盘映射字典

    def Op(key, op, ox, oy):

        if key == 4: # 鼠标移动

            mouse.move(ox, oy)

        elif key == 1:

```

```

        if op == 100:# 左键按下

            ag.mouseDown(button=ag.LEFT)

        elif op == 117:# 左键弹起

            ag.mouseUp(button=ag.LEFT)

    elif key == 2:# 滚轮事件

        if op == 0:# 向上

            ag.scroll(-SCROLL_NUM)

        else:# 向下

            ag.scroll(SCROLL_NUM)

    elif key == 3:# 鼠标右键

        if op == 100:# 右键按下

            ag.mouseDown(button=ag.RIGHT)

        elif op == 117:# 右键弹起

            ag.mouseUp(button=ag.RIGHT)

    else:

        # 如果不是鼠标或者特定按键操作, 则执行键盘事件, 从键盘映射字典中获取对应的键
        值

        k = keycodeMapping.get(key)

        # 如果获取到了键值

        if k is not None:

            if op == 100:# 按键按下

                ag.keyDown(k)

            elif op == 117:# 按键弹起

                ag.keyUp(k)

    try:

        plat = b" # 初始化一个空字节串, 用于存储平台标识数据

        while True:# 持续接收数据, 直到接收到3 个字节为止

            plat += conn.recv(3 - len(plat))

            if len(plat) == 3:

                break # 当接收到3 个字节后, 停止接收

```

```

print("Plat:", plat.decode()) # 打印接收到的平台标识数据

keycodeMapping = getKeycodeMapping(plat) # 根据平台标识获取键盘映射

base_len = 6 # 基本命令长度为6 个字节

while True:

    cmd = b'' # 初始化一个空字节串，用于存储接收到的命令数据

    rest = base_len - 0 # 计算还需要接收的字节数

    while rest > 0:

        # 持续接收数据，直到接收到 base_len 个字节为止

        cmd += conn.recv(rest)

        rest -= len(cmd)

    key = cmd[0] # 提取命令中的按键信息

    op = cmd[1] # 提取命令中的操作码信息

    x = struct.unpack('>H', cmd[2:4])[0] # 提取命令中的 x 坐标信息

    y = struct.unpack('>H', cmd[4:6])[0] # 提取命令中的 y 坐标信息

    Op(key, op, x, y) # 调用 Op 函数执行相应的操作

except:

    return # 捕获异常并结束函数执行

```

进行图像加法。将截取的图片进行编码，将其打包并计算数据长度。将数据长度与数据都发送给客户端。计算第二张图片的大小为 I1 与两张图片的差值 I2，如果 I1>I2，则将差值图片发送给客户端，否则则直接把第二张图片发给客户端。

```

def handle(conn):

    global img, imbyt

    lock.acquire()

    if imbyt is None:

        imorg = np.asarray(ImageGrab.grab()) # 使用 PIL 库获取屏幕截图，并转换为 NumPy 数
组

        _, imbyt = cv2.imencode('.jpg', imorg, [cv2.IMWRITE_JPEG_QUALITY, IMQUALITY])

        imnp = np.asarray(imbyt, np.uint8)

        img = cv2.imdecode(imnp, cv2.IMREAD_COLOR)

    lock.release()

```

```

lenb = struct.pack(">BI", 1, len(imbyt))

conn.sendall(lenb)

conn.sendall(imbyt)

while True:

    time.sleep(IDLE)

    gb = ImageGrab.grab()

    imgnnp = np.asarray(gb)

    _, timbyt = cv2.imencode(".jpg", imgnnp, [cv2.IMWRITE_JPEG_QUALITY, IMQUALITY])

    imnp = np.asarray(timbyt, np.uint8)

    imgnew = cv2.imdecode(imnp, cv2.IMREAD_COLOR)

    imgs = imgnew ^ img

    if (imgs != 0).any():

        pass

    else:

        continue

    imbyt = timbyt

    img = imgnew

    _, imb = cv2.imencode(".png", imgs)

    l1 = len(imbyt)  # 计算原编码图像的大小

    l2 = len(imb)  # 计算差异化图像的大小

    if l1 > l2:

        lenb = struct.pack(">BI", 0, l2)

        conn.sendall(lenb)

        conn.sendall(imb)

    else:

        lenb = struct.pack(">BI", 1, l1)

        conn.sendall(lenb)

        conn.sendall(imbyt)


```

## 4 测试和运行

### 4.1 测试环境

为了安全以及方便测试，本次课设控制端和被控制端均在虚拟机上完成，已测试不在虚拟机上也可成功。虚拟机我基于 VMware 17.0pro 中的 Windows10 虚拟机实现，控制端和被控制端网络信息如下图所示。

控制端：



```
Microsoft Windows [版本 10.0.18363.592]
(c) 2019 Microsoft Corporation。保留所有权利。

C:\Users\zws>ipconfig

Windows IP 配置

以太网适配器 Ethernet0:

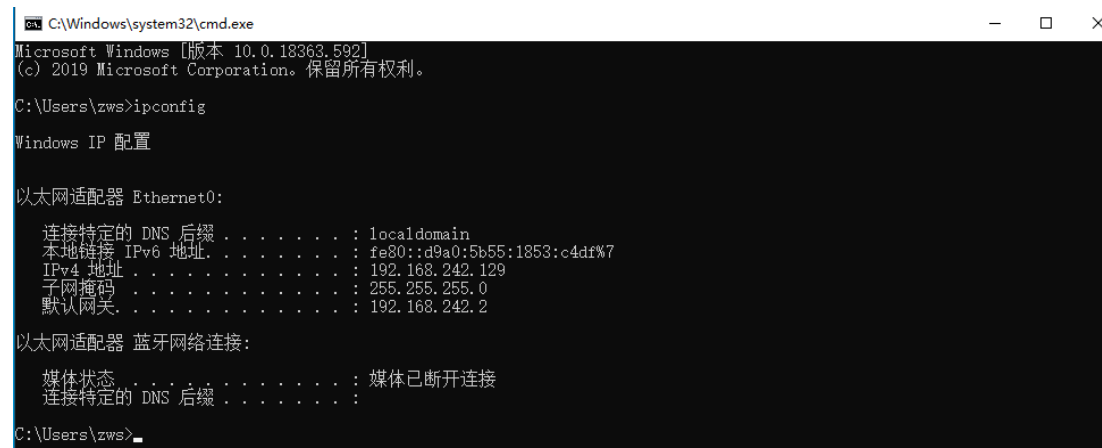
    连接特定的 DNS 后缀 . . . . . : localdomain
    本地链接 IPv6 地址. . . . . : fe80::55b0:bb2b:3443:eab%7
    IPv4 地址 . . . . . : 192.168.242.128
    子网掩码 . . . . . : 255.255.255.0
    默认网关. . . . . : 192.168.242.2

以太网适配器 蓝牙网络连接:

    媒体状态 . . . . . : 媒体已断开连接
    连接特定的 DNS 后缀 . . . . . :

C:\Users\zws>
```

被控制端：



```
Microsoft Windows [版本 10.0.18363.592]
(c) 2019 Microsoft Corporation。保留所有权利。

C:\Users\zws>ipconfig

Windows IP 配置

以太网适配器 Ethernet0:

    连接特定的 DNS 后缀 . . . . . : localdomain
    本地链接 IPv6 地址. . . . . : fe80::d9a0:5b55:1853:c4df%7
    IPv4 地址 . . . . . : 192.168.242.129
    子网掩码 . . . . . : 255.255.255.0
    默认网关. . . . . : 192.168.242.2

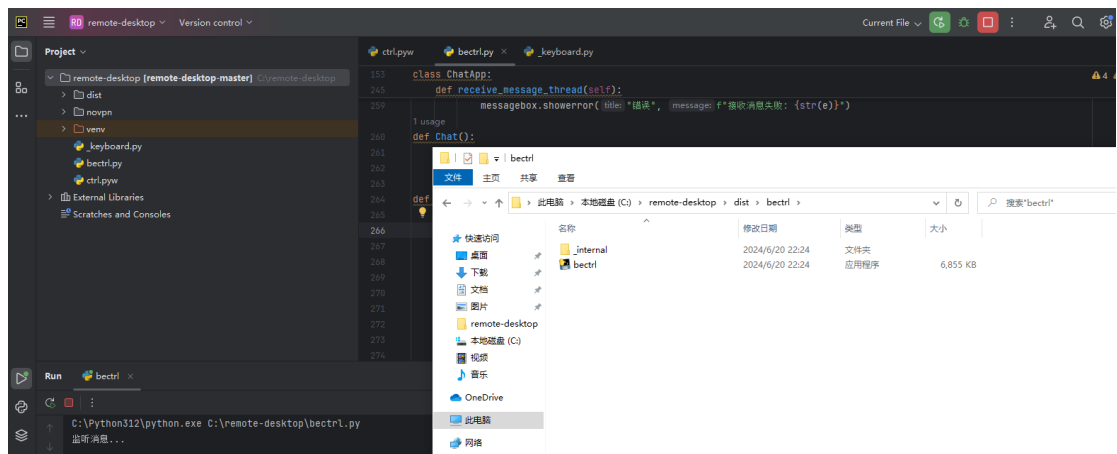
以太网适配器 蓝牙网络连接:

    媒体状态 . . . . . : 媒体已断开连接
    连接特定的 DNS 后缀 . . . . . :

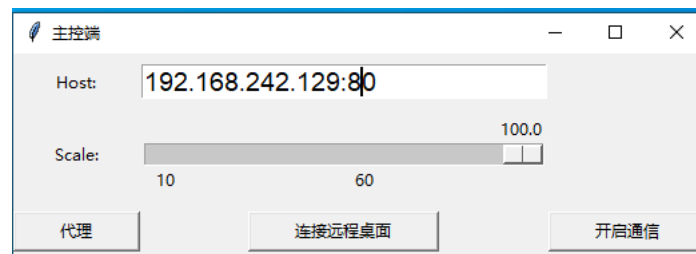
C:\Users\zws>
```

### 4.2 运行效果

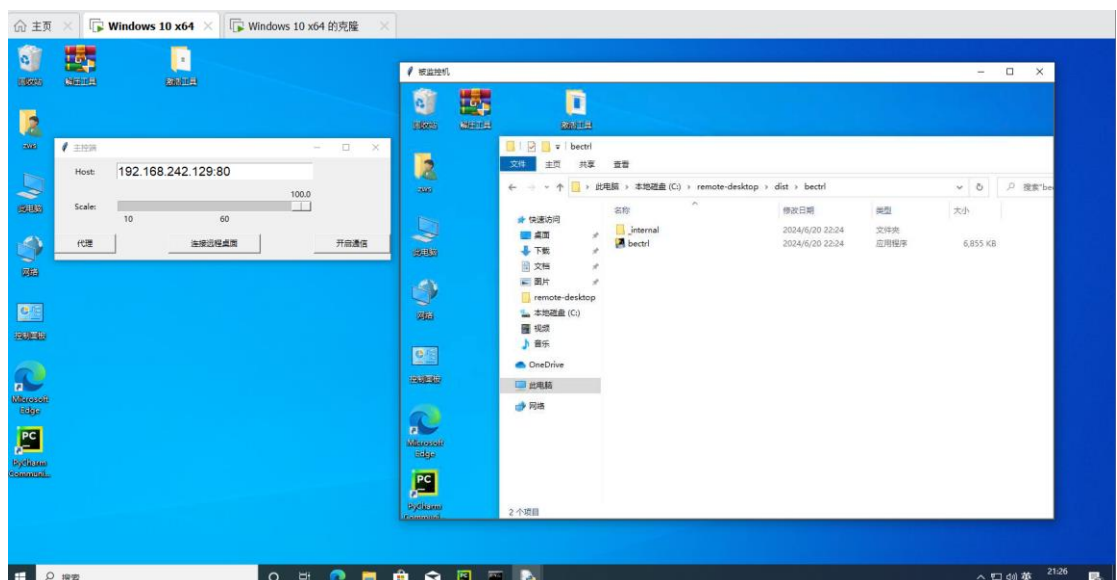
首先在被控端运行 bectrl.py, 你可以通过 python.exe 解释运行，也可以通过 pycharm 等软件运行，或者我已经将软件打包到 dist 中可直接点击运行。



之后在控制端运行 `ctrl.py`，开启主控端界面，输入被控制端 IP 和端口，默认 80 端口，之后就可以选择打开代理、连接远程桌面或者开启通信。

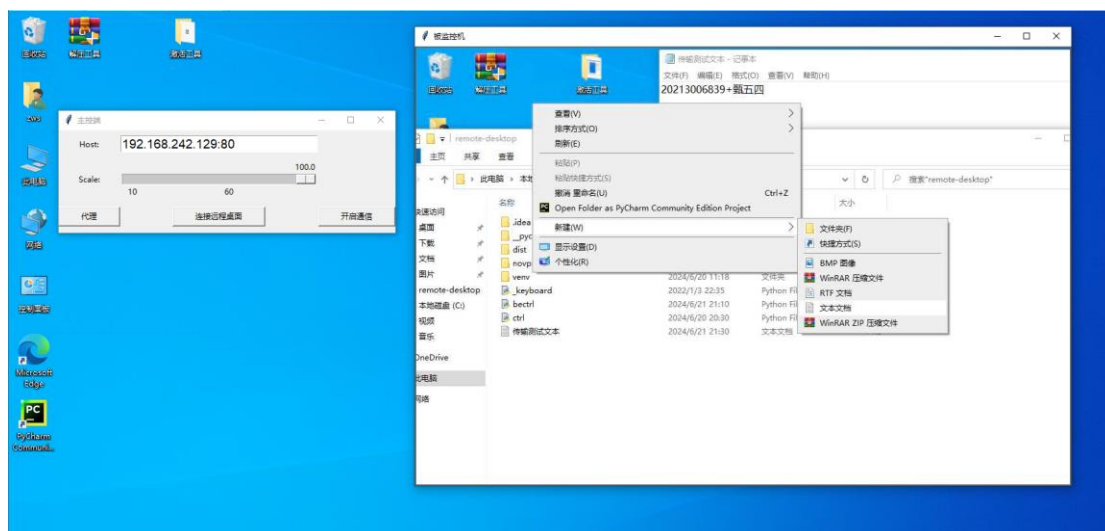


点击连接远程桌面开启远程控制桌面，可以根据需要调整被控制机窗口大小，此时可以远程对对方主机进行操作，如关闭线程，创建文件等。

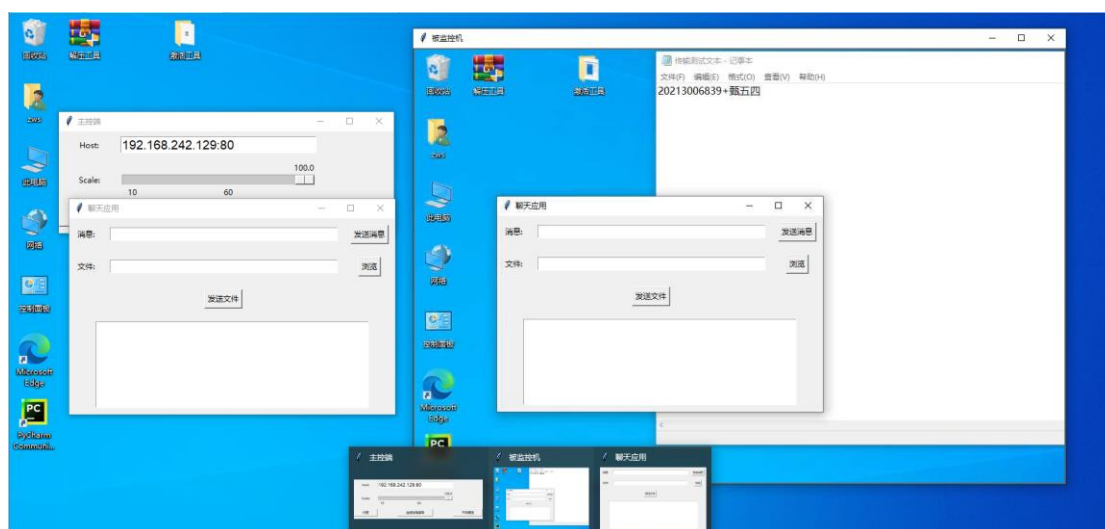




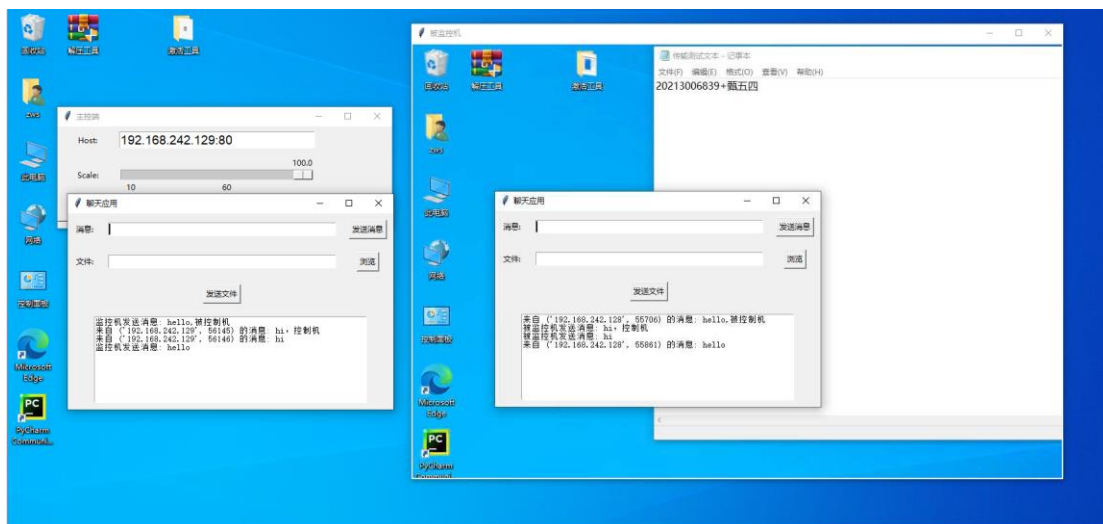
控制远程主机，建立传输测试文本.txt，输入学号+姓名



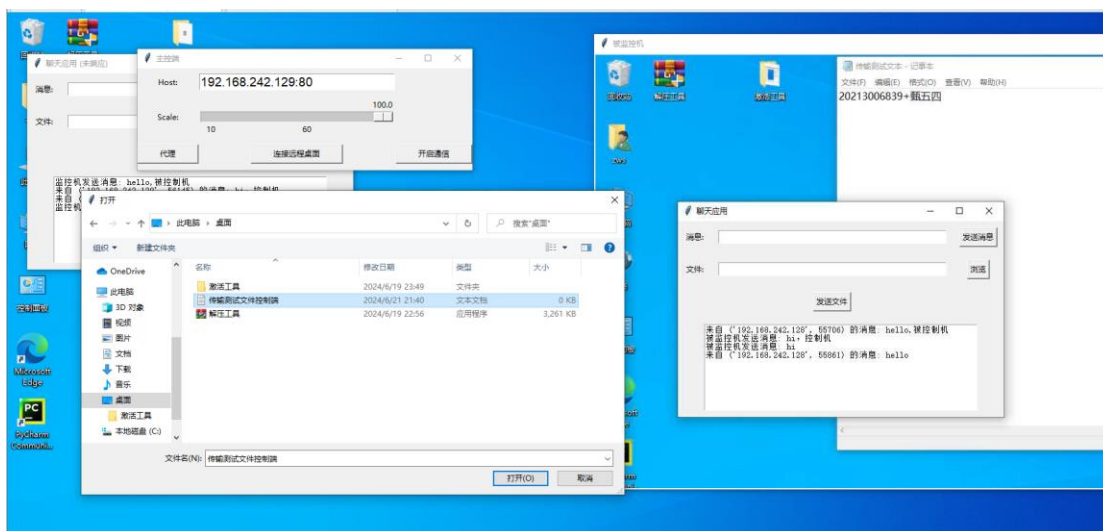
之后我们点击开启通信实现双方通信，此时被控制端也自动打开聊天应用窗口



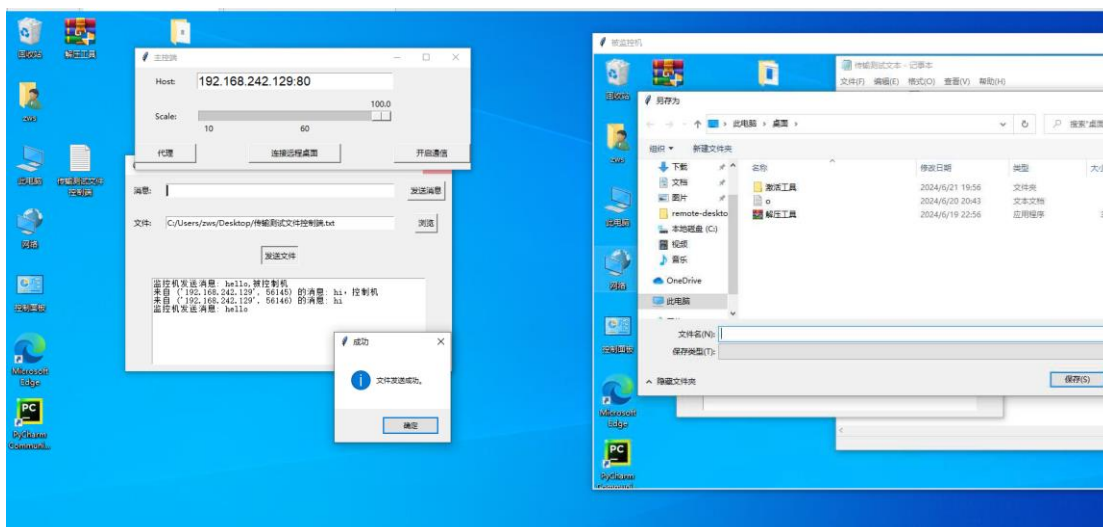
可以在消息框中输入发送消息，然后点击发送消息按钮实现发送消息，你可以在被控制端输入聊天消息实现通信，也可以在控制端的被控制端窗口中聊天应用窗口输入消息进行通信，下文文件传输也可以这样，这也体现了控制端可以远程对方主机进行操作。



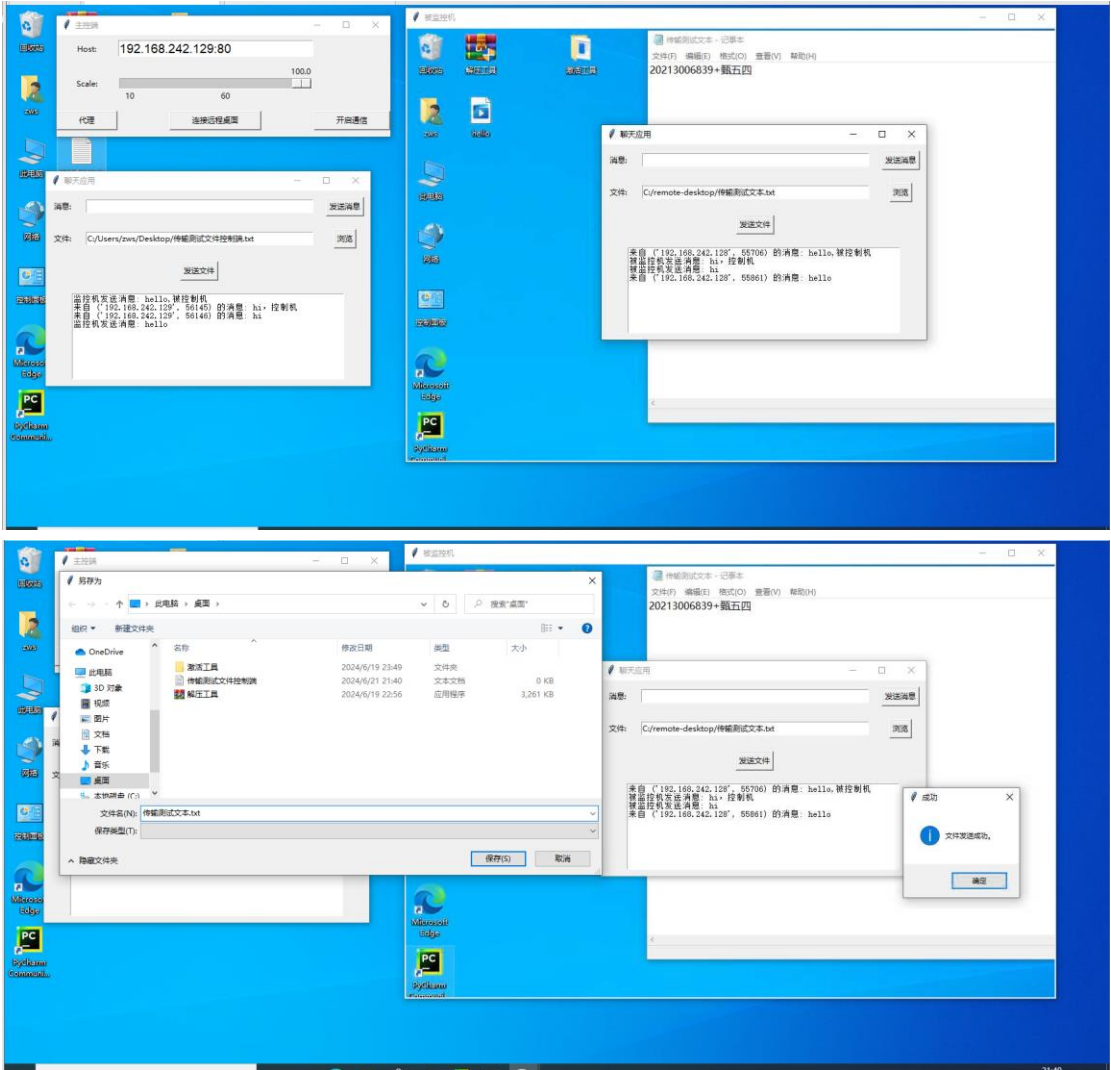
点击浏览可以选择文件，此处我在控制端选择传输测试文件控制端. txt



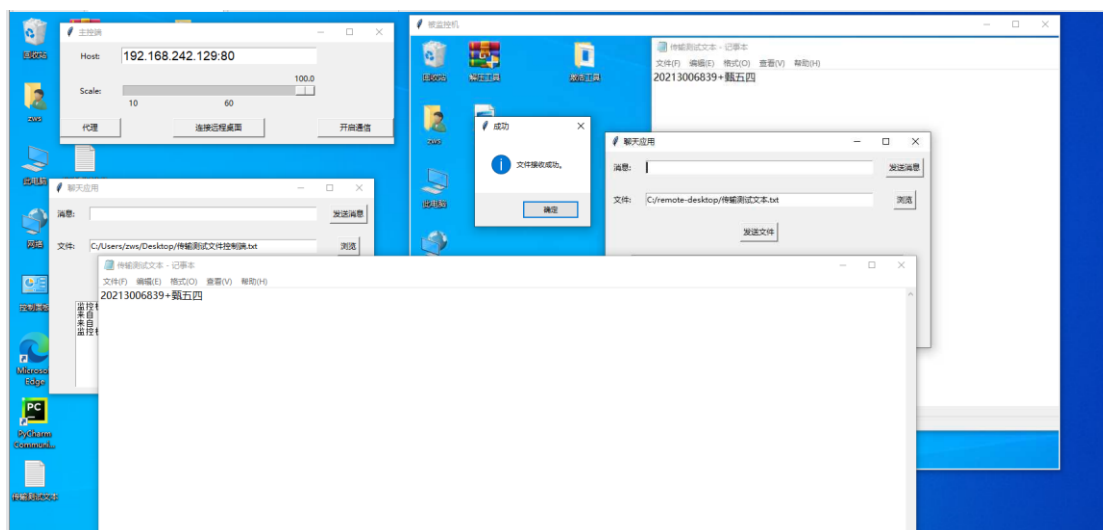
点击发送文件实现文件发送，此时被控制端接收到文件，并打开另外为窗口，可以选择保存位置以及确定文件名。



可以在控制端的被控制端窗口浏览文件发送，实现能够下载特定的文件的下载。



接收到被控制端发送的文件后保存到桌面，被控制端会显示文件接收成功，打开文件显示姓名+学会，文件传输无错误。



所有功能已成功测试并表现良好。在开始测试前，请确保在虚拟机和物理机上分别启动了 Novpn 的服务器和客户端，或者关闭了虚拟机的网络防火墙。这些步骤是为了确保网络畅通，从而顺利进行远程。

## 5 总结

本系统在设计与实现过程中，我遇到了诸多问题，比如选择合适的图形界面库和网络通信框架，确保系统稳定性和响应速度。优化界面设计和交互流程，提升用户操作的便捷性和直观性，如何实现远程通信，控制端操作如何传输到被控制端，以及被控制端如何恢复控制端操作等等。通过本次课程设计我利用截屏技术实现了监视远程主机屏幕的功能，使监控机能够实时查看被监控主机的操作界面。实现了远程控制功能，通过发送控制命令实现对被监控主机的操作，包括鼠标点击、键盘输入等。使用 UDP 和 TCP 协议实现了监控机和被监控机之间的通信，支持消息和文件的双向传输。实现了将文件从监控机上传到被监控主机指定位置的功能，以及从被监控主机下载特定文件到监控机的功能。然而由于时间和安全考虑，未实现真正的内网穿透功能。这限制了系统在复杂网络环境下的应用，需要依赖 VPN 等手段确保网络连通性。对于网络监控和远程控制系统，安全性是一个重要问题，未来需要加强数据加密和身份验证等安全机制。为了进一步完善和扩展本次课程设计，未来可以探索使用成熟的内网穿透工具或自行开发解决方案，确保系统能够在复杂网络环境中稳定运行，提升系统的实用性和普适性。引入端到端的加密通信和强化的身份验证机制，保护通信内容的机密性和系统的安

全性，防止未授权访问和数据泄露。进一步优化用户界面，提升用户操作的便捷性和友好性。增加更多的远程控制功能，如远程文件管理、远程执行命令等，满足用户对远程管理的多样化需求。还可以结合最新的技术发展，如 AI 辅助分析监控数据、实时反馈机制等，提升系统的智能化水平和用户体验，推动网络监控和管理系统向更高层次发展。通过持续的技术探索和功能优化，本次课程设计可以成为一个更加完善和安全的网络监控和管理系统，为用户提供稳定、安全、高效的远程管理解决方案。

## 参考文献

- [1] 耿庆安, 刘娜, 张镭. 基于远程桌面连接的现场设备远程控制 [J]. 电子技术应用, 2011(1):117-119.
- [2] 王建. 基于消息传递的远程桌面控制协议实现 [J]. 内江师范学院学报, 2007(2):41-45.
- [3] 谢志俊. 远程桌面最快的软件《向日葵远程控制》 [J]. 计算机与网络, 2017(8):37-37.
- [4] 郭建伟. 轻松实现远程控制 [J]. 网管员世界, 2010(2):116-117.
- [5] 王卫国. 远程控制局域网内 PC [J]. 网管员世界, 2009(19):112-112.
- [6] Muneer P, Mohammed Shareef C, Vipin K R. Remote Desktop Controller Using Android[J].2016.
- [7] Kuzminykh I, Ghita B, Silonosov A. Impact of network and host characteristics on the keystroke pattern in remote desktop sessions[J]. arXiv preprint arXiv:2012.03577, 2020.

## 附录

1. 安装说明: 首先在主控机和被控机上分别安装 novpn-vpn-client 和 novpn-service, 然后将 bectrl 文件夹放置在被控机端, ctrl 文件夹留在主控端上。我是基于 python3.12.4 完成的本次课程设计, 所需主要库有 numpy==2.0.0, opencv-python==4.10.0.84 , paramiko==3.4.0 , pillow==10.3.0 , PyAutoGUI==0.9.54 你可以通过 pip install -r requirements.txt 安装所需函数库。
2. 使用说明: 在被控机运行 novpn-service, 按注册用户登录。在主机运行 novpn-vpn-

client，确保其与服务器连接正常。可以在控制端和被控制端相互 ping 测试连通，之后先在被控制端运行 bectrl.py，再在控制端运行 ctrl.py，具体操作可以看 4.2 运行效果。

3. 参考资料：本次课程设计实验参考诸多网络开源代码以及公开资料如下，在此对开源代码资料表示感谢

[Python 实现远程控制另一台电脑 python 远程控制电脑-CSDN 博客](#)

[python 实用【大技巧】之 Python 手把手实现远程控制桌面 python 远程控制电脑-CSDN 博客](#)

[Python 实现远程控制桌面功能（不使用微信等任何接口），并在不同局域网下控制（已更新）\\_py 远程桌面实现-CSDN 博客](#)

[python 网络编程：通过 socket 实现 TCP 客户端和服务端 python tcp 客户端-CSDN 博客](#)

[解决 python 远程传输文件的具体操作步骤 mob649e8157ebce 的技术博客 51CTO 博客](#)

[Python 实现远程文件传输 192.168.101.20-CSDN 博客](#)

[https://www.bilibili.com/video/BV1e54yly7eq/?vd\\_source=6edb456df706494ac750a8dd17ec65a1](https://www.bilibili.com/video/BV1e54yly7eq/?vd_source=6edb456df706494ac750a8dd17ec65a1)

[python+socket 实现网络信息交互及文件传输 基于 socket 实现信息交互和文件上传及下载流程图-CSDN 博客](#)

[python 实现远程服务器控制和文件传输（SSH 协议） - mghhz816 - 博客园 \(cnblogs.com\)](#)

[Python——Socket 编写简单脚本木马远程控制截屏并发送截图 python socket 截屏-CSDN 博客](#)

[（python）远程操作模块-Paramiko python 远程执行模块-CSDN 博客](#)

[remote\\_desktop: 远程桌面控制-局域网-python \(gitee.com\)](#)

[开源项目推荐：基于 Qt 开发的远程桌面监控和文件传输软件 Aspia-CSDN 博客](#)

[Python 语言实现两台计算机用 TCP 协议跨局域网通信 - liyishui - 博客园 \(cnblogs.com\)](#)