

**CENTRAL SOUTH UNIVERSITY**

**计算机网络实验报告**

|  |  |
| --- | --- |
| 学生姓名 | 吕环翥 |
| 专业班级 | T2201 |
| 学 号 | 8211220709 |
| 学 院 | 计算机学院 |
| 完成日期 | 2024.6.4 |

**1.实验A.3简单Web服务器端程序实现**

**实验要求：**本实验要求学生实现一个简单的Web服务器端程序，该程序监听TCP 80端口，能够接受传入的HTTP连接请求并进行解析，并且能够正确的响应请求，回送相关的网页。为简单起见，仅要求正确解析常用的get请求，并只需要支持一个HTTP连接。

**实验目的与要求**

实验目的：

本实验的目的是通过开发一个简单的Web服务器，学习和掌握面向TCP连接的套接字编程基础知识以及HTTP协议的基本格式和工作原理。具体包括以下几点：

理解套接字（Socket）的概念及其在网络通信中的作用。

学习如何使用套接字接口创建服务器和客户端应用程序。

掌握HTTP协议的基本结构和操作方法。

能够实现一个简单的Web服务器，处理HTTP请求并返回响应。

实验要求：

创建一个服务器端程序，能够监听指定端口上的TCP连接请求。

接受并解析客户端发送的HTTP请求。

从服务器的文件系统中获取请求的文件内容，并生成HTTP响应消息。

将HTTP响应消息发送给客户端。

如果请求的文件不存在，向客户端返回“404 Not Found”错误消息。

实现的服务器应能处理至少一个HTTP连接请求。

**实验原理与实验内容**

实验原理：

Socket编程：

套接字（Socket）是网络通信的基本编程接口，它为应用程序提供了一种抽象层，用于发送和接收数据。

在TCP连接中，服务器端套接字需要经历创建、绑定、监听和接受连接等步骤。客户端套接字则需要经历创建和连接等步骤。

HTTP协议：

HTTP（超文本传输协议）是Web上进行通信的协议，定义了客户端与服务器之间的请求和响应格式。

一个典型的HTTP请求包括请求行、头部行、空行和可选的消息主体。响应包括状态行、头部行、空行和消息主体。

实验内容：

创建服务器套接字：

使用socket()函数创建一个流式套接字（SOCK\_STREAM）。

使用bind()函数将套接字绑定到本地地址和端口。

使用listen()函数使套接字进入监听状态，准备接受连接请求。

接受连接：

使用accept()函数接受客户端的连接请求，返回一个新的套接字用于通信。

接收HTTP请求：

从连接套接字中读取客户端发送的HTTP请求消息。

解析请求消息，提取请求的文件名。

处理请求：

根据请求的文件名，从服务器的文件系统中读取文件内容。

如果文件存在，生成包含文件内容的HTTP响应消息。

如果文件不存在，生成“404 Not Found”错误消息。

发送响应：

将生成的HTTP响应消息发送给客户端。

关闭连接套接字，准备处理下一个连接请求。

**实验具体设计实现及结果**

关键代码说明

创建服务器套接字并绑定到本地地址和端口：

serverSocket = socket(AF\_INET, SOCK\_STREAM)

serverSocket.bind(("127.0.0.1", 8080)) # 绑定到本地地址和端口8080

serverSocket.listen(1) # 监听连接请求

接受客户端连接请求：

connectionSocket, addr = serverSocket.accept()

接收客户端HTTP请求：

message = connectionSocket.recv(1024).decode() # 获取客户发送的报文

解析HTTP请求并获取文件名

parts = message.split()

filename = parts[1] # 获取请求的文件名

if filename == "/":

filename = "/index.html" # 默认返回index.html

读取请求的文件内容：

with open(filename[1:], 'r') as f:

outputdata = f.read()

生成并发送HTTP响应：

header='HTTP/1.1200OK\nConnection:close\nContent-Type: text/html\nContent-Length: %d\n\n' % (len(outputdata))

connectionSocket.send(header.encode())

connectionSocket.send(outputdata.encode())

处理文件未找到情况：

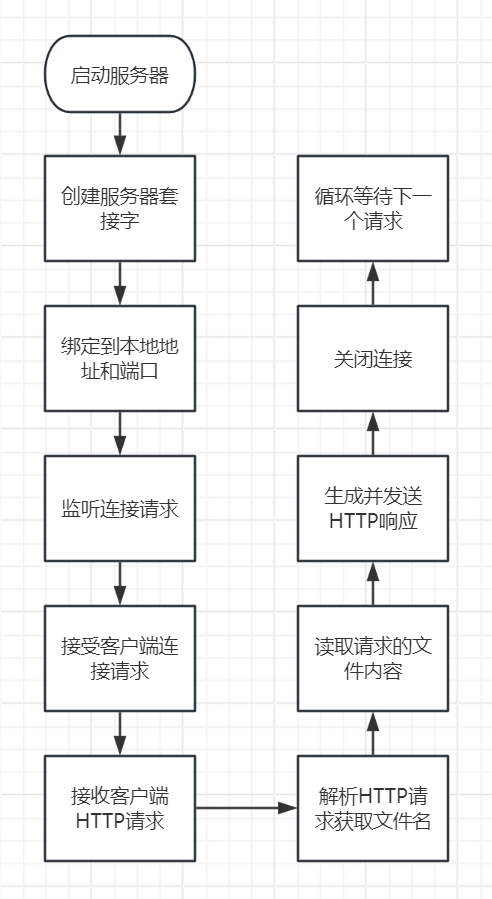
except IOError:

header = 'HTTP/1.1 404 Not Found\n\n'

connectionSocket.send(header.encode())

关闭连接：

connectionSocket.close()



程序源代码

from socket import \*

import os

print("Current directory contents:")

print(os.listdir("."))  # 列出当前目录的文件

#创建服务器套接字并绑定到本地地址和端口

serverSocket = socket(AF\_INET, SOCK\_STREAM)

# Prepare a server socket

serverSocket.bind(("127.0.0.1", 8080))  # 这里将端口改为8080

serverSocket.listen(1)

try:

    while True:

        print('开始WEB服务...')

        #接受客户端连接请求：

        connectionSocket, addr = serverSocket.accept()

        try:

            #接收客户端HTTP请求

            message = connectionSocket.recv(1024).decode()  # 获取客户发送的报文

            print(f"Received message: {message}")  # 调试信息

            if not message:

                connectionSocket.close()

                continue

            # 解析HTTP请求并读取文件内容

            parts = message.split()

            #检查分割后的列表长度是否小于2。如果小于2,说明收到的请求格式不正确,缺少必要的部分。

            if len(parts) < 2:#至少要有请求方法和路径

                header = 'HTTP/1.1 400 Bad Request\n\n'

                connectionSocket.send(header.encode())

                connectionSocket.close()

                continue

            filename = parts[1]  # message=["GET", "/index.html", "HTTP/1.1", ...]

            print(f"Requested filename: {filename}")  # 添加调试信息

            if filename == "/":

                filename = "/index.html"  # 默认返回index.html

            try:

                with open(filename[1:], 'r') as f:

                    outputdata = f.read()

                print(f"File content: {outputdata}")  # 打印文件内容

                # 向套接字发送头部信息

                header = 'HTTP/1.1 200 OK\nConnection: close\nContent-Type: text/html\nContent-Length: %d\n\n' % (

                    len(outputdata))

                connectionSocket.send(header.encode())

                print("Sent header")  # 添加调试信息

                # 发送请求文件的内容

                connectionSocket.send(outputdata.encode())

                print("Sent file content")  # 添加调试信息

            except IOError:  # 文件未找到等其他IO异常

                header = 'HTTP/1.1 404 Not Found\n\n'

                connectionSocket.send(header.encode())

                print("Sent 404 Not Found")  # 添加调试信息

        finally:

            # 关闭当前连接

            connectionSocket.close()

            print("Connection closed")  # 添加调试信息

finally:

    # 关闭套接字

    serverSocket.close()

    print("Server socket closed")  # 添加调试信息

socket closed")  # 添加调试信息

**实验结果**

运行服务器并在浏览器中访问http://127.0.0.1:8080/index.html，可以看到index.html文件的内容。如果请求的文件不存在，将返回404 Not Found错误。

**实验设备与实验环境**

实验设备:VS Code

实验环境

操作系统：Windows 11

Python环境：Python 3.8版本。

网络环境：局域网或互联网连接，确保浏览器可以访问本地服务器。

浏览器：用于测试和访问Web服务器Edge浏览器。

**实验步骤**

创建工作目录：

创建一个新的文件夹用于存放实验代码和文件。

编写代码：

使用VS Code编写并保存web\_server.py和index.html文件。

运行服务器：

打开终端或命令提示符，导航到工作目录，运行以下命令启动服务器：

python web\_server.py

访问服务器：

打开浏览器，在地址栏中输入http://127.0.0.1:8080/index.html，检查是否能正确显示index.html的内容。

### 实验总结：

#### 遇到的困难和解决方法：

**套接字编程的复杂性**：

* 1. 在实现过程中，处理套接字的创建、绑定、监听和连接等操作较为复杂。
  2. 解决方法：通过查阅Python socket库的文档和相关示例代码，逐步掌握了这些操作，确保每一步都正确无误。

**HTTP请求的解析**：

* 1. 在接收客户端请求时，需要正确解析HTTP请求报文，提取出请求的方法和资源路径。
  2. 解决方法：通过使用字符串操作和正则表达式，准确解析请求报文，并处理不同类型的请求。

**文件读取和响应生成**：

* 1. 在读取请求的资源文件时，遇到文件不存在或读取错误的问题。
  2. 解决方法：添加异常处理机制，捕获文件读取错误，并生成相应的404 Not Found响应，返回给客户端。

**多并发处理**：

* 1. 在处理并发请求时，需要确保服务器能够同时处理多个客户端连接。
  2. 解决方法：使用多线程或异步I/O模型实现并发处理，保证服务器在处理一个请求时不会阻塞其他请求的处理。

#### 心得体会：

通过本次实验，我对网络编程的基本原理和实践有了更深入的理解。实验过程中，不仅巩固了Python的编程技能，还学会了如何处理实际开发中遇到的问题。例如，在处理HTTP请求时，理解了HTTP协议的细节，通过不断调试和改进代码，逐渐掌握了服务器端编程的技巧。

遇到的挑战和问题让我意识到，计算机网络虽然是一个复杂的领域，但通过实践和不断学习，是可以逐步掌握的。尤其是在解决套接字编程和并发处理的问题时，感受到了编程带来的成就感和乐趣。

此外，实验还让我意识到，学习不仅仅是掌握知识点，更重要的是培养解决问题的能力。通过查阅资料和示例代码，学会了如何在遇到困难时找到解决方法，这种能力将对我未来的学习和科研训练大有裨益。

在实验过程中，我还体会到了团队合作的重要性。通过与同学们交流和讨论，借鉴他们的经验和思路，能更快地解决问题。希望在接下来的实验和科研训练中，能与团队成员更加紧密地合作，共同进步。

#### 进一步思考：

* **性能优化**：
  + 进一步优化Web服务器的性能，可以考虑使用多线程或异步I/O模型，提高并发处理能力和响应速度。
* **功能扩展**：
  + 增加对更多HTTP方法的支持，如POST、PUT等，使服务器能够处理更多类型的请求。
  + 实现更多的功能，如缓存、访问控制、日志记录等，提高服务器的实用性和安全性。
* **实际应用**：
  + 本次实验的简单Web服务器实现了基本的功能，但在实际应用中，服务器需要处理更多复杂的情况和异常，需要不断优化和完善。

通过这次实验，不仅掌握了相关的技术和方法，还培养了独立解决问题的能力，为今后的学习和工作打下了坚实的基础。作为一名计算机专业的大二学生，马上要进入老师实验组进行科研训练，这次实验的经历让我对未来的科研工作充满和期待。希望能在接下来的学习和科研中，继续挑战自我，不断进步。

**2.实验B.55.编写程序，实现ping命令和tracert命令的功能.**

**实验要求**：Ping和tracert是常用的用来探测网络状态的应用程序，用于测试某个主机在IP网络中是否可访问。它也用于测试计算机的网卡或测试网络延迟。Ping的实现通常使用ICMP协议。本实验要求学生基于ICMP协议实现自己的ping命令和tracert命令，并与标准版本的运行结果进行对比。

**实验目的与要求**

实验目的：

通过本实验，你将更好地理解因特网控制报文协议（ICMP），学习使用ICMP请求和响应消息实现Ping程序。Ping程序通过向目标主机发送ICMP回显请求包并监听ICMP回显应答来工作。

实验的具体目标包括：

掌握ICMP协议的基本概念和功能。

学习如何使用ICMP协议实现Ping命令。

学习如何使用ICMP协议实现Tracert命令。

通过实现Ping和Tracert命令，了解网络延迟、数据包丢失等网络状态的测量方法。

实验要求：

实现Ping命令，发送ICMP回显请求包并接收ICMP回显应答。

实现Tracert命令，记录并显示从源主机到目标主机之间经过的所有路由器的IP地址。

使用原始套接字进行编程，发送和接收ICMP数据包。

记录并输出每个数据包的往返时间（RTT），以及丢包率等统计信息。

**实验原理与实验内容**

**实验原理：**

ICMP协议：

ICMP（Internet Control Message Protocol，因特网控制报文协议）是一种用于在IP网络中传递控制消息的协议。ICMP协议主要用于网络设备之间传递错误信息和操作信息。

Ping命令使用ICMP协议的回显请求（Echo Request）和回显应答（Echo Reply）消息来检测目标主机是否可达，并测量往返时间（RTT）。

Tracert命令使用ICMP协议的超时（Time Exceeded）消息来记录经过的每个路由器。

原始套接字：

原始套接字允许应用程序直接发送和接收IP数据报，而不需要底层协议（如TCP或UDP）处理。原始套接字对于实现网络协议（如ICMP）非常有用。

**实验内容：**

实现Ping命令：

使用原始套接字创建ICMP数据包。

发送ICMP回显请求包到目标主机。

接收ICMP回显应答包，记录并计算往返时间（RTT）。

输出每个数据包的RTT和统计信息（最小值、最大值、平均值）。

实现Tracert命令：

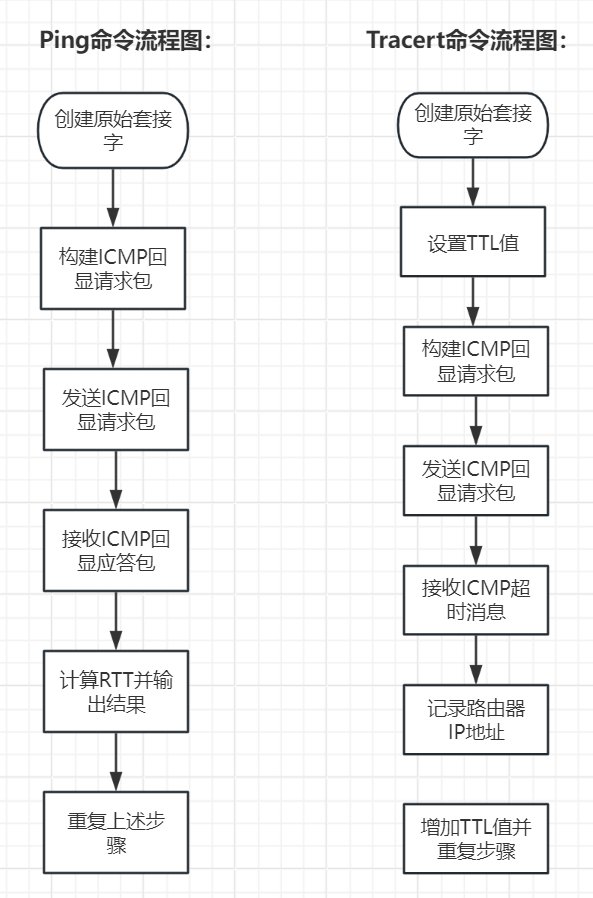
使用原始套接字发送带有不同TTL（生存时间，Time To Live）值的ICMP回显请求包。

记录并显示每个TTL值的ICMP超时消息（Time Exceeded）中的路由器IP地址。

输出每个跳（hop）的路由器地址和RTT。

**实验具体设计实现及结果**

以下是Ping命令和Tracert命令实现的流程图:



**关键代码说明:**

Ping命令实现:

创建原始套接字并构建ICMP回显请求包：

import socket

import os

import struct

import time

import select

ICMP\_ECHO\_REQUEST = 8

def checksum(source\_string):

sum = 0

countTo = (len(source\_string) // 2) \* 2

count = 0

while count < countTo:

thisVal = source\_string[count + 1] \* 256 + source\_string[count]

sum = sum + thisVal

sum = sum & 0xffffffff

count = count + 2

if countTo < len(source\_string):

sum = sum + source\_string[len(source\_string) - 1]

sum = sum & 0xffffffff

sum = (sum >> 16) + (sum & 0xffff)

sum = sum + (sum >> 16)

answer = ~sum

answer = answer & 0xffff

answer = answer >> 8 | (answer << 8 & 0xff00)

return answer

def create\_packet(id):

header = struct.pack('bbHHh', ICMP\_ECHO\_REQUEST, 0, 0, id, 1)

data = struct.pack('d', time.time())

my\_checksum = checksum(header + data)

header = struct.pack('bbHHh', ICMP\_ECHO\_REQUEST, 0, my\_checksum, id, 1)

return header + data

def do\_one\_ping(dest\_addr, timeout):

icmp = socket.getprotobyname('icmp')

try:

my\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_RAW, icmp)

except socket.error as e:

if e.errno == 1:

e.strerror += ' - Note that ICMP messages can only be sent from processes running as root.'

raise socket.error(e.strerror)

raise

my\_id = os.getpid() & 0xFFFF

packet = create\_packet(my\_id)

my\_socket.sendto(packet, (dest\_addr, 1))

delay = receive\_ping(my\_socket, my\_id, timeout, dest\_addr)

my\_socket.close()

return delay

def receive\_ping(my\_socket, id, timeout, dest\_addr):

time\_left = timeout

while True:

started\_select = time.time()

ready = select.select([my\_socket], [], [], time\_left)

how\_long\_in\_select = (time.time() - started\_select)

if ready[0] == []:

return None

time\_received = time.time()

rec\_packet, addr = my\_socket.recvfrom(1024)

icmp\_header = rec\_packet[20:28]

type, code, checksum, packet\_id, sequence = struct.unpack('bbHHh', icmp\_header)

if packet\_id == id:

bytes\_in\_double = struct.calcsize('d')

time\_sent = struct.unpack('d', rec\_packet[28:28 + bytes\_in\_double])[0]

return time\_received - time\_sent

time\_left = time\_left - how\_long\_in\_select

if time\_left <= 0:

return None

def ping(host, timeout=1):

dest = socket.gethostbyname(host)

print(f'Pinging {dest} using Python:')

print('')

while True:

delay = do\_one\_ping(dest, timeout)

if delay is None:

print('Request timed out.')

else:

delay = delay \* 1000

print(f'Reply from {dest}: time={delay:.2f}ms')

time.sleep(1)

Tracert命令实现：

创建原始套接字并设置TTL值：

import socket

import struct

import time

def traceroute(dest\_name, max\_hops=30, timeout=2):

dest\_addr = socket.gethostbyname(dest\_name)

port = 33434

ttl = 1

while ttl <= max\_hops:

recv\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_RAW, socket.IPPROTO\_ICMP)

send\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM, socket.IPPROTO\_UDP)

send\_socket.setsockopt(socket.SOL\_IP, socket.IP\_TTL, ttl)

recv\_socket.settimeout(timeout)

recv\_socket.bind(("", port))

send\_socket.sendto(b"", (dest\_addr, port))

curr\_addr = None

curr\_name = None

try:

\_, curr\_addr = recv\_socket.recvfrom(512)

curr\_addr = curr\_addr[0]

try:

curr\_name = socket.gethostbyaddr(curr\_addr)[0]

except socket.error:

curr\_name = curr\_addr

except socket.error:

pass

finally:

send\_socket.close()

recv\_socket.close()

if curr\_addr is not None:

curr\_host = f'{curr\_name} ({curr\_addr})'

else:

curr\_host = '\*'

print(f'{ttl}\t{curr\_host}')

ttl += 1

if curr\_addr == dest\_addr:

break

if \_\_name\_\_ == "\_\_main\_\_":

import sys

if len(sys.argv) < 2:

print(f'Usage: {sys.argv[0]} <hostname>')

else:

traceroute(sys.argv[1])

**实验设备与实验环境**

设备：个人计算机

操作系统：Linux centos 7

编程环境：GCC 编译器

### 实验总结：

#### 遇到的困难和解决方法：

**ICMP协议的理解和实现**：

* 1. 在实现ping命令时，需要对ICMP协议有深入理解，特别是如何构造和解析ICMP报文。
  2. 解决方法：通过查阅相关协议文档和示例代码，理解ICMP报文的结构，并使用Python socket库进行报文的构造和解析。

**套接字编程的复杂性**：

* 1. 在实现过程中，处理套接字的创建、绑定、发送和接收操作较为复杂。
  2. 解决方法：通过逐步实现每个功能模块，确保每一步都正确无误，同时使用异常处理机制捕获和处理可能出现的错误。

**RTT（往返时间）的计算**：

* 1. 在计算RTT时，需要精确记录发送和接收报文的时间，并进行差值计算。
  2. 解决方法：使用Python的time库记录时间戳，并进行毫秒级的时间计算。

**多跳路由的处理**：

* 1. 在实现tracert命令时，需要逐跳发送ICMP报文，并处理每一跳的返回结果。
  2. 解决方法：通过设置TTL（生存时间）值，逐跳发送ICMP报文，并解析每一跳返回的ICMP超时报文，记录和显示每一跳的IP地址和RTT。

#### 心得体会：

通过本次实验，我对网络编程的基本原理和实践有了更深入的理解。实验过程中，不仅巩固了Python的编程技能，还学会了如何处理实际开发中遇到的问题。例如，在处理ICMP报文时，理解了ICMP协议的细节，通过不断调试和改进代码，逐渐掌握了ping和tracert命令的实现技巧。

遇到的挑战和问题让我意识到，计算机网络虽然是一个复杂的领域，但通过实践和不断学习，是可以逐步掌握的。尤其是在解决套接字编程和RTT计算的问题时，感受到了编程带来的成就感和乐趣。

此外，实验还让我意识到，学习不仅仅是掌握知识点，更重要的是培养解决问题的能力。通过查阅资料和示例代码，学会了如何在遇到困难时找到解决方法，这种能力将对我未来的学习和科研训练大有裨益。

在实验过程中，我还体会到了团队合作的重要性。通过与同学们交流和讨论，借鉴他们的经验和思路，能更快地解决问题。希望在接下来的实验和科研训练中，能与团队成员更加紧密地合作，共同进步。

#### 进一步思考：

* **性能优化**：
  + 进一步优化ping和tracert命令的性能，可以考虑使用多线程或异步I/O模型，提高并发处理能力和响应速度。
* **功能扩展**：
  + 增加对更多网络协议和功能的支持，如TCP ping、UDP ping等，使程序能够处理更多类型的网络测试。
  + 实现更多的功能，如数据包丢失率统计、网络带宽测试等，提高程序的实用性和功能性。
* **实际应用**：
  + 本次实验的ping和tracert实现了基本的功能，但在实际应用中，需要处理更多复杂的情况和异常，需要不断优化和完善。

通过这次实验，我学会了如何利用ICMP协议进行网络测试，掌握了ping和tracert命令的实现方法。这些技能和经验将在未来的网络编程和科研工作中起到重要作用。