

**CENTRAL SOUTH UNIVERSITY**

**计算机网络实验报告**

|  |  |
| --- | --- |
| 学生姓名 | 张子洋 |
| 专业班级 | 计科2203班 |
| 学 号 | 8208221223 |
| 学 院 | 计算机学院 |
| 完成日期 | 2024.6.1 |

目录

[一、实验目的与要求 3](#_Toc7088)

[1.1 实验A.3：简单Web服务器端实现 3](#_Toc13717)

[1.1.1 实验目的 3](#_Toc7640)

[1.1.2 实验要求 3](#_Toc18277)

[1.2 实验B.3：套接字编程实验 4](#_Toc22316)

[1.2.1 实验目的 4](#_Toc10303)

[1.2.2 实验要求 4](#_Toc705)

[二、 实验原理与实验内容 5](#_Toc23867)

[2.1 Socket编程接口 5](#_Toc22078)

[2.2 HTTP传输协议 7](#_Toc16011)

[2.3 TCP与UDP 7](#_Toc1428)

[2.3.1 TCP/IP网络模型 7](#_Toc733)

[2.3.2 TCP 8](#_Toc23878)

[2.3.3 UDP 11](#_Toc15085)

[三、实验具体设计实现及结果 13](#_Toc28742)

[3.1 简单Web服务器端实现 13](#_Toc26240)

[3.1.1 处理客户端连接（客户端套接字） 13](#_Toc32116)

[3.1.2 启动服务器 13](#_Toc22111)

[3.1.3 流程图 14](#_Toc4663)

[3.2 套接字编程实验 16](#_Toc27408)

[3.2.1 tcp\_scan()函数 16](#_Toc23250)

[3.2.2 udp\_scan()函数 16](#_Toc32343)

[3.2.3 scan\_ports()函数 16](#_Toc22107)

[3.2.4 流程图 17](#_Toc30152)

[3.3 实验结果 18](#_Toc4997)

[3.3.1 A3实验结果 18](#_Toc12022)

[3.3.2 B3实验结果 19](#_Toc1177)

[四、实验设备与实验环境 20](#_Toc14847)

[五、实验总结 21](#_Toc11344)

[5.1 改进建议 21](#_Toc15746)

[5.1.1 A3 21](#_Toc12912)

[5.1.2 B3 21](#_Toc30533)

[5.2 未来展望 21](#_Toc9817)

# 一、实验目的与要求

参考实验指导书，本次实验选择了 A3及B3两个题目，下面给出两个题目各自的目的及要求。

## 实验A.3：简单Web服务器端实现

### **实验目的**

学习面向TCP连接的套接字编程基础知识：如何创建套接字，将其绑定到特定的地址和端口，以及发送和接收数据包。学习 HTTP 协议格式的相关知识，在此基础上，开发一个简单的 Web 服务器，它仅能处理一个HTTP连接请求。

### **实验要求**

Web 服务器的基本功能是接受并解析客户端的 HTTP 请求，然后从服务器的文件系统获取所请求的文件，生成一个由头部和响应文件内容所构成成的 HTTP 响应消息，并将该响应消息发送给客户端。如果请求的文件不存在于服务器中，则服务器应该向客户端发送“404 Not Found”差错报文。 具体的过程和步骤分为：

1. 当一个客户（浏览器）连接时，创建一个连接套接字；
2. 从这个连接套接字接收 HTTP 请求；
3. 解释该请求以确定所请求的特定文件；
4. 从服务器的文件系统获得请求的文件；
5. 创建一个由请求的文件组成的 HTTP 响应报文，报文前面有首部行；
6. 经 TCP 连接向请求浏览器发送响应；
7. 如果浏览器请求一个在该服务器中不存在的文件，服务器应当返回一个“404 Not Found”差错报文。

## 实验B.3：套接字编程实验

### 1.2.1 实验目的

网络编程是通过使用[套接字](http://baike.baidu.com/view/538713.htm" \t "_blank)来达到[进程间通信](http://baike.baidu.com/view/1492468.htm" \t "_blank)目的的编程，Socket编程是网络编程的主流工具，Socket API是实现进程间通信的一种编程设施，也是一种为进程间提供底层抽象的机制，提供了访问下层通信协议的大量系统调用和相应的数据结构。本实验利用Socket API编写网络通信程序，具体实验要求及内容如下：

* 掌握C++、JAVA或Python等集成开发环境编写网络程序的方法；
* 掌握客户/服务器（C/S）应用的工作方式；
* 学习网络中进程之间通信的原理和实现方法；
* 要求本机既是客户端又是服务器端；

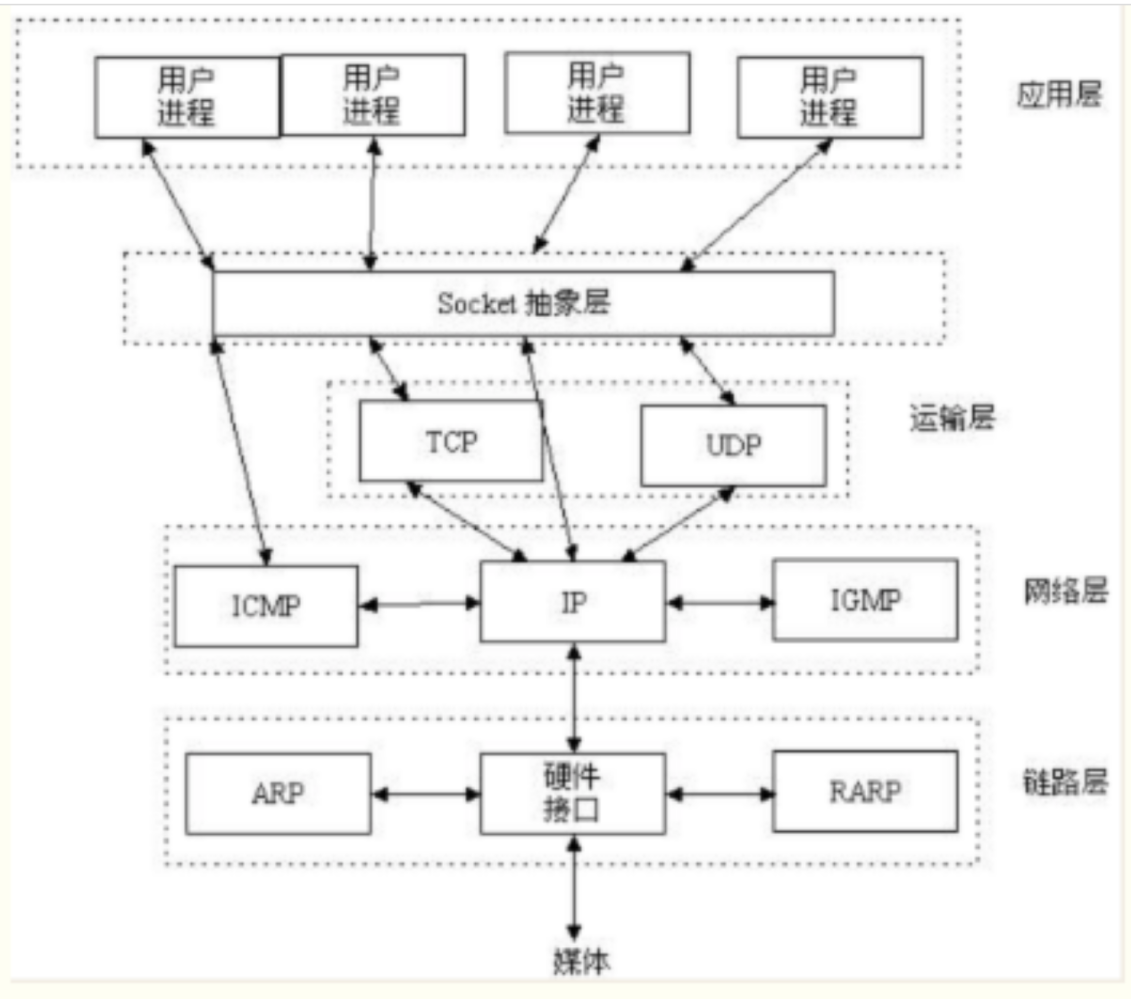
### 1.2.2 实验要求

编写一个端口扫描器（类似nmap的简易版本）。给定目的ip地址，可以扫描目的IP地址在哪个端口上可以接受tcp连接和udp连接。

# 实验原理与实验内容

## 2.1 Socket编程接口

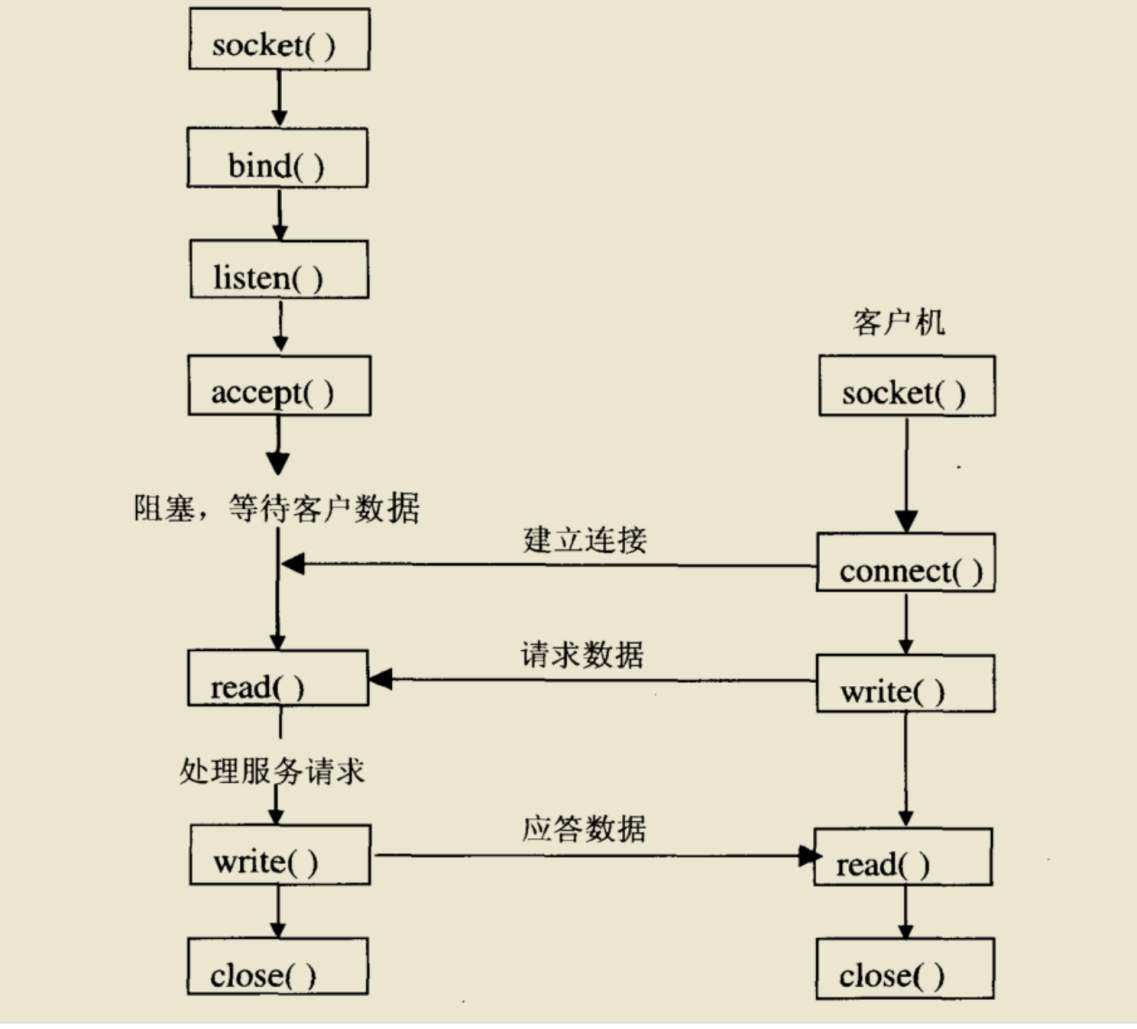
要实现 Web 服务器，需使用套接字 Socket编程接口来使用操作系统提供的网络通信功能。 Socket 是应用层与 TCP/IP 协议族通信的中间软件抽象层，是一组编程接口。它把复杂的 TCP/IP 协议族隐藏在 Socket 接口后面，对用户来说，一组简单的接口就是全部，让 Socket 去组织数据，以符合指定的协议。使用 Socket 后，无需深入理解 TCP/UDP 协议细节（因为Socket 已经为我们封装好了），只需要遵循 Socket 的规定去编程，写出的程序自然就是遵循 TCP/UDP 标准的。Socket 的地位如下图所示：



从某种意义上说，Socket 由地址IP和端口Port构成。IP 是用来标识互联网中的一台主机的位置，而 Port 是用来标识这台机器上的一个应用程序，IP 地址是配置到网卡上的，而 Port 是应用程序开启的，IP 与 Port 的绑定就标识了互联网中独一无二的一个应用程序。

套接字类型 流式套接字（SOCK\_STREAM）：用于提供面向连接、可靠的数据传输服务。 数据报套接字（SOCK\_DGRAM）：提供了一种无连接的服务。该服务并不能保证数据传输的可靠性，数据有可能在传输过程中丢失或出现数据重复，且无法保证顺序地接收到数据。 原始套接字（SOCK\_RAW）：主要用于实现自定义协议或底层网络协议。

在本 WEB 服务器程序实验中，采用流式套接字进行通信。其基本模型如下图所示：



其工作过程如下：服务器首先启动，通过调用 socket() 建立一个套接字，然后调用绑定方法 bind() 将该套接字和本地网络地址联系在一起，再调用 listen() 使套接字做好侦听连接的准备，并设定的连接队列的长度。客户端在建立套接字后，就可调用连接方法 connect() 向服务器端提出连接请求。服务器端在监听到连接请求后，建立和该客户端的连接，并放入连接队列中，并通过调用 accept() 来返回该连接，以便后面通信使用。客户端和服务器连接一旦建立，就可以通过调用接收方法 recv()／recvfrom() 和发送方法send()／sendto() 来发送和接收数据。最后，待数据传送结束后，双方调用 close() 关闭套接字。

## **2.2 HTTP传输协议**

超文本传输协议（HTTP）是用于Web上进行通信的协议：它定义Web浏览器如何从Web服务器请求资源以及服务器如何响应。为简单起见，在该实验中将处理HTTP协议的1.0版。HTTP通信以事务形式进行，其中事务由客户端向服务器发送请求，然后读取响应组成。 请求和响应消息共享一个通用的基本格式：

* 初始行（请求或响应行）
* 零个或多个头部行
* 空行（CRLF）
* 可选消息正文。

对于大多数常见的HTTP事务，协议归结为一系列相对简单的步骤：

首先，客户端创建到服务器的连接；然后客户端通过向服务器发送一行文本来发出请求。这请求行包HTTP方法(比如GET，POST、PUT等)，请求URI(类似于URL)，以及客户机希望使用的协议版本(比如HTTP/1.0)；接着，服务器发送响应消息，其初始行由状态线（指示请求是否成功)，响应状态码(指示请求是否成功完成的数值)，以及推理短语(一种提供状态代码描述的英文消息组成)；最后一旦服务器将响应返回给客户端，它就会关闭连接。

## 2.3 TCP与UDP

TCP/IP 中有两个具有代表性的传输层协议，分别是 TCP 和 UDP。

### 2.3.1 TCP/IP网络模型

计算机与网络设备要相互通信，双方就必须基于相同的方法。比如，如何探测到通信目标、由哪一边先发起通信、使用哪种语言进行通信、怎样结束通信等规则都需要事先确定。不同的硬件、操作系统之间的通信，所有的这一切都需要一种规则。而我们就把这种规则称为协议（protocol）。

TCP/IP 是互联网相关的各类协议族的总称，比如：TCP，UDP，IP，FTP，HTTP，ICMP，SMTP 等都属于 TCP/IP 族内的协议。TCP/IP模型是互联网的基础，它是一系列网络协议的总称。这些协议可以划分为四层，分别为应用层、传输层、网络层和链路层。

1. 应用层：负责向用户提供应用程序，比如HTTP、FTP、Telnet、DNS、SMTP等。
2. 传输层：负责对报文进行分组和重组，并以TCP或UDP协议格式封装报文。
3. 网络层：负责路由以及把分组报文发送给目标网络或主机。
4. 链路层：负责封装和解封装IP报文，发送和接受ARP/RARP报文等。



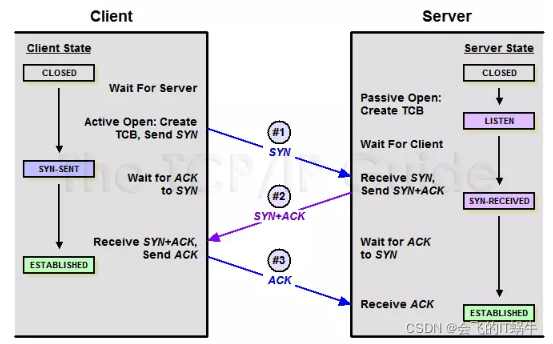
接下去介绍TCP/IP 中有两个具有代表性的传输层协议----TCP 和 UDP。

### 2.3.2 TCP

当一台计算机想要与另一台计算机通讯时，两台计算机之间的通信需要畅通且可靠，这样才能保证正确收发数据。例如，当你想查看网页或查看电子邮件时，希望完整且按顺序查看网页，而不丢失任何内容。当你下载文件时，希望获得的是完整的文件，而不仅仅是文件的一部分，因为如果数据丢失或乱序，都不是你希望得到的结果，于是就用到了TCP。

TCP协议全称是传输控制协议是一种面向连接的、可靠的、基于字节流的传输层通信协议。

#### ⑴ TCP连接过程



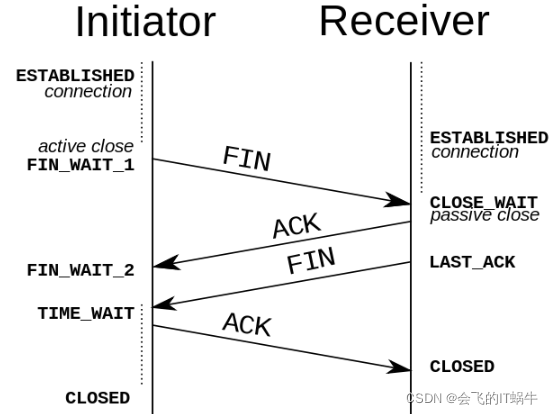
第一次握手：客户端向服务端发送连接请求报文段。该报文段中包含自身的数据通讯初始序号。请求发送后，客户端便进入 SYN-SENT 状态。

第二次握手：服务端收到连接请求报文段后，如果同意连接，则会发送一个应答，该应答中也会包含自身的数据通讯初始序号，发送完成后便进入 SYN-RECEIVED 状态。

第三次握手：当客户端收到连接同意的应答后，还要向服务端发送一个确认报文。客户端发完这个报文段后便进入 ESTABLISHED 状态，服务端收到这个应答后也进入 ESTABLISHED 状态，此时连接建立成功。

#### ⑵ TCP断开链接

TCP 是全双工的，在断开连接时两端都需要发送 FIN 和 ACK。



1. 第一次握手

若客户端 A 认为数据发送完成，则它需要向服务端 B 发送连接释放请求。

1. 第二次握手

B 收到连接释放请求后，会告诉应用层要释放 TCP 链接。然后会发送 ACK 包，并进入 CLOSE\_WAIT 状态，此时表明 A 到 B 的连接已经释放，不再接收 A 发的数据了。但是因为 TCP 连接是双向的，所以 B 仍旧可以发送数据给 A。

1. 第三次握手

B 如果此时还有没发完的数据会继续发送，完毕后会向 A 发送连接释放请求，然后 B 便进入 LAST-ACK 状态。

1. 第四次握手

A 收到释放请求后，向 B 发送确认应答，此时 A 进入 TIME-WAIT 状态。该状态会持续 2MSL（最大段生存期，指报文段在网络中生存的时间，超时会被抛弃） 时间，若该时间段内没有 B 的重发请求的话，就进入 CLOSED 状态。当 B 收到确认应答后，也便进入 CLOSED 状态。

#### ⑶ TCP协议的特点

1. 面向连接

面向连接，是指发送数据之前必须在两端建立连接。建立连接的方法是“三次握手”，这样能建立可靠的连接。建立连接，是为数据的可靠传输打下了基础。

1. 仅支持单播传输

每条TCP传输连接只能有两个端点，只能进行点对点的数据传输，不支持多播和广播传输方式。

1. 面向字节流

TCP不像UDP一样那样一个个报文独立地传输，而是在不保留报文边界的情况下以字节流方式进行传输。

1. 可靠传输

对于可靠传输，判断丢包，误码靠的是TCP的段编号以及确认号。TCP为了保证报文传输的可靠，就给每个包一个序号，同时序号也保证了传送到接收端实体的包的按序接收。然后接收端实体对已成功收到的字节发回一个相应的确认(ACK)；如果发送端实体在合理的往返时延(RTT)内未收到确认，那么对应的数据（假设丢失了）将会被重传。

1. 提供拥塞控制

当网络出现拥塞的时候，TCP能够减小向网络注入数据的速率和数量，缓解拥塞

### 2.3.3 UDP

UDP协议全称是用户数据报协议，在网络中它与TCP协议一样用于处理数据包，是一种无连接的协议。在OSI模型中，在第四层——传输层，处于IP协议的上一层。UDP有不提供数据包分组、组装和不能对数据包进行排序的缺点，也就是说，当报文发送之后，是无法得知其是否安全完整到达的。

它有以下几个特点：

1. 面向无连接

首先 UDP 是不需要和 TCP一样在发送数据前进行三次握手建立连接的，想发数据就可以开始发送了。并且也只是数据报文的搬运工，不会对数据报文进行任何拆分和拼接操作。

1. 有单播，多播，广播的功能

UDP 不止支持一对一的传输方式，同样支持一对多，多对多，多对一的方式，也就是说 UDP 提供了单播，多播，广播的功能。

1. UDP是面向报文的

发送方的UDP对应用程序交下来的报文，在添加首部后就向下交付IP层。UDP对应用层交下来的报文，既不合并，也不拆分，而是保留这些报文的边界。因此，应用程序必须选择合适大小的报文。

1. 不可靠性

首先不可靠性体现在无连接上，通信都不需要建立连接，想发就发，这样的情况肯定不可靠。并且收到什么数据就传递什么数据，并且也不会备份数据，发送数据也不会关心对方是否已经正确接收到数据了。

再者网络环境时好时坏，但是 UDP 因为没有拥塞控制，一直会以恒定的速度发送数据。即使网络条件不好，也不会对发送速率进行调整。这样实现的弊端就是在网络条件不好的情况下可能会导致丢包，但是优点也很明显，在某些实时性要求高的场景（比如电话会议）就需要使用 UDP 而不是 TCP。

# **三、实验具体设计实现及结果**

## 3.1 简单Web服务器端实现

用 Python 实现了一个基本的多线程 HTTP 服务器。该服务器监听指定的主机和端口，接受传入连接，并从本地文件系统提供请求的文件。如果文件不存在，返回404错误。

### 3.1.1 处理客户端连接（客户端套接字）

1. 接收请求: 通过 recv 方法从客户端接收请求数据，最多接收1024字节，并解码为UTF-8字符串。

2. 解析请求: 将请求按行拆分，取第一行（请求行），再将其按空格拆分，取第二个部分，即请求的资源路径。

3. 确定文件路径: 如果请求的是根目录（/），则默认返回 index.html。将请求路径的第一个字符（/）去掉，得到文件路径。

4. 检查文件是否存在:

* 文件存在: 打开文件并读取内容，构造HTTP响应，包括状态行、响应头（内容类型、内容长度、连接关闭）、空行和响应体（文件内容）。
* 文件不存在: 构造404响应，包括状态行、响应头（内容类型、内容长度、连接关闭）、空行和响应体（“404 Not Found”）。

5.发送响应: 通过 sendall 方法将响应发送给客户端。

6.关闭连接: 关闭客户端连接，确保资源释放。

### 3.1.2 启动服务器

1. 创建服务器套接字: 使用 socket 方法创建一个IPv4（AF\_INET）和TCP（SOCK\_STREAM）套接字。

2. 设置套接字选项: 允许地址重用，以便服务器重新启动时不必等待。

3. 绑定地址和端口: 使用 bind 方法绑定服务器套接字到指定的主机和端口。

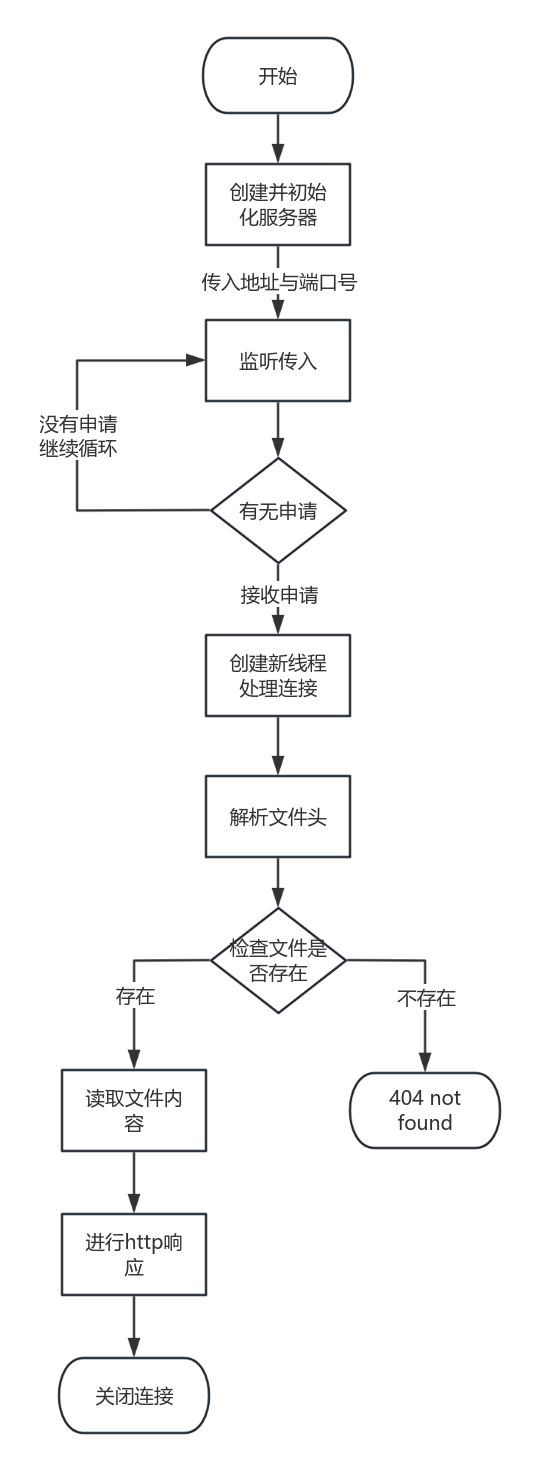
4. 开始监听: 使用 listen 方法开始监听传入的连接，请求队列的最大长度为5。

5. 处理连接:

* 接受连接: 使用 accept 方法接受传入的连接，这会阻塞，直到有客户端连接。
* 创建新线程: 为每个客户端连接创建一个新的线程，调用 handle\_client\_connection 函数处理连接。

1. 服务器持续运行: 在一个无限循环中，持续接受和处理新的连接。

### 3.1.3 流程图



## 3.2 套接字编程实验

实现了一个简单的端口扫描器，可以扫描指定IP地址范围内的TCP和UDP端口，并返回开放的端口列表。

### 3.2.1 tcp\_scan()函数

尝试连接指定的IP和端口以确定TCP端口是否开放。

1.创建TCP套接字并设置超时时间。

2.尝试连接指定IP和端口。

3. 如果连接成功，端口是开放的，将端口添加到结果列表。

4. 无论连接是否成功，都关闭套接字以释放资源。

### 3.2.2 udp\_scan()函数

尝试发送UDP数据包到指定的IP和端口，以确定UDP端口是否开放。

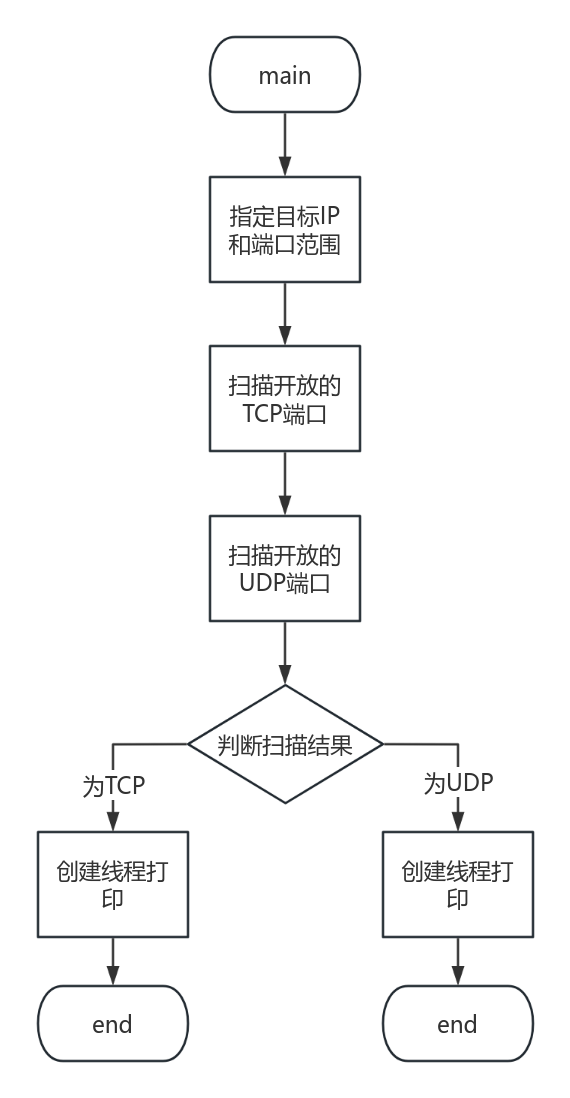
1. 创建UDP套接字并设置超时时间。
2. 发送一个空的UDP数据包到指定IP和端口。
3. 尝试接收响应数据，如果接收到响应数据，端口可能是开放的，将端口添加到结果列表。
4. 无论是否接收到响应数据，都关闭套接字以释放资源。

### 3.2.3 scan\_ports()函数

扫描指定范围的端口，根据扫描类型（TCP或UDP）确定开放的端口。

1. 初始化线程列表和结果列表。
2. 遍历指定的端口范围，为每个端口创建一个扫描线程（根据指定的扫描类型）。
3. 启动所有扫描线程，并将其添加到线程列表。
4. 等待所有线程完成扫描。
5. 返回结果列表，其中包含所有开放的端口。

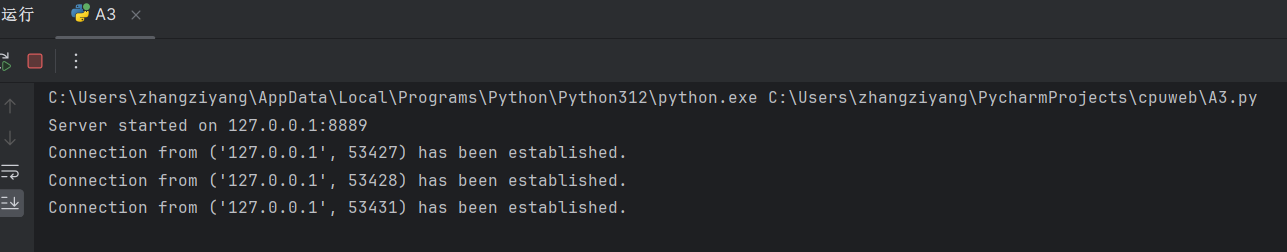
### 3.2.4 流程图



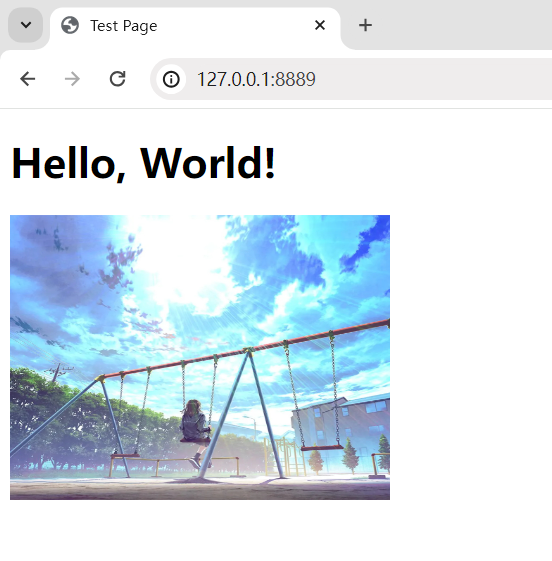
## 3.3 实验结果

### 3.3.1 A3实验结果

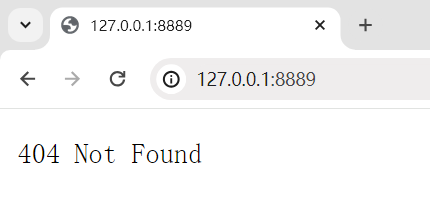
1. 运行服务器进行监听



1. 响应成功

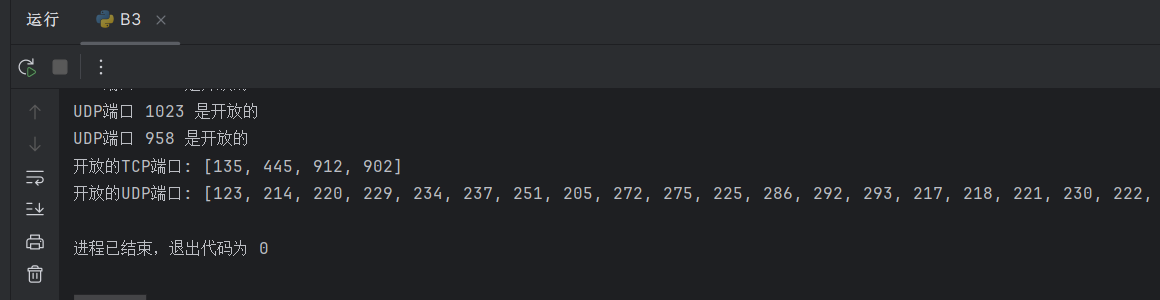


1. 找不到文件，输出404 not found



### 3.3.2 B3实验结果

直接进行扫描，每行输出开放端口，最终存在两个列表中。



# **四、实验设备与实验环境**

本次两个实验均使用python进行编写。

具体运行环境：python 3.12解释器

编译器：PyCharm Community Edition 2023.3.4

# **五、实验总结**

## 5.1 改进建议

### 5.1.1 A3

1. 支持其他HTTP方法: 增加对POST、PUT等方法的支持。

2. 日志记录功能: 记录客户端请求和服务器响应的详细信息。

3. HTTPS支持: 提升安全性，支持加密通信。

4. 错误处理: 实现更复杂的错误处理机制。

### 5.1.2 B3

1. 扫描结果记录: 记录端口状态和扫描时间等详细信息。

2. 检测防火墙和安全设置: 提高扫描结果的准确性。

3. 带宽和资源控制: 防止扫描过程对网络和设备造成过大负载。

## 5.2 未来展望

通过这两个实验，我深刻理解了网络编程的基本原理和实际应用。HTTP服务器的实现让我掌握了如何处理客户端请求、返回响应以及多线程编程的技巧。而端口扫描器则让我了解了TCP和UDP协议的不同特性及其在网络安全中的应用。这些实践不仅增强了我的编程技能，还让我意识到优化和改进代码的重要性，为未来开发更复杂的网络应用打下了坚实基础。

源代码：

A3:

import socket  
import threading  
import os  
  
def handle\_client\_connection(client\_socket):  
 """处理客户端连接的函数"""  
 try:  
 # 接收客户端请求数据  
 request = client\_socket.recv(1024).decode('utf-8')  
 #从客户端接收最多1024字节的数据，并将其解码为UTF-8字符串。  
 headers = request.split('\n')  
 filename = headers[0].split()[1]  
 #将请求按行拆分成多个字符串，取第一行（请求行），再将其按空格拆分，取第二个部分，即请求的资源路径。  
  
 # 如果请求的文件是根目录，默认返回index.html  
 if filename == '/':  
 filename = '/index.html'  
  
 # 去掉URL中的'/'  
 filepath = filename[1:]  
  
 # 检查请求的文件是否存在  
 if os.path.exists(filepath):  
 # 如果文件存在，读取文件内容并构造HTTP响应  
 with open(filepath, 'rb') as f:  
 content = f.read()  
 response = (  
 'HTTP/1.1 200 OK\r\n' # 1.首部行  
 'Content-Type: text/html; charset=UTF-8\r\n' # 2.响应头:内容类型  
 'Content-Length: {}\r\n' # 内容长度  
 'Connection: close\r\n' # 连接关闭  
 '\r\n' # 3.空行，分隔首部和内容  
 ).format(len(content)).encode() + content#4.响应体  
 else:  
 # 如果文件不存在，返回404 Not Found错误  
 response = (  
 'HTTP/1.1 404 Not Found\r\n' # 首部行  
 'Content-Type: text/plain; charset=UTF-8\r\n' # 内容类型  
 'Content-Length: 13\r\n' # 内容长度  
 'Connection: close\r\n' # 连接关闭  
 '\r\n' # 空行，分隔首部和内容  
 '404 Not Found' # 错误消息  
 ).encode()  
  
 # 发送响应给客户端  
 client\_socket.sendall(response)  
 except Exception as e:  
 # 如果出现异常，打印错误信息  
 print(f"An error occurred: {e}")  
 finally:  
 # 关闭客户端连接  
 client\_socket.close()  
  
def start\_server(host='127.0.0.1', port=8889):  
 """启动服务器，监听响应的函数"""  
 # 创建服务器套接字  
 # 可以看作是服务器和客户端之间的通信通道  
 server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 # 设置套接字选项，允许地址重用  
 server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)  
 # 绑定地址和端口  
 server\_socket.bind((host, port))  
 # 开始监听传入的连接，最大连接数为5  
 server\_socket.listen(5)  
 print(f"Server started on {host}:{port}")  
  
 while True:  
 # 接受客户端连接  
 client\_socket, addr = server\_socket.accept()  
 print(f"Connection from {addr} has been established.")  
 # 创建新线程处理客户端连接  
 client\_handler = threading.Thread(target=handle\_client\_connection, args=(client\_socket,))  
 client\_handler.start()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 # 启动服务器  
 start\_server()  
#127.0.0.1:888

B3：

import socket  
import threading  
  
# TCP扫描可以通过尝试连接目标IP的特定端口来实现。如果连接成功，则该端口是开放的；如果连接失败，则该端口是关闭的。  
# UDP扫描比TCP扫描更复杂，因为UDP是无连接的。我们可以发送一个空的UDP数据包到目标端口，并等待可能的ICMP端口不可达消息。  
# 如果没有收到这样的消息，我们可以认为该端口可能是开放的。  
  
# TCP扫描函数，尝试连接指定IP和端口  
def tcp\_scan(ip, port, results):  
 try:  
 # 创建TCP套接字  
 sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)   
 sock.settimeout(1)  
 sock.connect((ip, port))  
 except (socket.timeout, ConnectionRefusedError):  
 # 连接超时或被拒绝  
 return  
 except socket.error:  
 # 其他套接字错误  
 return  
 else:  
 # 连接成功，添加到结果列表  
 results.append(port)  
 finally:  
 sock.close()  
  
# UDP扫描函数，尝试发送UDP数据包到指定IP和端口  
def udp\_scan(ip, port, results):  
 try:  
 # 创建UDP套接字  
 sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  
 sock.settimeout(1)  
 sock.sendto(b'', (ip, port)) # 向目标IP的指定端口发送一个空的UDP数据包。b''表示一个空的字节串  
 data, \_ = sock.recvfrom(1024) # 只将接收到的数据赋值给data变量  
 except socket.timeout:  
 # 没有响应可能表示端口开放或被过滤  
 results.append(port)  
 except socket.error:  
 # 其他套接字错误  
 return  
 finally:  
 sock.close()  
  
# 扫描指定范围的端口  
def scan\_ports(ip, start\_port, end\_port, scan\_type="tcp"):  
 threads = []  
 results = []  
  
 for port in range(start\_port, end\_port + 1):  
 if scan\_type == "tcp":  
 # 创建TCP扫描线程  
 thread = threading.Thread(target=tcp\_scan, args=(ip, port, results))  
 elif scan\_type == "udp":  
 # 创建UDP扫描线程  
 thread = threading.Thread(target=udp\_scan, args=(ip, port, results))  
 thread.start()  
 threads.append(thread)  
  
 for thread in threads:  
 thread.join()# 等待所有创建的线程完成执行  
  
 return results  
  
def main():  
 target\_ip = "127.0.0.1" # 修改为你要扫描的IP地址  
 start\_port = 1  
 end\_port = 1024  
  
 print(f"扫描 {target\_ip} 的开放TCP端口...")  
 open\_tcp\_ports = scan\_ports(target\_ip, start\_port, end\_port, "tcp")  
 for port in open\_tcp\_ports:  
 print(f"TCP端口 {port} 是开放的")  
  
 print(f"扫描 {target\_ip} 的开放UDP端口...")  
 open\_udp\_ports = scan\_ports(target\_ip, start\_port, end\_port, "udp")  
 for port in open\_udp\_ports:  
 print(f"UDP端口 {port} 是开放的")  
  
 print(f"开放的TCP端口: {open\_tcp\_ports}")  
 print(f"开放的UDP端口: {open\_udp\_ports}")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()