**项目设计报告**

题 目 端口扫描器的设计与实现

学生姓名 毛搴舟

日期：2023年6月22日

**摘要**

开放在主机和服务器上端口是通信的通道,是交流的“门户”，也是一个方便攻击者入侵“裂口”。对目标主机进行端口扫描,能帮助用户判断目标主机上服务开启与否的状态，还能一定程度上用来判断目标主机的操作系统型号版本等等。鉴于进行端口扫描能得到如此多的有用信息，本次项目设计选择开发一个端口扫描器。

该端口扫描器基于Python语言，使用Pycharm环境进行设计开发，能够实现以下基本功能和任务：可以对指定的单个主机进行扫描；可以对指定的网段内的主机进行扫描；可以对指定的部分端口号进行扫描；可以对指定的端口段内的端口进行扫描。

最终实现了基于threading多线程的简易端口扫描器，使用argparse命令行参数，能够让用户根据使用手册提示输入指定参数，对于错误的命令，程序可以及时检测到并终止运行，很好地实现了目标功能。

**关键词：**TCP/UDP端口扫描；port status端口状态检测；queue队列结合threading库多线程使用；socket库函数的应用

目录

[1.有关端口扫描的基本知识 5](#_Toc19842)

[1.1端口的基本概念 5](#_Toc24724)

[1.2端口的分类 5](#_Toc18494)

[1.3扫描方式的分类 6](#_Toc32713)

[2.端口扫描的基本原理 6](#_Toc23040)

[2.1基本原理 6](#_Toc9094)

[2.2端口扫描的设计流程图 8](#_Toc28259)

[3.基于Python的端口扫描器的核心代码 9](#_Toc18352)

[3.1 Python语言的优势与特点 9](#_Toc15756)

[3.2端口扫描的核心代码 9](#_Toc2150)

[4.端口扫描的设计优化 16](#_Toc19837)

[4.1多线程扫描 16](#_Toc19573)

[4.2端口服务名称获取 20](#_Toc26436)

[4.3常用端口列表获取 21](#_Toc2147)

[4.4端口合法性判断和IP地址合法性判断 24](#_Toc26191)

[4.5用户命令获取 25](#_Toc1758)

[5.端口扫描的实现效果 27](#_Toc15273)

[项目设计心得体会 31](#_Toc18832)

[参考文献 32](#_Toc3062)

**端口扫描器的设计与实现**

# 1.有关端口扫描的基本知识

## 1.1端口的基本概念

“端口”，即“服务端口”，是专门为计算机通信而设计的非硬件逻辑概念，不同于计算机中的真实存在于主板上的“插槽”，端口是个逻辑意义上的“软插槽”。一台计算机中的端口通过16bit进行编号，最多可以有65536个端口。端口是通过端口号来标记的，端口号只有整数，范围从0 到65535。

端口是由计算机的通信协议TCP/IP协议定义的。其中规定，用IP地址和端口作为套接字，它代表TCP连接的一个连接端，一般称为Socket。具体来说，就是用IP和端口来定位一台主机中的进程。例如IP+80端口常对应主机的HTTP服务

可见，如果某个进程正在等待连接，称之为该进程处于LISTENING状态，此时主机上与它相对应的端口出现。显而易见，通过目标主机端口状态便可以判断出目标主机有哪些进程是正在等待连接的。

## 1.2端口的分类

端口是一个16 bit的地址，用端口号进行标识不同作用的端口。端口一般分为以下两类：

（1）常用端口号：范围从0到1023，一般情况下，这些端口号固定分配给一些服务。例如

80：HTTP（HyperText Transfer Protocol，超文本传输协议）

135：RPC协议（Remote Procedure Call，远程过程调用）并提供DCOM（分布式组件对象模型）服务，极易受到DOS攻击

21：FTP（File Transfer Protocol，文件传输协议）代理服务器常用端口

23：Telnet（远程登录协议）协议代理服务器常用端口号

443：HTTPS（Hypertext Transfer Protocol Secure，超文本传输安全协议）服务器，是HTTP协议的安全版本

69：TFTP（Trivial File Transfer Protocol，简单文件传输协议），是TCP/IP中的一个用来在客户机与服务器之间进行简单文件传输的协议，提供不复杂、开销不大的文件传输服务。

（2）动态端口号：动态端口的范围从1024到65535，一般情况下，这些端口号不固定分配给某个服务。只要运行的程序向计算机系统提出申请，那么系统就可以给该程序分配一个端口。例如8080端口常分配给第一个向系统发出申请的程序。此程序进程结束时，所占用的端口号就会被释放。

## 1.3扫描方式的分类

（1） 全TCP连接，Python中socket.connect()和socket.connect\_ex()函数即使用此方法，通过三次握手与目标主机建立标准的TCP连接。此方法速度快、效率高、可靠传输有重传机制，但存在丢包可能，且极容易因为多次重传被目标主机防火墙发现并记录。  
  （2）SYN扫描，扫描主机自动向目标主机指定端口发送SYN数据段，请求建立连接。若目标主机的回应报文SYN=1，ACK=1，表示目标主机同意建立连接，则认为该端口是开启的，扫描主机再发送一个RST给目标主机，表示拒绝连接。导致三次握手失败。若目标主机回应RST则认为端口是关闭的。  
  （3）FIN扫描，发送一个FIN=1的报文到一个关闭的窗口，该报文将丢失，并返回一个RST，此时认为目标端口为关闭状态；若该FIN报文发送到活动窗口，报文丢失，则不会有任何反应，此时认为目标端口为开启状态。此方法速度快且隐蔽性好，但适用范围仅限于Linux系统，对Windows系统无效。因为在Windows下，无论目标端口是否开启，都会回应RST，若扫描机未收到回应，无法确定目标端口是监听状态还是回应包丢失

# 2.端口扫描的基本原理

## 2.1基本原理

端口扫描的基本原理是尝试与目标主机建立连接，如果目标主机回复扫描机则认为端口开放。本次项目设计使用Python语言，对程序进行封装，主要通过调用Socket库中的函数，与目标主机的端口进行连接。本扫描器分为针对TCP端口的扫描和针对UDP端口的扫描。

（1）针对TCP端口的扫描原理：创建socket，设置超时时间，通过socket 的connect\_ex()函数来远程连接IP地址以及对应的端口，若返回错误代码或超时，则认为目标TCP端口处于关闭状态，反之认为端口开放。此项技术的一个最大的优点是，不需要任何权限，系统中的任何用户都有权利使用这个调用。

（2）针对UDP端口的扫描原理：创建socket，设置超时时间，通过socket的sendto()函数向目标端口发送数据，使用recvfrom()函数接受返回数据，若出现错误则认为目标UDP端口处于关闭状态，反之认为端口开放

本设计引入了threading多线程机制，速度快、效率高，多线程同时使用多个套接字，开启多个连接请求，从而加速扫描。

## 2.2端口扫描的设计流程图

开始

用户命令输入

选择模式

获取指定IP或IP段

获取指定端口或端口段

与TCP端口建立连接和向UDP端口发送信息是否成功

N

Y

根据端口号获取服务名

获取扫描队列队首元素

输出目标端口状态为关闭

输出目标端口状态为开启和对应服务名

扫描队列是否为空

N

Y

结束

图2.1 端口扫描的设计流程图

用户指定参数后，程序开始运行，根据用户输入的参数选择模式，获取用户指定的IP或IP段和若干端口号或端口段，针对TCP端口，使用TCP全连接的方式尝试与目标端口建立连接，连接建立成功则认为目标端口开启，否则认为目标端口关闭；针对UDP端口，向其发送数据，若成功收到回复则认为目标端口开启，否则认为目标端口关闭。若目标端口开启，则输出目标端口状态为开启和对应服务名称，否则输出目标端口状态为关闭。再判断扫描队列是否为空，不为空则获取队首元素，继续扫描，为空表示扫描完毕，结束程序。

# 3.基于Python的端口扫描器的核心代码

## 3.1 Python语言的优势与特点

Python语言的优点主要表现在：语法简单、面向对象、解释性、可移植性等方面。

（1）**语法简单。**Python语法比C/C++简单，其去除了C/C++中一些繁琐且不必要的功能，如头文件、预处理文件、指针、结构、运算符重载、多重继承以及自动强迫同型。使用Python开发程序时，只需要专注于解决问题本身，而不用过分关注语法的细枝末节，可极大提升程序的编写速度

（2）**面向对象。**不同于Java 强制以类和对象的形式来组织代码的面向对象，Python 虽支持面向对象，但它不强制使用面向对象，自由度极高。

（3）**解释性。**Python使用了解释器，其代码逐行执行。不同于C/C++、Java等其他编程语言。Python的代码无需编译，因此更易于调试。Python的源代码被转换为字节码，是代码的实例化。

（4）**可移植性。**所有Python程序无需修改便可以在绝大多数平台上运行，具有极好的兼容性

## 3.2端口扫描的核心代码

TCP（Transmission Control Protocol传输控制协议）是一种可靠的面向连接的运输层协议。发送方和接收方各自的Socket必须建立连接才能在TCP协议的基础上进行通信，当一个服务端socket等待建立连接时，另一个客户端socket可以要求进行连接，一旦二者连接建立成功，它们就可以进行双向数据传输。

本程序针对不同情况可使用不同模式，共有四种模式可供选择，分别为00：单主机指定端口模式；01：单主机指定端口段模式；10：多主机指定端口模式；11：多主机指定端口段模式。

以下为此次设计核心代码：

def running(args,mode):#运行函数

if mode=="00" or mode=="01":#单主机扫描模式

if mode=="00":#单主机指定端口号扫描

IP=args.IP#获取指定IP

top=args.top\_list

if top!=None:#指定了top则优先使用top

p0rt\_list=Get\_p0rt\_l15t5(0,65535,top)#根据top获取端口list

else:

p0rt\_list=args.port#获取指定端口列表

t1me0ut=3#设置超时时间

for p0rt in p0rt\_list:

if Port\_judge(p0rt)!=1:#如果用户输入的端口不合法则不扫描该端口

sys.stdout.write("%d端口不合法"%p0rt)

else:

try:

TCP\_c0nnect=socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)#TCP流

TCP\_c0nnect.settimeout(t1me0ut)

re5u1t\_c0de\_TCP=TCP\_c0nnect.connect\_ex((IP,p0rt))#TCP端口开启情况

re5u1t\_c0de\_UDP=C0nnect\_UDP\_P0rt(IP,p0rt,t1me0ut)#UDP端口开启情况

if re5u1t\_c0de\_TCP==0 and re5u1t\_c0de\_UDP!=0:#TCP端口开启，UDP端口关闭

s1="% 6d [OPEN] TCP\n"%p0rt

TCP\_5erver\_name=TCP\_p0rt\_5erver\_name(p0rt)

s2=" %s\n"%TCP\_5erver\_name

sys.stdout.write(s1)

sys.stdout.write(s2)

if re5u1t\_c0de\_TCP!=0 and re5u1t\_c0de\_UDP==0:#TCP端口关闭，UDP端口开启

s1="% 6d [OPEN] UDP\n"%p0rt

UDP\_5erver\_name=UDP\_p0rt\_5erver\_name(p0rt)

s2=" %s\n"%UDP\_5erver\_name

sys.stdout.write(s1)

sys.stdout.write(s2)

if re5u1t\_c0de\_TCP==0 and re5u1t\_c0de\_UDP==0:#TCP端口和UDP端口均开启

s1="% 6d [OPEN] TCP UDP\n"%p0rt

TCP\_5erver\_name=TCP\_p0rt\_5erver\_name(p0rt)

UDP\_5erver\_name=UDP\_p0rt\_5erver\_name(p0rt)

s2=" {0} {1}\n".format(TCP\_5erver\_name,UDP\_5erver\_name)

sys.stdout.write(s1)

sys.stdout.write(s2)

if re5u1t\_c0de\_TCP!=0 and re5u1t\_c0de\_UDP!=0:#TCP端口和UDP端口均关闭

s1="% 6d [CLOSE] \n"%p0rt

sys.stdout.write(s1)

except Exception as e:

print(e)

finally:

TCP\_c0nnect.close()

elif mode=="01":#单主机指定端口段扫描

p0rt\_5cner=Multi\_p0rt\_mode\_P0rt5caner()#实例化一个Multi\_p0rt\_mode\_P0rt5caner()类

p0rt\_queue=queue.Queue()#实例化一个Queue()类

thread\_num=args.thread#线程数量

threads=[]#保存新线程

ip=args.IP#扫描的IP

\_5tart\_p0rt=args.start\_port#起始端口

end\_p0rt=args.end\_port#结束端口

top=args.top\_list#取端口top数

if top!=None:#指定了top则优先使用top

port\_list=Get\_p0rt\_l15t5(\_5tart\_p0rt,end\_p0rt,top)#根据top获取端口list

elif Port\_judge(\_5tart\_p0rt) and Port\_judge(end\_p0rt):#如果起始端口和结束端口均合法

port\_list=Get\_p0rt\_l15t5(\_5tart\_p0rt,end\_p0rt,top)#根据起始端口号和结束端口号获取端口list

else:#若起始端口和结束端口任意一个非法则终止程序

print("Program was terminated unexpectedly!")

exit()

for port in port\_list:#对port列表进行遍历，将先遍历到的先入队

p0rt\_queue.put(port)

for t in range(thread\_num):

threads.append(p0rt\_5cner.P0rt5can(p0rt\_queue,ip,timeout=3))#将实例化后的P0rt5can加入到threads列表中

for thread in threads:#启动线程

thread.start()

thread.join(0.01)#设置阻塞时间（单位:s，默认阻塞到主进程结束），此处阻塞时间短时同一端口显示在一起，阻塞时间长时同一IP显示在一起

elif mode=="10" or mode=="11":#多主机扫描模式

if mode=="10":#多主机指定端口号扫描

p0rt\_5cner=Multi\_IP\_mode\_P0rt5caner()#实例化一个Multi\_IP\_mode\_P0rt5caner()类

IP\_queue=queue.Queue()#实例化一个Queue()类

thread\_num=args.thread#线程数量

threads=[]#保存新线程

top=args.top\_list #取端口top数

if top!=None:#指定了top则优先使用top

p0rt\_list=Get\_p0rt\_l15t5(0,65535,top)#根据top获取端口list

else:

p0rt\_list=args.port#获取端口列表

start\_IP=args.start\_multi\_IP#获取起始IP

end\_IP=args.end\_multi\_IP#获取结束IP

if IP\_judge(start\_IP) and IP\_judge(end\_IP):#如果起始IP地址和结束IP地址均合法

IP\_list=Get\_IP\_l15t5(start\_IP,end\_IP)#根据起始IP地址和结束IP地址获取IP地址列表

else: #若起始IP地址和结束IP地址任意一个非法则终止程序

print("Program was terminated unexpectedly!")

exit()

for IP in IP\_list:#对IP列表进行遍历，将先遍历到的先入队

IP\_queue.put(IP)

for t in range(thread\_num):

threads.append(p0rt\_5cner.P0rt5can(IP\_queue,p0rt\_list,timeout=3))#将实例化后的P0rt5can加入到threads列表中

for thread in threads:#启动线程

thread.start()

thread.join(0.1)#测试了一下，这里0.1最适合，设置阻塞时间（单位:s，默认阻塞到主进程结束），用于确保扫描完一个IP再去扫描下一个IP。作用不大，因为影响程序运行时间的主要为目标端口。

elif mode=="11":#多主机指定端口段扫描

start\_IP=args.start\_multi\_IP#起始IP地址

end\_IP=args.end\_multi\_IP#结束IP地址

if IP\_judge(start\_IP) and IP\_judge(end\_IP):#如果起始IP地址和结束IP地址均合法，则获取IP地址列表

IP\_list=Get\_IP\_l15t5(start\_IP,end\_IP)

for IP in IP\_list:#这里采取以IP地址为准对每个端口进行线程扫描

sys.stdout.write("Scanning IP address : %s\n"%IP)

p0rt\_5cner=Multi\_p0rt\_mode\_P0rt5caner()#实例化一个Multi\_p0rt\_mode\_P0rt5caner()类

p0rt\_queue=queue.Queue()#实例化一个Queue()类

thread\_num=args.thread#线程数量

threads=[]#保存新线程

top=args.top\_list#取端口top数

\_5tart\_p0rt=args.start\_port#起始端口

end\_p0rt=args.end\_port#结束端口

if top!=None:#指定了top则优先使用top

port\_list=Get\_p0rt\_l15t5(\_5tart\_p0rt,end\_p0rt,top)#根据top获取总端口list

elif Port\_judge(\_5tart\_p0rt) and Port\_judge(end\_p0rt):#如果起始端口和结束端口均合法

port\_list=Get\_p0rt\_l15t5(\_5tart\_p0rt,end\_p0rt,top)#根据参数获取总端口list

else:#若起始端口和结束端口任意一个非法则终止程序

print("Program was terminated unexpectedly!")

exit()

for port in port\_list:#对port列表进行遍历，将先遍历到的先入队

p0rt\_queue.put(port)

for t in range(thread\_num):

threads.append(p0rt\_5cner.P0rt5can(p0rt\_queue,IP,timeout=3))

for thread in threads:#启动线程

thread.start()

thread.join()#这里如果不使用默认阻塞会导致子程序都没结束主程序就全部开始，输出会非常混乱。设置阻塞时间（单位:s，默认阻塞到主进程结束），用于确保扫描完一个IP再去扫描下一个IP。作用不大，因为影响程序运行时间的主要为目标端口回应速度

# 4.端口扫描的设计优化

## 4.1多线程扫描

本次设计使用了Python的threading库，可在一个程序中同时打开多处运行单元，各线程同时执行，从而大大提高了扫描速度。在本程序中，用户可使用默认线程数，也可以指定线程数，提升了程序的可操作性和可修改性，一定程度上节省了扫描时间，提升了扫描效率。

本次设计的端口扫描器通过Python封装Multi\_p0rt\_mode\_P0rt5caner类，其子类P0rt5can继承threading模块的Thread类，初始化port队列，加入线程，并对run方法进行重写来判断某端口是否成功连接。对于TCP，使用socket库的connect\_ex()函数判断连接情况，连接成功视为目标端口开启，返回错误代码则视目标端口处于关闭状态；对于UDP，通过自定义函数C0nnect\_UDP\_P0rt()，使用socket库的sendto()函数向目标端口发送数据，并使用recvfrom()函数接收返回数据，若成功接收到返回数据而不发生错误，则认为目标端口开启，否则认为目标端口处于关闭状态。Multi\_IP\_mode\_P0rt5caner类同理。

以下为多线程扫描的核心代码：

class Multi\_p0rt\_mode\_P0rt5caner(object):#多端口模式扫描类

class P0rt5can(threading.Thread):#继承了threading模块的Thread类，对run方法进行了重写

def \_\_init\_\_(self, port\_queue, ip, timeout=3):#初始化参数

threading.Thread.\_\_init\_\_(self)

self.\_\_port\_queue=port\_queue

self.\_\_ip=ip

self.\_\_timeout=timeout

def run(self):#一旦此类被调用，此方法中的代码自动运行

while True:

if self.\_\_port\_queue.empty():#如果port队列不空则继续运行

break

p0rt=self.\_\_port\_queue.get(timeout=0.5)#从port队列中取出队首的port

IP=self.\_\_ip

t1me0ut=self.\_\_timeout

try:

TCP\_c0nnect=socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)#TCP流

TCP\_c0nnect.settimeout(t1me0ut)#设置超时时间

re5u1t\_c0de\_TCP=TCP\_c0nnect.connect\_ex((IP,p0rt))#TCP端口开启情况

re5u1t\_c0de\_UDP=C0nnect\_UDP\_P0rt(IP,p0rt,t1me0ut)#UDP端口开启情况

if re5u1t\_c0de\_TCP==0 and re5u1t\_c0de\_UDP!=0:#TCP端口开启，UDP端口关闭

s1="% 6d [OPEN] TCP\n"%p0rt

TCP\_5erver\_name=TCP\_p0rt\_5erver\_name(p0rt)

s2=" %s\n"%TCP\_5erver\_name

sys.stdout.write(s1)

sys.stdout.write(s2)

if re5u1t\_c0de\_TCP!=0 and re5u1t\_c0de\_UDP==0:#TCP端口关闭，UDP端口开启

s1="% 6d [OPEN] UDP\n"%p0rt

UDP\_5erver\_name=UDP\_p0rt\_5erver\_name(p0rt)

s2=" %s\n"%UDP\_5erver\_name

sys.stdout.write(s1)

sys.stdout.write(s2)

if re5u1t\_c0de\_TCP==0 and re5u1t\_c0de\_UDP==0:#TCP端口和UDP端口均开启

s1="% 6d [OPEN] TCP UDP\n"%p0rt

TCP\_5erver\_name=TCP\_p0rt\_5erver\_name(p0rt)

UDP\_5erver\_name=UDP\_p0rt\_5erver\_name(p0rt)

s2=" {0} {1}\n".format(TCP\_5erver\_name,UDP\_5erver\_name)

sys.stdout.write(s1)

sys.stdout.write(s2)

if re5u1t\_c0de\_TCP!=0 and re5u1t\_c0de\_UDP!=0:#TCP端口和UDP端口均关闭

s1="% 6d [CLOSE] \n"%p0rt

sys.stdout.write(s1)

except Exception as e:

print(e)

break

finally:

TCP\_c0nnect.close()

class Multi\_IP\_mode\_P0rt5caner(object):#多主机模式扫描类

class P0rt5can(threading.Thread):#继承了threading模块的Thread类，对run方法进行了重写

def \_\_init\_\_(self, ip\_queue, port\_list, timeout=3):#初始化参数

threading.Thread.\_\_init\_\_(self)

self.\_\_ip\_queue=ip\_queue

self.\_\_port=port\_list

self.\_\_timeout=timeout

def run(self):#一旦此类被调用，此方法中的代码自动运行

while True:

if self.\_\_ip\_queue.empty():#如果IP队列不空则继续运行

break

IP=self.\_\_ip\_queue.get(timeout=0.5)#从IP队列中取出队首的IP

p0rt\_list=self.\_\_port

t1me0ut=self.\_\_timeout

for p0rt in p0rt\_list:

try:

TCP\_c0nnect=socket.socket(socket.AF\_INET,socket.SOCK\_STREAM)#TCP流

TCP\_c0nnect.settimeout(t1me0ut)#设置超时时间

re5u1t\_c0de\_TCP=TCP\_c0nnect.connect\_ex((IP,p0rt))#TCP端口开启情况

re5u1t\_c0de\_UDP=C0nnect\_UDP\_P0rt(IP,p0rt,t1me0ut)##UDP端口开启情况

if re5u1t\_c0de\_TCP==0 and re5u1t\_c0de\_UDP!=0:#TCP端口开启，UDP端口关闭

s1=" {0} [OPEN] {1} TCP\n".format(IP,p0rt)

TCP\_5erver\_name=TCP\_p0rt\_5erver\_name(p0rt)

s2=" %s\n"%TCP\_5erver\_name

sys.stdout.write(s1)

sys.stdout.write(s2)

if re5u1t\_c0de\_TCP!=0 and re5u1t\_c0de\_UDP==0:#TCP端口关闭，UDP端口开启

s1=" {0} [OPEN] {1} UDP\n".format(IP,p0rt)

UDP\_5erver\_name=UDP\_p0rt\_5erver\_name(p0rt)

s2=" %s\n"%UDP\_5erver\_name

sys.stdout.write(s1)

sys.stdout.write(s2)

if re5u1t\_c0de\_TCP==0 and re5u1t\_c0de\_UDP==0:#TCP端口和UDP端口均开启

s1=" {0} [OPEN] {1} TCP UDP\n".format(IP,p0rt)

TCP\_5erver\_name=TCP\_p0rt\_5erver\_name(p0rt)

UDP\_5erver\_name=UDP\_p0rt\_5erver\_name(p0rt)

s2=" {0} {1}\n".format(TCP\_5erver\_name,UDP\_5erver\_name)

sys.stdout.write(s1)

sys.stdout.write(s2)

if re5u1t\_c0de\_TCP!=0 and re5u1t\_c0de\_UDP!=0:#TCP端口和UDP端口均关闭

s1=" {0} [CLOSE] {1} \n".format(IP,p0rt)

sys.stdout.write(s1)

except Exception as e:

print(e)

break

finally:

TCP\_c0nnect.close()

## 4.2端口服务名称获取

IP是目标访问的地址，而端口是目标访问的“门户”，通过端口号可以区分不同的服务，例如80对应HTTP协议，443端口对应HTTPS协议。一般情况下，一台主机上同一个端口可以对应多个服务，前提是它们的协议不同，即存在TCP协议的253端口，也存在UDP协议下的253端口。对于TCP和UDP的前1024个端口，可以通过端口号，使用socket库的getservbyport()函数获取对应的服务名称。

以下为端口服务名称获取的核心代码：

def TCP\_p0rt\_5erver\_name(port):#获取TCP端口服务名称函数

try:

result=socket.getservbyport(port,"tcp")#获取TCP下指定端口对应的服务名

return result

except Exception as e:

return "None"

def UDP\_p0rt\_5erver\_name(port):#获取UDP端口服务名称函数

try:

result=socket.getservbyport(port,"udp")#获取UDP下指定端口对应的服务名

return result

except Exception as e:

return "None"

## 4.3常用端口列表获取

通过对常用端口的扫描，可以帮助使用者快速获取目标存活端口，搜寻可能存在的漏洞裂口。

以下为常用端口列表获取的核心代码：

def Get\_p0rt\_l15t5(\_5tart\_p0rt,end\_p0rt,top=None):#获取常用端口列表函数

T0p1000\_l15t=[1,3,6,9,13,17,19,20,21,22,23,24,25,30,32,37,42,49,53,70,79,80,81,82,83,84,88,89,99,106,109,110,113,119,125,135,139,143,146,161,163,179,199,211,222,254,255,259,264,280,301,306,311,340,366,389,406,416,425,427,443,444,458,464,481,497,500,512,513,514,524,541,543,544,548,554,563,587,593,616,625,631,636,646,648,666,667,683,687,691,700,705,711,714,720,722,726,749,765,777,783,787,800,808,843,873,880,888,898,900,901,902,911,981,987,990,992,995,999,1000,1001,1007,1009,1010,1021,1022,1023,1024,1025,1026,1027,1028,1029,1030,1031,1032,1033,1034,1035,1036,1037,1038,1039,1040,1041,1042,1043,1044,1045,1046,1047,1048,1049,1050,1051,1052,1053,1054,1055,1056,1057,1058,1059,1060,1061,1062,1063,1064,1065,1066,1067,1068,1069,1070,1071,1072,1073,1074,1075,1076,1077,1078,1079,1080,1081,1082,1083,1084,1085,1086,1087,1088,1089,1090,1091,1092,1093,1094,1095,1096,1097,1098,1099,1102,1104,1105,1106,1107,1110,1111,1112,1113,1117,1119,1121,1122,1123,1126,1130,1131,1137,1141,1145,1147,1148,1151,1154,1163,1164,1165,1169,1174,1183,1185,1186,1192,1198,1201,1213,1216,1217,1233,1236,1244,1247,1259,1271,1277,1287,1296,1300,1309,1310,1322,1328,1334,1352,1417,1433,1443,1455,1461,1494,1500,1503,1521,1524,1533,1556,1580,1583,1594,1600,1641,1658,1666,1687,1700,1717,1718,1719,1720,1723,1755,1761,1782,1801,1805,1812,1839,1862,1863,1875,1900,1914,1935,1947,1971,1974,1984,1998,1999,2000,2001,2002,2003,2004,2005,2006,2007,2008,2009,2013,2020,2021,2030,2033,2034,2038,2040,2041,2042,2045,2046,2047,2048,2065,2068,2099,2103,2105,2106,2111,2119,2121,2126,2135,2144,2160,2170,2179,2190,2196,2200,2222,2251,2260,2288,2301,2323,2366,2381,2382,2393,2399,2401,2492,2500,2522,2525,2557,2601,2604,2607,2638,2701,2710,2717,2725,2800,2809,2811,2869,2875,2909,2920,2967,2998,3000,3003,3005,3006,3011,3013,3017,3030,3052,3071,3077,3128,3168,3211,3221,3260,3268,3283,3300,3306,3322,3323,3324,3333,3351,3367,3369,3370,3371,3389,3404,3476,3493,3517,3527,3546,3551,3580,3659,3689,3703,3737,3766,3784,3800,3809,3814,3826,3827,3851,3869,3871,3878,3880,3889,3905,3914,3918,3920,3945,3971,3986,3995,3998,4000,4001,4002,4003,4004,4005,4045,4111,4125,4129,4224,4242,4279,4321,4343,4443,4444,4445,4449,4550,4567,4662,4848,4899,4998,5000,5001,5002,5003,5009,5030,5033,5050,5054,5060,5080,5087,5100,5101,5120,5190,5200,5214,5221,5225,5269,5280,5298,5357,5405,5414,5431,5440,5500,5510,5544,5550,5555,5560,5566,5631,5633,5666,5678,5718,5730,5800,5801,5810,5815,5822,5825,5850,5859,5862,5877,5900,5901,5902,5903,5906,5910,5915,5922,5925,5950,5952,5959,5960,5961,5962,5987,5988,5998,5999,6000,6001,6002,6003,6004,6005,6006,6009,6025,6059,6100,6106,6112,6123,6129,6156,6346,6389,6502,6510,6543,6547,6565,6566,6580,6646,6666,6667,6668,6689,6692,6699,6779,6788,6792,6839,6881,6901,6969,7000,7001,7004,7007,7019,7025,7070,7100,7103,7106,7200,7402,7435,7443,7496,7512,7625,7627,7676,7741,7777,7800,7911,7920,7937,7999,8000,8001,8007,8008,8009,8010,8021,8031,8042,8045,8080,8081,8082,8083,8084,8085,8086,8087,8088,8089,8093,8099,8180,8192,8193,8200,8222,8254,8290,8291,8300,8333,8383,8400,8402,8443,8500,8600,8649,8651,8654,8701,8800,8873,8888,8899,8994,9000,9001,9002,9009,9010,9040,9050,9071,9080,9090,9099,9100,9101,9102,9110,9200,9207,9220,9290,9415,9418,9485,9500,9502,9535,9575,9593,9594,9618,9666,9876,9877,9898,9900,9917,9929,9943,9968,9998,9999,10000,10001,10002,10003,10009,10012,10024,10082,10180,10215,10243,10566,10616,10621,10626,10628,10778,11110,11967,12000,12174,12265,12345,13456,13722,13782,14000,14238,14441,15000,15002,15003,15660,15742,16000,16012,16016,16018,16080,16113,16992,17877,17988,18040,18101,18988,19101,19283,19315,19350,19780,19801,19842,20000,20005,20031,20221,20828,21571,22939,23502,24444,24800,25734,26214,27000,27352,27355,27715,28201,30000,30718,30951,31038,31337,32768,32769,32770,32771,32772,32773,32774,32775,32776,32777,32778,32779,32780,32781,32782,32783,32784,33354,33899,34571,34572,35500,38292,40193,40911,41511,42510,44176,44442,44501,45100,48080,49152,49153,49154,49155,49156,49157,49158,49159,49160,49163,49165,49167,49175,49400,49999,50000,50001,50002,50006,50300,50389,50500,50636,50800,51103,51493,52673,52822,52848,52869,54045,54328,55055,55555,55600,56737,57294,57797,58080,60020,60443,61532,61900,62078,63331,64623,64680,65000,65129,65389]

T0p100\_l15t=[7,9,13,21,22,25,37,53,79,80,88,106,110,113,119,135,139,143,179,199,389,427,443,444,465,513,514,543,548,554,587,631,646,873,990,993,995,1025,1026,1027,1028,1110,1433,1720,1723,1755,1900,2000,2049,2121,2717,3000,3128,3306,3389,3986,4899,5000,5009,5051,5060,5101,5190,5357,5432,5631,5666,5800,5900,6000,6646,7070,8000,8008,8080,8443,8888,9100,9999,32768,49152,49153,49154,49155,49156]

T0p50\_l15t=[21,22,25,53,80,110,113,135,139,143,179,199,443,445,465,514,548,554,587,646,993,995,1025,1026,1433,1720,1723,2000,3306,3389,5060,5666,5900,6001,8000,8008,8080,8443,8888,10000,32768,49152,49154]

if (top!=None):#指定了top列表则按top列表返回

if (top==50):

return T0p50\_l15t

elif (top==100):

return T0p100\_l15t

else:

return T0p1000\_l15t

else:#指定了扫描端口范围则start port-end port

return list(range(\_5tart\_p0rt,end\_p0rt+1))

**4.4端口合法性判断和IP地址合法性判断**

端口范围为0—65535,其中0端口为保留端口，不投入使用。

构造一个正则表达式。判断用户输入的IP与正则表达式是否匹配。若匹配则是正确的IP地址，返回True，否则是错误的IP地址，返回False

下面给出相对应的验证IP的正则表达式：

^(1\d{2}|2[0-4]\d|25[0-5]|[1-9]\d|[1-9])\.(1\d{2}|2[0-4]\d|25[0-5]|[1-9]\d|\d)\.(1\d{2}|2[0-4]\d|25[0-5]|[1-9]\d|\d)\.(1\d{2}|2[0-4]\d|25[0-5]|[1-9]\d|\d)$

^表示字符串开头

\d表示匹配0~9中任何一个数字

{2}表示正好匹配两次

[0-4]表示匹配0~4中任何一个数字

| 表示或者

1\d{2}表示100~199之间的任意一个数字

2[0-4]\d表示200~249之间的任意一个数字

25[0-5]表示250~255之间的任意一个数字

[1-9]\d表示10~99之间的任意一个数字

[1-9])表示1~9之间的任意一个数字

$表示字符串末尾

以下为端口合法性判断和IP地址合法性判断的代码：

def Port\_judge(port):#端口合法性判断函数

if port in range(0,65536):

return True

else:

print("Port is invalid!")

return False

def IP\_judge(ip):#IP地址合法性判断函数

pattern=re.compile('^(1\d{2}|2[0-4]\d|25[0-5]|[1-9]\d|[1-9])\.(1\d{2}|2[0-4]\d|25[0-5]|[1-9]\d|\d)\.(1\d{2}|2[0-4]\d|25[0-5]|[1-9]\d|\d)\.(1\d{2}|2[0-4]\d|25[0-5]|[1-9]\d|\d)$')#re正则匹配IP地址

if pattern.match(ip):

return True

else:

return False

**4.5用户命令获取**

本次设计使用Python的argparse库进行命令行参数操作，即方便设计，又使得用户自定义修改和升级变得尤为简单，只需要使用ArgumentParser类的add\_argument方法添加新的参数，即可在parse\_args()中使用这个新参数

以下为用户命令获取的代码：

def \_0rDer5\_Get():#获取用户命令函数

parser=argparse.ArgumentParser(description="Usage of program")

parser.add\_argument("-m","--mode",metavar="MODE",type=str,default="00",choices=["00","01","10","11"],required=True,help="Running mode(00:Single Host-Single Port;01:Single Host-Multi Port;10:Multi Host-Single Port;11:Multi Host-Multi Port)")

parser.add\_argument("-ip","--IP",metavar="IPADRESS",type=str,default="127.0.0.1",help="Single Host scaning mode destination IP address")

parser.add\_argument("-smip","--start\_multi\_IP",metavar="START\_MULTI\_IP\_ADDRESS",type=str,default="192.168.0.0",help="Multi Host scaning mode,Start destination IP address")

parser.add\_argument("-emip","--end\_multi\_IP",metavar="END\_MULTI\_IP\_ADDRESS",type=str,default="192.168.255.255",help="Multi Host scaning mode,End destination IP address")

parser.add\_argument("-p","--port",metavar="PORT",type=int,default=80,nargs='\*',help="Single Port scaning mode,Destination host port")

parser.add\_argument("-sp","--start\_port",metavar="START\_PORT",type=int,default=1,help="Multi Port scaning mode,Start destination port (1~65534)")

parser.add\_argument("-ep","--end\_port",metavar="END\_PORT",type=int,default=65535,help="Multi Port scaning mode,End destination port (2~65535)")

parser.add\_argument("-t","--thread",metavar="THREAD",type=int,default=100,help="The number of threads")

parser.add\_argument("-top","--top\_list",metavar="TOP",type=int,default=None,choices=[50,100,1000],help="Top port lists")

args=parser.parse\_args()

return args

# 5.端口扫描的实现效果

本设计采用argparse命令行输入形式，有如下参数：

-h：帮助信息，可选参数，告知用户本程序的使用方式。

-m：模式，必选参数，在[00,01,10,11]中取值。00：单主机指定端口模式；01：单主机指定端口段模式；10：多主机指定端口模式；11：多主机指定端口段模式。

-ip：目标IP地址，可选参数，默认本地127.0.0.1。适用于[00,01]模式

-smip：起始IP地址，可选参数，默认192.168.0.0。适用于[10,11]模式

-emip：结束IP地址，可选参数，默认192.168.255.255。适用于[10,11]模式

-p：目标端口，可选参数，默认80。适用于[00,01]模式

-sp：起始目标端口，可选参数，默认1，适用于[10,11]模式

-ep：结束目标端口，可选参数，默认65535，适用于[10,11]模式

-t：线程数，可选参数，默认100。

-top：常用端口排名，可选参数，在[50,100,1000]中取值，默认None。

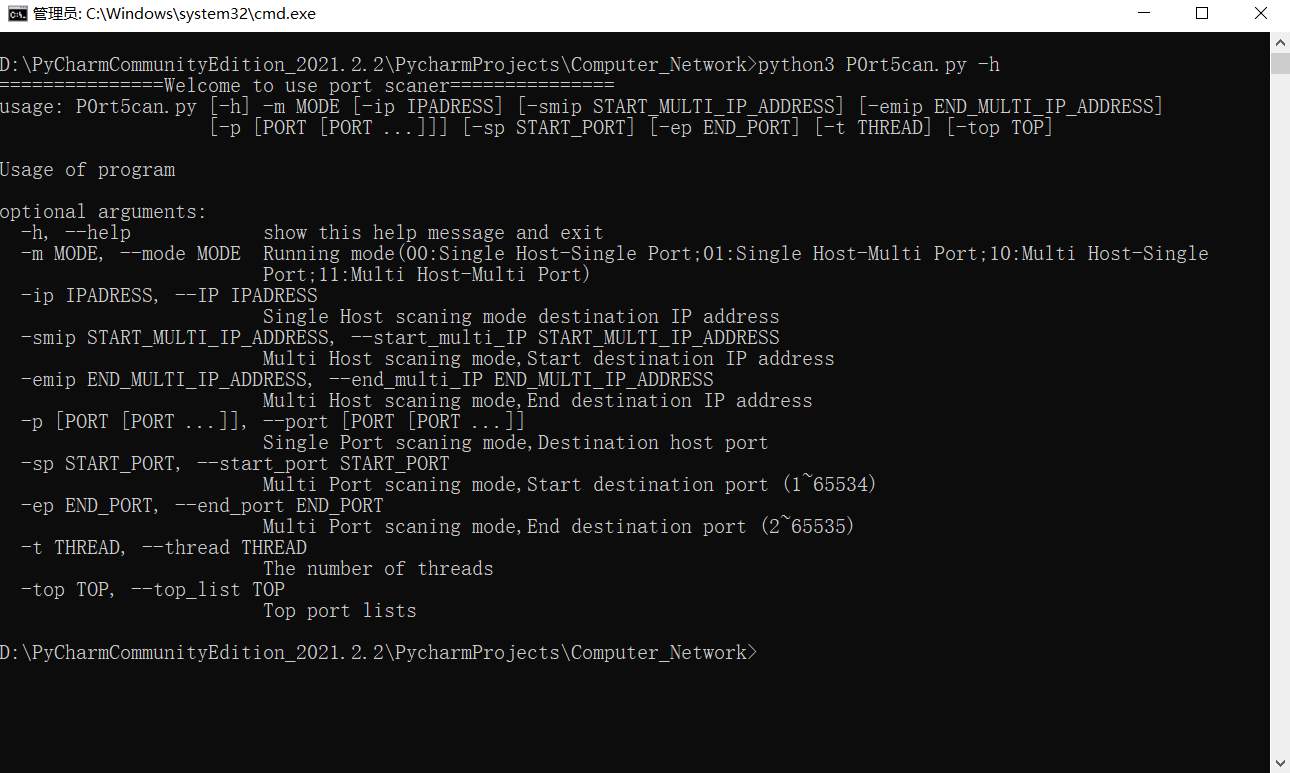


图5.1 argparse命令行帮助信息

实现了如下四个功能：

（1）扫描目标IP指定若干端口

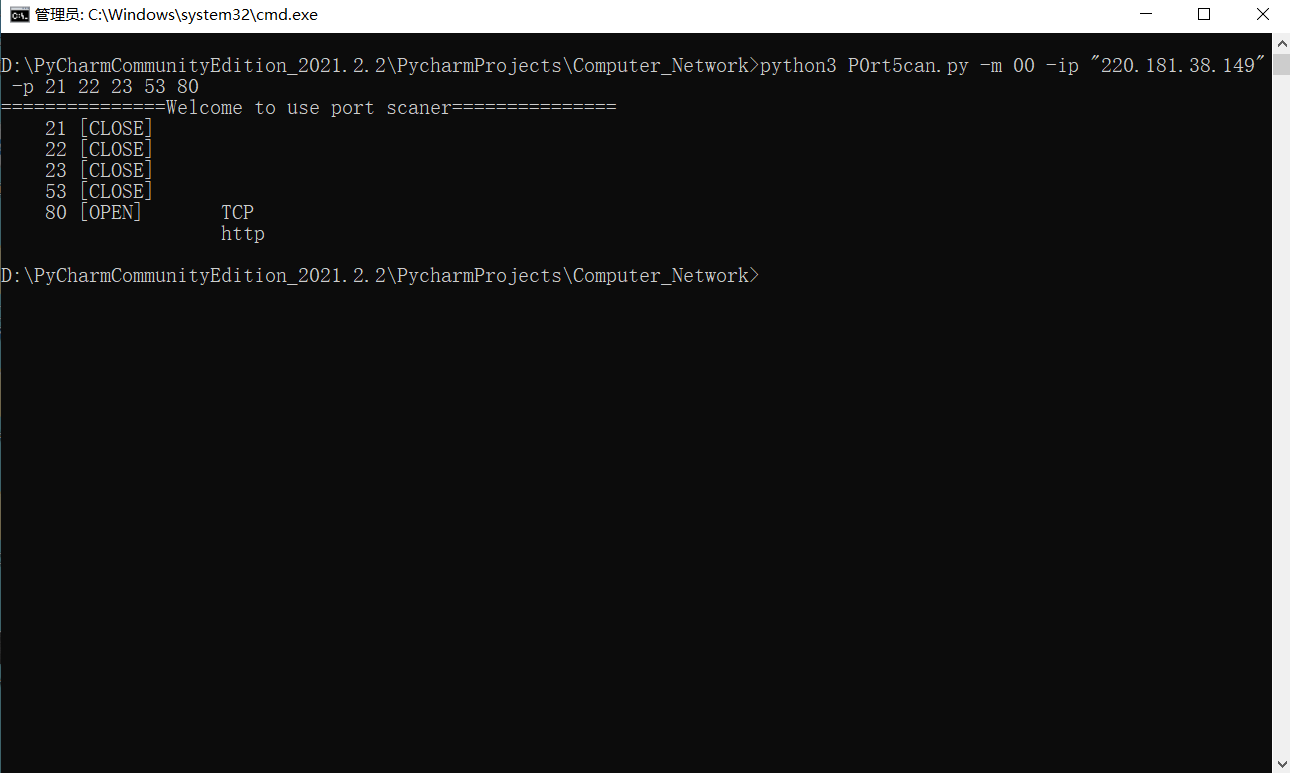


图5.2 单主机指定若干端口扫描

如上图所示，使用00：单主机指定端口模式，对“220.181.38.149”的21、22、23、53、80端口进行扫描，结果为21、22、23、53端口均关闭，80端口开放

（2）扫描目标IP指定端口段

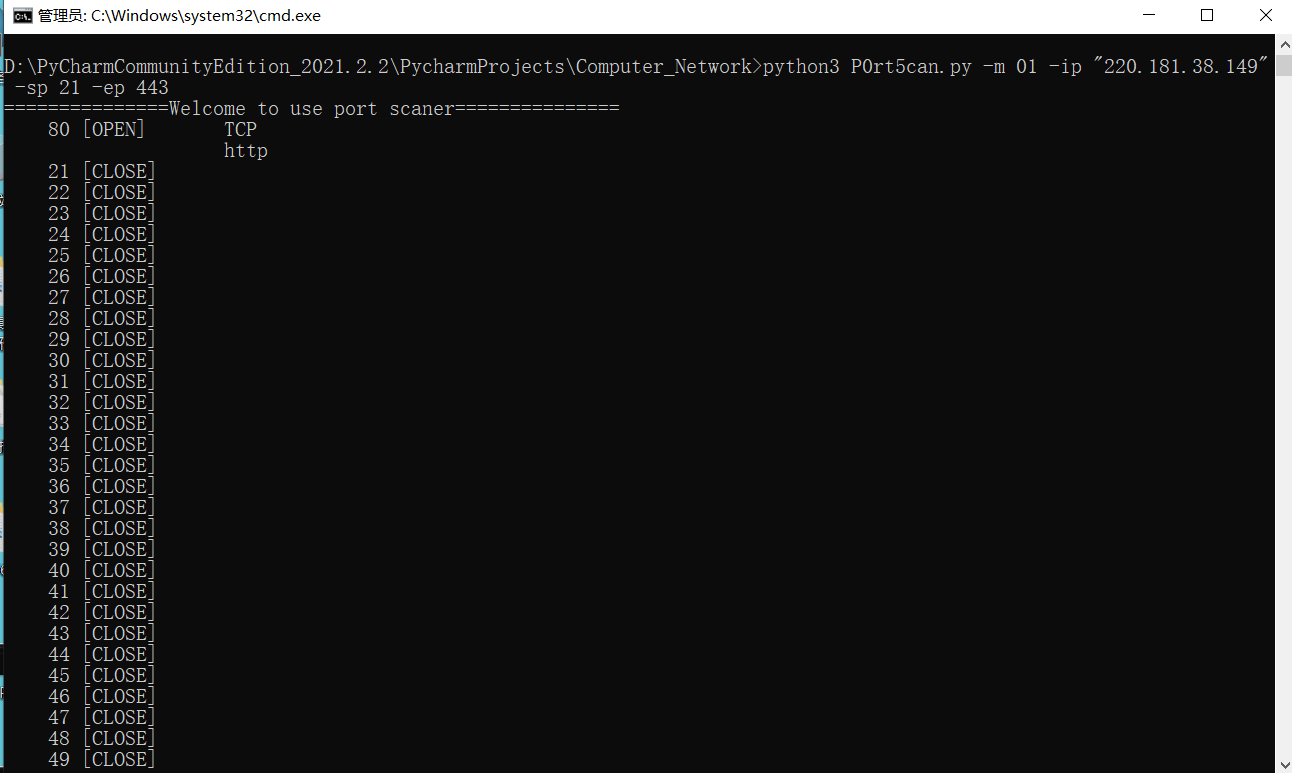


图5.3 单主机指定端口段扫描

如上图所示，使用01：单主机指定端口段模式，对“220.181.38.149”的21-443之间的端口进行扫描，结果为绝大部分端口关闭，80端口开启

（3）扫描IP段指定若干端口

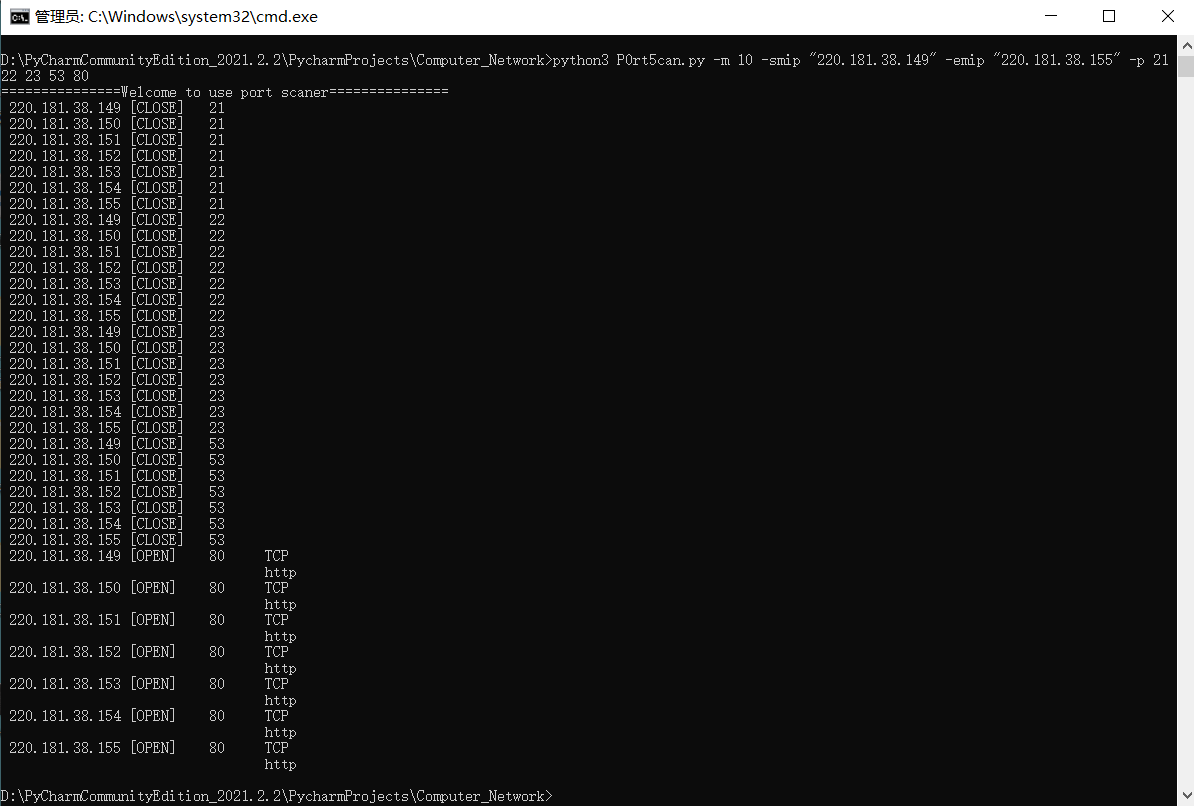


图5.4 主机段指定若干端口扫描

如上图所示，使用10：多主机指定端口模式，对“220.181.38.149”-“220.181.38.155”的21、22、23、53、80端口进行扫描，发现“220.181.38.149”-“220.181.38.155”的21、22、23、53端口关闭，其80端口开放

（4）扫描IP段指定端口段

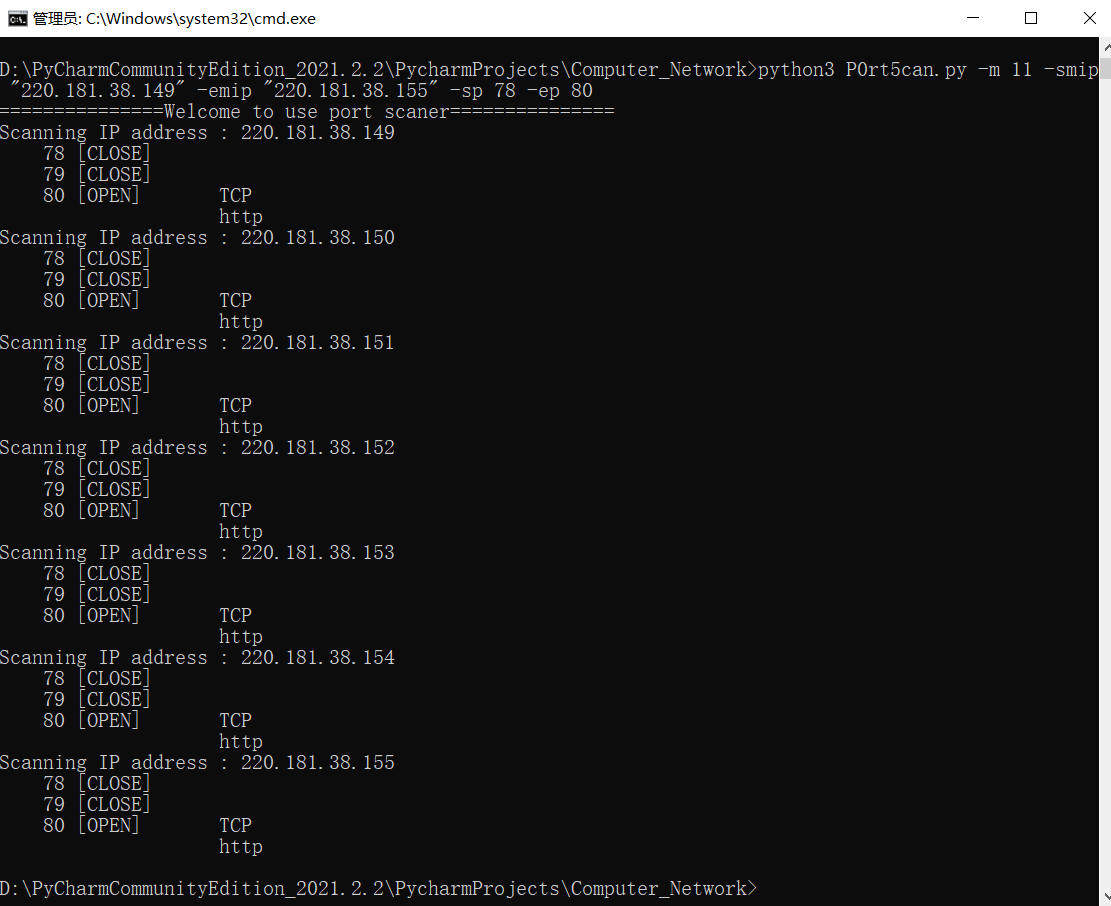


图5.5 主机段指定端口段扫描

如上图所示，使用11：多主机指定端口段模式，对“220.181.38.149”-“220.181.38.155”的78-80端口进行扫描，发现“220.181.38.149”-“220.181.38.155”的78、79端口关闭，80端口开放

本设计的TCP\_p0rt\_5erver\_name()函数和UDP\_p0rt\_5erver\_name()函数会分别根据端口号，通过socket库的getservbyport()函数获取端口对应的服务名称，例如下表的常用端口对应的服务名称

表5-1 常用端口对应服务名称

|  |  |  |
| --- | --- | --- |
| 协议 | 服务 | 端口号 |
| HTTP | 超文本传输服务 | 80 |
| SMTP | 简单邮件传输服务 | 25 |
| FTP | 文件传输服务 | 21 |
| Telnet | 远程登录服务 | 23 |
| MySQL | MySQL数据库服务 | 3306 |

遗憾的是socket库的getservbyport()函数搜索情况并不完善，无法找出所有拥有服务的端口的服务名称。

# 项目设计心得体会

在此次项目设计中，我对计算机网络的运输层端口的相关知识有了更深一步的理解。通过建立TCP连接来扫描远程主机的TCP端口是否开放和向目标UDP端口发送数据探测目标UDP端口是否开放让我更加了解了TCP/IP协议的原理，同时我更加熟练掌握了关于Python基本的网络编程技术和方法，建立网络编程整体概念。这次实验也锻炼了我使用Python语言解决实际问题的能力，掌握了socket库、threading库和argparse库函数的各种方法。

在设计的过程中遇到了许多问题，第一个问题是多线程中使用print()函数造成输出混乱，出现多个输出结果混杂的情况，Python使用多线程时是并发而非并行，线程之间并非严格遵守顺序，这就会造成线程不安全的情况，例如print()函数不具备原子性，函数自动添加换行而end的默认参数”\n”与print()函数的内容并非同时操作，而是先输出value，再输出”\n”，显然这极易导致输出发生混乱。解决方法是使用sys库的stdout.write类进行输出同时设置阻塞时间或给程序上锁，用于确保主进程的子进程全部运行完后再开启下一个主进程。但这并不是万无一失的，对于改变程序输出速度作用不大，因为影响程序运行时间的主要为目标端口回应速度，当某些子进程同时进行时，因为扫描对象各异，部分端口反应迅速，返回数据快，会先一步获取并输出结果，所以所有结果并非完全按顺序输出。

另一个问题是UDP的无状态无连接和不可靠性导致对目标UDP端口探测结果不可靠。不同于TCP可靠的建立连接，UDP只负责尽可能地传输数据，无任何确认机制。这意味着对UDP端口的探测也就不可能像TCP端口的探测那样依赖于连接建立过程，这也使得UDP端口扫描的可靠性不高。所以虽然UDP协议较之TCP协议显得更为简单便捷，但是对UDP端口的扫描却是相当困难的。仅简单地使用connect\_ex()函数返回值判断目标UDP端口是否开放不可靠，因为返回的内容不管是什么，connect\_ex()函数都会认为探测成功，函数返回0。在UDP中调用connect()函数，实际上内核只是将第二个参数addr指代的目的host和port记录下来，形成了网络四元组，但是网络上不会有任何变化。所以UDP无法利用connect()检测端口是否开放。扫描机向目标端口发送UDP报文，可能发生如下情况：

（1）数据包中途丢失或被防火墙拦截，未到达目标端口，扫描机未收到回复

（2）数据包到达目标端口，目标端口开启，因为无连接不作反应，扫描机未收到回复

（3）数据包到达目标端口，目标端口未开启，目标主机被禁止回复，扫描机未收到回复

（4）数据包到达目标端口，目标端口未开启，目标主机回复ICMP port-unreachable报文，扫描机收到回复

综上所述，无论目标UDP端口是否向扫描机发送回复报文，都无法判断目标端口是否开放，此问题无法解决。由此可见，对UDP的端口扫描意义不大，收效甚微。故本次设计默认向目标UDP端口发送信息，若正确接收到回复信息则认为目标UDP端口开放。

最后，感谢我的计算机网络启蒙老师温晓飞和渗透测试启蒙导师小迪，以及每一个传授我知识的老师、同学和网友，多亏你们我才能够深入的理解本次设计，按质按量按时的完成本次设计的端口扫描器，感谢你们！

# 

# 参考文献

[1]谢希仁.《计算机网络第七版》[M].电子工业出版社:北京市,2017.

[2]欧阳逸轩.Python网络编程——通过指定的端口和协议找到服务名.https://www.cnblogs.com/ouyangyixuan/p/5871946.html,2016-09-14.

[3]huanghelouzi.python信息安全工具之端口扫描器.https://blog.csdn.net/huanghelouzi/article/details/83833737,2018-11-08.

[4]小夕nike.udp检测对端端口是否开放.https://www.cnblogs.com/moonwalk/p/15878733.html,2022-02-10.