

网络安全实验报告

题 目 基于 socket 的扫描器设计

专 业 信息安全

学 号 120L020121

学 生 刘旭

指 导 教 师 王彦

# 一、实验目的

熟悉socket 编程，可以利用socket 编程编写基于linux平台的 C/S程序和基于 windows 平台的扫描器。

**二、实验内容**

1. 熟悉 Linux 编程环境
2. 在Windows机器上安装Linux虚拟机
3. 在 Linux 环境下编写 C/S 程序，熟悉 socket 编程。要求客户端和服务器端能够传送指定文件。该程序在后续实验中仍需使用。客户端与服务器端在不同的机器中。
4. 在 Windows 环境下利用 socket 的 connect 函数进行扫描器的设计，要求有界面，界面能够输入扫描的 ip 范围和端口范围，和需使用的线程数，显示结果。
5. 实验课的时候，检验结果和现场截图，为撰写实验报告做准备。

# 三、实验过程

**（一）Linux 环境下的 C/S 程序**

## 实验基本信息：

实验环境：Ubuntu 20.04 x64 编程语言: Python

## 1. 需求分析

需要在两台 linux 虚拟机之间传送文件，所以需要给两台 linux 虚拟机都配置一个可以访问的 ip。

程序功能：

(1)客户端：

a.可以向服务端发送一个本目录下指定的文件，文件名由用户输入；

b.可以从服务端下载一个服务端目录下的文件，先从服务端获得文件名列表，再由用户输入需要的文件名。

(2)服务端：

a.可以监听来自客户端的连接请求；

b.可以接收客户端传送的文件；

c.可以向客户端传送一个指定的文件，文件由客户端给出。

(3)传送文件要求：任何二进制文件。

## 2. 环境配置

配置虚拟机的网卡即可：



## 3. 客户端编写

这段代码实现了一个socket编程的客户端，可以实现上传和下载文件的功能。具体编写过程如下：

* 导入所需模块和库，包括socket和os模块。
* 创建TCP socket对象，并连接到指定的服务器IP和端口。
* 使用while循环，提示用户输入指令（上传/下载/断开连接）。
* 如果用户输入的指令为“上传”，则向服务器发送上传命令，并显示当前目录下的文件列表，提示用户输入需要上传的文件名。如果文件存在，则计算文件大小并发送文件名和大小给服务端。等待接受服务器发回的确认文件大小和名称的信息，然后打开文件并发送给服务端。最后显示上传完成信息或者上传失败信息。
* 如果用户输入的指令为“下载”，则向服务器发送下载命令，并显示服务端当前目录下的文件列表，提示用户输入需要下载的文件名。发送需要下载的文件名给服务端，并等待接收服务端发送的文件名和大小。如果文件不存在，则显示错误信息，否则打开文件，从服务端接收数据并写入文件，完成文件下载。
* 如果用户输入的指令为“断开连接”，则向服务器发送断开连接命令，关闭客户端socket，程序结束。
* 如果用户输入的指令不是“上传”、“下载”或“断开连接”，则提示用户输入有效指令。

总的来说，这段代码实现了基本的客户端上传下载文件的功能，但也存在一些需要优化的地方，例如对于异常情况的处理不够完善，需要加入更多的异常处理代码。

## 4. 服务端编写

**创建TCP socket对象**

使用socket模块中的socket函数创建TCP socket对象，使用IPv4地址族和流式传输方式。

**设置socket地址重用选项**

使用setsockopt函数设置socket地址重用选项，以便在服务器socket关闭后，能够立即重新使用该地址。

**绑定IP地址和端口号**

使用bind函数将服务器端口号与IP地址绑定。

**开始监听连接请求**

使用listen函数开始监听连接请求，并设置请求队列的最大长度。

**接受客户端连接请求**

使用accept函数等待客户端连接，并返回新的socket对象和客户端地址信息。

**接收客户端请求**

使用recv函数接收客户端请求，并使用decode函数将请求转换为字符串。

**根据请求类型处理请求**

根据客户端请求类型，处理上传、下载、关闭请求。如果客户端请求上传文件，则接收文件名和文件大小，准备打开文件并写入数据；如果客户端请求下载文件，则获取当前目录下的文件列表，将文件列表发送给客户端，并等待接收客户端请求的文件名。如果文件存在，则发送文件给客户端，否则返回文件不存在的错误信息。如果客户端请求关闭连接，则关闭客户端socket对象。

**文件传输**

如果客户端请求上传文件，则使用recv函数接收客户端发送的数据，并使用write函数将数据写入文件中。在接收完所有数据后，关闭文件。如果客户端请求下载文件，则使用read函数从文件中读取数据，并使用send函数将数据发送给客户端，直到文件的所有数据都被发送完毕。

**关闭socket**

使用close函数关闭服务器socket对象，以便释放端口号和相关资源。

**（二）Windows 环境下的扫描器程序**

## 实验基本信息：

实验环境：Windows10 x64

PyQt6 编程语言：Python

## 1. 需求分析

实验指导中要求编写界面，我选择用PyQt6。且Qt独有的信号与槽机制能使很多操作变得方 便。另外，在程序的设计各方面都追求人性化，用户误操作时会给出准确的提示信息。

程序功能：

1. 用户可以输入需要扫描的 ip 范围、端口范围和想使用的线程数，其中 ip 范围分为四个小框，每个框中的可输入的范围为0-255。默认 ip 地址范围设置为 127.0.0.1-127.0.0.1。
2. 如果用户在输入未完成的时候就按下了开始扫描按钮，提示输入未完成，如果用户输入的范围错误，提示范围错误；
3. 当所有输入都正确无误后，按下开始扫描，程序开始扫描用户指定的 ip 和端口；
4. 关于扫描的线程分配：

扫描线程是通过scan\_ip函数和scan\_thread函数实现的。在start\_scan函数中，首先获取用户输入的参数，然后计算IP地址列表。对于每个IP地址，调用scan\_ip函数启动多个线程扫描端口。在scan\_ip函数中，根据num\_threads计算每个线程应该扫描的端口数，并为每个线程创建一个threading.Thread对象。然后，对于每个线程，调用start方法启动线程，并使用join方法等待线程完成。在scan\_thread函数中，对于指定IP地址和端口范围，循环遍历每个端口并调用scan\_port函数扫描端口。scan\_port函数创建一个TCP socket并使用connect\_ex方法尝试连接到指定的IP地址和端口。如果连接成功，则将该端口标记为开放，并将结果添加到self.results列表中。

本着用户友好原则，在扫描过程中打印所有的扫描结果，但因为多线程的原因，扫描出的顺序是乱的，所以在扫描结束后单独打印出开启的端口号，并且打印此次扫描花费的时间、扫描的总端口数以及开启的端口数。

(6) 用户可以在扫描正在进行时按结束扫描的按钮来中断扫描，点击按钮后会跳出对话框确认以防止用户误点，当程序收到结束扫描的信号时会中断所有线程，这一过程是安全的。

## 2. 界面编写

实现了一个简单的GUI页面，包含了以下功能：

* IP地址范围和端口范围的输入框
* 线程数设置输入框
* 开始扫描和停止扫描按钮
* 扫描结果显示框
* 对输入的端口范围进行验证，只接受1-65535的整数
* 对线程数进行验证，只接受1-9的整数

这个页面的整体布局使用了QVBoxLayout和QHBoxLayout布局管理器，将所有部件垂直和水平排列。其中，IP地址范围的输入框和端口范围的输入框使用了QLineEdit控件，线程数设置的输入框同样使用了QLineEdit控件，对于输入内容的验证使用了QIntValidator验证器。开始扫描和停止扫描按钮则使用了QPushButton控件。扫描结果的显示框则使用了QTextEdit控件，并将其设置为只读。最后，通过setLayout()方法将整个布局添加到MainWindow对象中，设置窗口的标题并显示出来。

## 3. 控件逻辑编写

GUI部分代码主要是通过 PyQt5 库创建各种控件来实现用户界面的编写。其中包括：

* IPInputBox 控件：用于用户输入 IP 地址范围。该控件由两个 QLineEdit 组成，分别用于输入起始和结束 IP 地址。
* QLineEdit 控件：用于用户输入端口范围和线程数。
* QPushButton 控件：用于启动和停止端口扫描。
* QTextEdit 控件：用于显示扫描结果。

这些控件之间的逻辑关系是通过 QVBoxLayout 和 QHBoxLayout 进行排列和组合的。其中 QVBoxLayout 控件用于实现垂直排列，而 QHBoxLayout 控件用于实现水平排列。通过 addLayout() 和 addWidget() 方法将各种控件添加到 QVBoxLayout 和 QHBoxLayout 中，最终实现了一个整体的用户界面。

通过 clicked.connect() 方法将 QPushButton 控件的点击事件与 start\_scan() 和 stop\_scan() 两个方法进行了绑定。这两个方法是在 MainWindow 类中自定义的，用于启动和停止端口扫描的操作。

## 4. 具体功能编写

1. 扫描主方法 start\_scan(self)

函数start\_scan()的功能如下：

* 禁用开始扫描按钮。
* 获取用户输入的参数，包括起始IP地址、结束IP地址、起始端口号、结束端口号和线程数。
* 计算IP地址列表。
* 清空扫描结果框并显示"Scanning..."。
* 初始化扫描结果列表。
* 对IP地址列表中的每个IP启动多个线程扫描端口。
* 等待所有线程完成后，显示扫描结果。
* 启用开始扫描按钮。

在具体实现过程中，该函数首先禁用了开始扫描按钮，然后获取了用户输入的参数，并计算出要扫描的IP地址列表。接着，该函数清空了扫描结果框并显示"Scanning..."，然后初始化了扫描结果列表。接下来，该函数对IP地址列表中的每个IP启动了多个线程扫描端口。最后，该函数等待所有线程完成后，显示扫描结果，并启用开始扫描按钮。

1. 扫描线程方法 scan\_thread(self, ip, start\_port, end\_port)

start\_thread() 函数实现了在指定IP地址范围内启动多个线程扫描端口的功能。

具体实现步骤如下：

* 计算每个线程需要扫描的端口范围，根据线程数将端口范围划分为多个子范围。
* 创建多个线程，并将每个线程分配到对应的子范围内扫描端口。
* 启动所有线程并等待所有线程执行完毕。
* 在每个线程中调用 scan\_port() 函数扫描指定IP地址和端口。

在这个函数中，通过调用 scan\_thread() 函数来扫描指定IP地址和端口范围。scan\_thread() 函数会在每个线程中执行，从而实现多线程扫描端口的功能。

在扫描完成后，通过调用 show\_results() 函数显示扫描结果。

(3) 向 TextEdit 打印扫描结果 show\_results(self)

show\_results()函数的功能是展示扫描结果。具体实现如下：

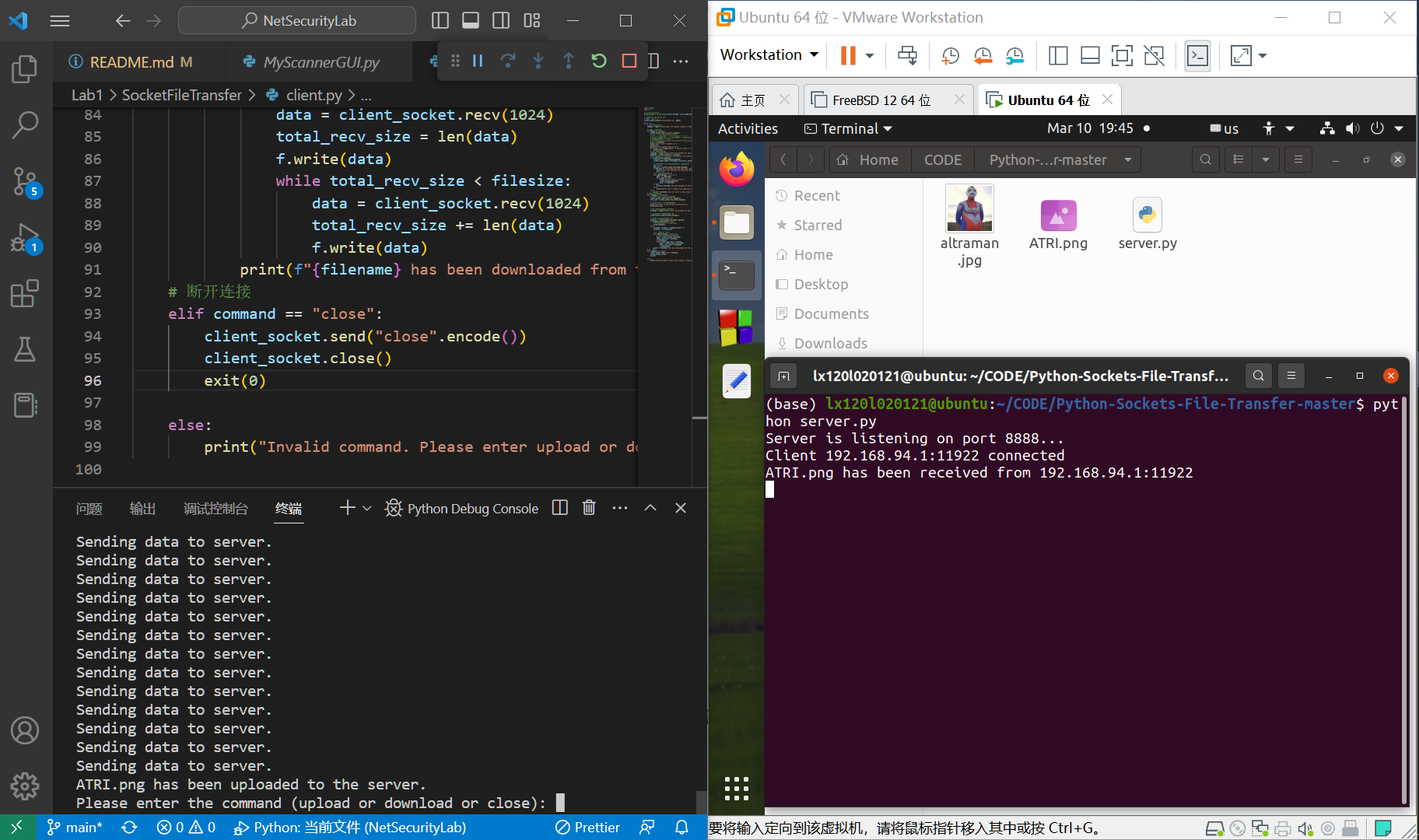
* 对扫描结果按照端口号进行排序，调用list.sort()方法并传递一个lambda表达式作为key参数，lambda表达式返回的是一个元组，包含了IP地址和端口号。
* 在结果框中显示扫描用时和扫描的端口数，根据self.start\_time记录的开始扫描时间和用户输入的开始和结束端口号计算出扫描的端口数，然后将这些信息显示在结果框中。
* 如果有端口开放，将所有开放的端口和它们对应的IP地址显示在结果框中，并显示开放的端口数。
* 如果没有开放的端口，将显示"No open ports found."。

因此，show\_results()函数主要实现了结果的展示和信息的统计。

# 四、实验结果

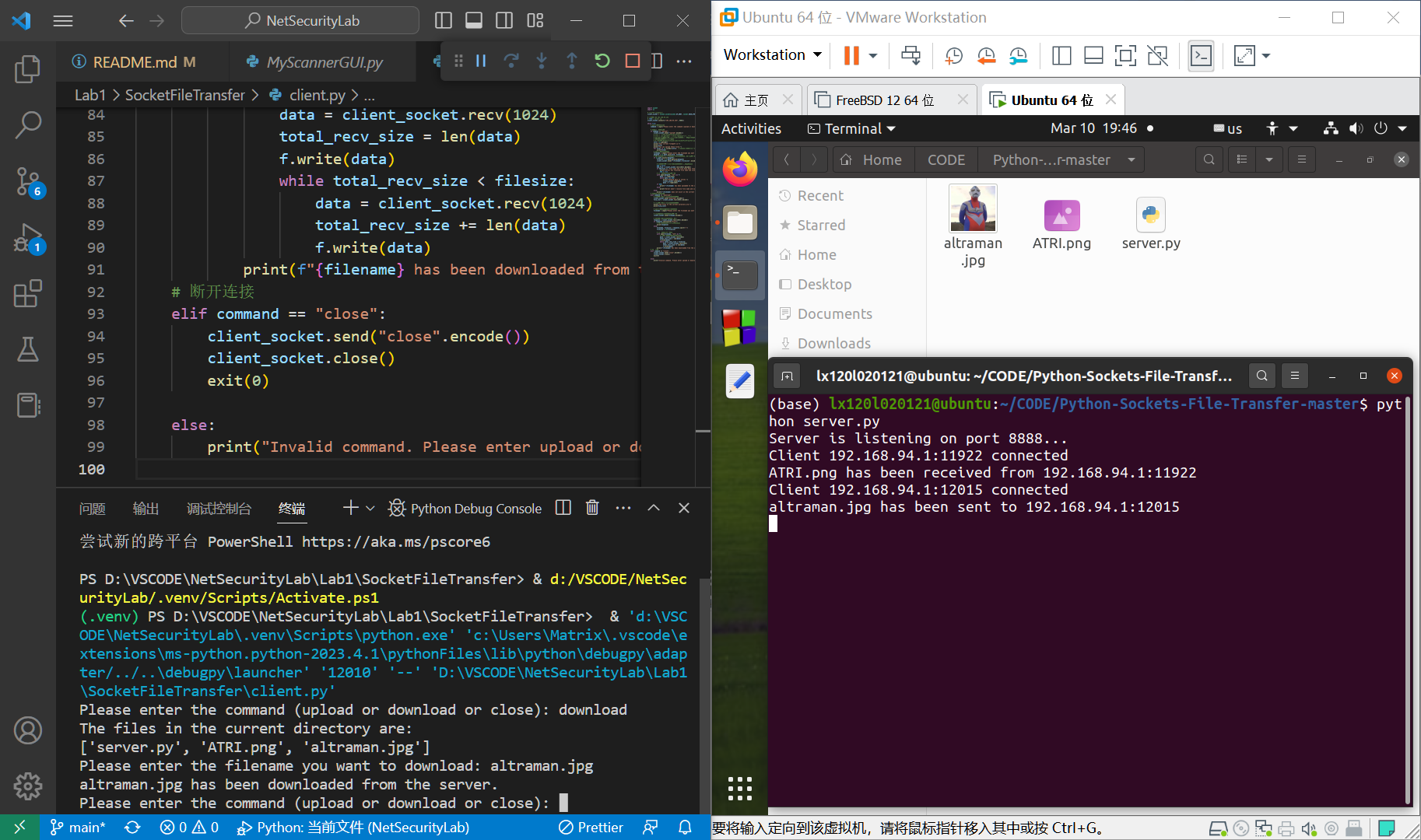
1. **Linux 环境下的 C/S 程序**

上传文件到服务器：



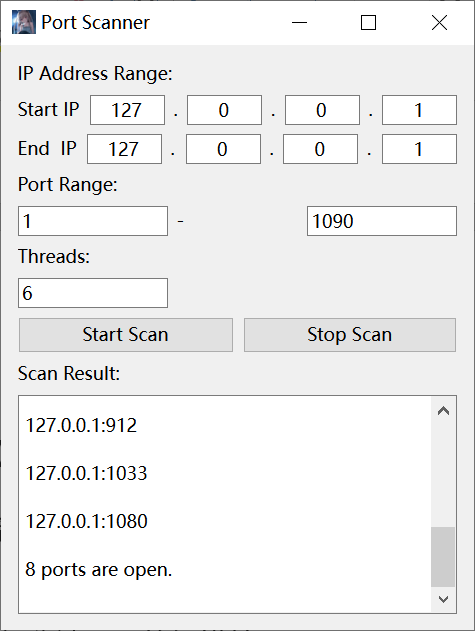
图中左边为客户端，右边为服务器端。

从服务器下载文件：



1. **Windows 环境下的扫描器程序**

此为本机部分端口号的扫描结果：



此为百度部分端口号的扫描结果：

