电 子 科 技 大 学

UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

工业控制网络报告二

Experimental Report

报告题目 TCP/IP 通信测试程序实现数据传输

学　　院 机械与电气工程学院

专　　业 机器人工程

学　　号 2021040902007

作者姓名 经彭宇

指导教师 袁太文

目 录

第一章 基本理论

1.1 TCP/IP 定义及工作原理

TCP/IP 是一组通信协议，用于在网络中实现数据传输和通信。

TCP 是一种面向连接的协议，确保数据在网络中可靠地传输，它提供了可靠的数据传输机制，包括数据分段、序列号、确认应答、流量控制和拥塞控制等功能。TCP 使用三次握手建立连接，客户端发送 syn->服务端回复 syn+ack->客户端回复 ack，在第一和第二步可确认客户端输入、输出是通的，在第二和第三部可确认服务端输入输出是通的。TCP 三次握手只在数据发送阶段之前，建立连接，客户端和服务端在内核中创建资源，用于后续数据的发送，三次握手之后，双方内核开辟资源为对方提供服务，即为面向连接了。 在连接断开时使用四次挥手进行关闭，客户端发起 fin，请求释放资源。服务端收到，回复 fin +ack 服务端发起 fin 请求释放资源，客户端回复 ack。

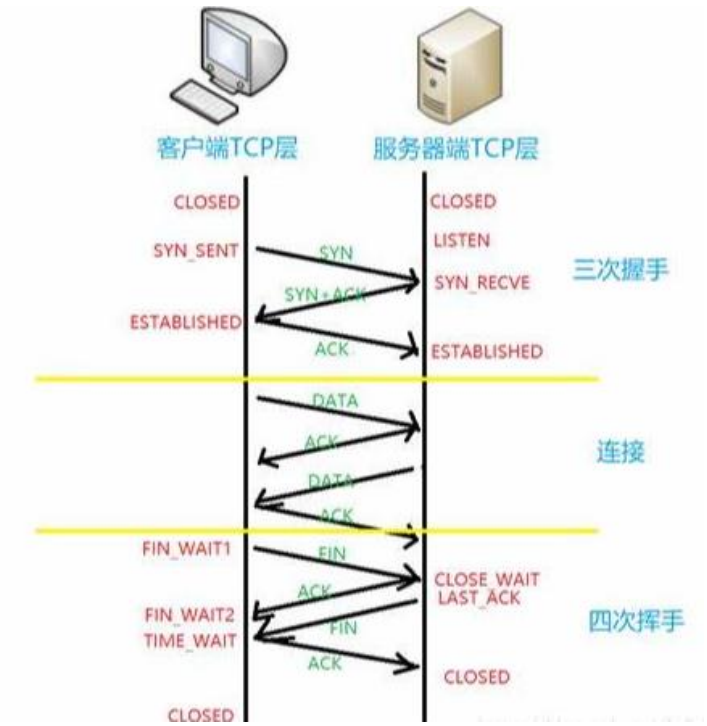

图 1-1 TCP 三次握手四次挥手

IP 是一种网络层协议，负责在网络中路由数据包并实现主机之间的通信。IP 地址用于标识网络中的主机和设备，IPv4 使用 32 位地址，而 IPv6 使用 128 位地址。IP 数据包包含源 IP 地址和目标 IP 地址，用于确定数据包的发送和接收方。

套接字是实现网络通信的一种机制，允许应用程序通过网络发送和接收数据。TCP/IP 通信中使用的套接字包括服务器套接字（监听连接）和客户端套接字（建立连接）。

TCP 通信包括服务器和客户端两端，服务器监听指定端口并等待连接，客户端连接服务器并发送数据。通信过程中，客户端和服务器之间进行数据交换，使用 TCP 提供的数据传输功能保证数据可靠传输。通信完成后，客户端和服务器可以关闭连接，释放资源。

1.2 Socket

在 Python 中，使用内置的 socket 模块可以创建 Socket 对象，并进行网络通信。通过 Socket，可以实现各种类型的网络应用，如 Web 服务器、聊天程序、远程控制等。Socket 通信涉及到客户端和服务器两个角色，客户端发送请求或数据，服务器接收并响应。

Socket 是一种应用程序编程接口（API），它允许应用程序通过网络进行通信。它提供了一种机制，使得可以在网络上发送和接收数据，从而实现了不同设备之间的通信和数据交换。在 Python 中，可以使用内置的 socket 模块来创建套接字，以便在客户端和服务器之间建立连接并进行通信。

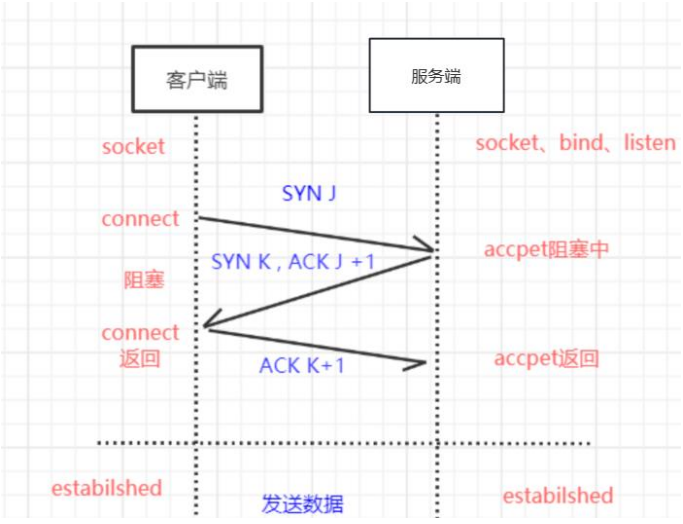

图 1-2 socket 三次握手快速建立连接

第二章 设计分析

2.1 总体思路

本报告选择使用现有的网络库，Python 的 socket 模块。在编写 TCP/IP 通信测试程序时，需要实现服务器和客户端两个部分。

其中服务器负责监听连接、接收数据、处理数据并发送响应，客户端负责连接服务器、发送数据并接收服务器响应。

使用 Python 的 `socket` 模块，编程语言提供的网络套接字 API 实现 TCP 服务器和客户端，客户端与服务器之间通过套接字建立连接，客户端发送消息给服务器，服务器接收并回复确认消息。

2.2 实现方法

首先设计服务器端程序，即编写一个 TCP 服务器程序，用于监听客户端连接并处理数据。需要确定服务器程序的功能，包括创建套接字、绑定端口、监听连接、接收数据、处理数据、发送响应等。

其次要设计客户端程序，编写一个 TCP 客户端程序，用于连接服务器并发送测试数据。同时确定客户端程序的功能，包括创建套接字、连接服务器、发送数据、接收服务器响应等。

最后测试和验证，在实际的网络环境中运行测试程序，验证通信功能和性能。

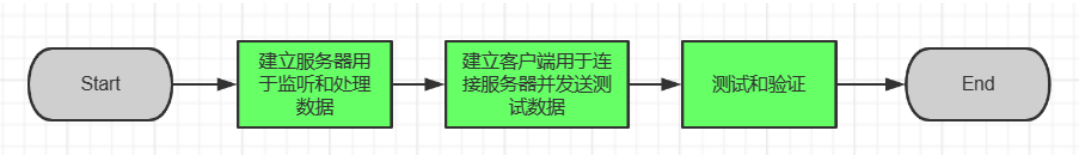

图 2-1 实现流程

2.3 代码思路

（1）客户端部分

首先，客户端使用`socket.create_connection`函数创建一个套接字对象并连接到指定的服务器地址和端口。

```
sock = socket.create_connection(('127.0.0.1', 7897))
```

图 2- 2 创建一个套接字对象连接到指定的服务器地址和端口

使用`getConstants`函数获取了 IP 协议、地址族和套接字类型的常量名称，便于后续打印输出。

```
def getConstants(prefix):
    return {
        getattr(socket, n): n
        for n in dir(socket)
        if n.startswith(prefix)
    }
```

图 2- 3 getConstants 函数

发送一条消息给服务器，然后等待接收服务器的回复。最后，关闭套接字连接。

```
try:
    msg = b"Are you there?"
    sock.sendall(msg)
    data = sock.recv(1024)
    print(data.decode())
finally:
    sock.close()
```

图 2-4 尝试发送消息

（2） 服务器部分

服务器创建一个套接字对象，指定地址族为 IPv4（`AF_INET`）和套接字类型为 TCP（`SOCK_STREAM`）。

```
# 1.创建一个套接字,
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

图 2-5 创建套接字对象

使用`bind`函数将套接字与服务器地址关联，并使用`listen`函数将套接字设置为监听模式，等待客户端连接。

```
# 2.使用bind()函数将套接字与服务器地址关联
sock.bind(('localhost', 7897))
# 3.调用listen()函数将套接字设置为服务器模式
sock.listen(1)
```

图 2-6 相关函数

通过循环持续监听客户端连接请求。 当有客户端连接时，接受连接并获取客户端地址。 接收客户端发送的数据，并回复确认消息。最后，关闭连接。

```
while True:
    # 4.调用accept()等待客户端的消息连接
    # 如果有客户端进行连接，那么accept()函数会返回一个打开的连接与客户端地址
    connection, client_address = sock.accept()
    print("连接客户端地址：", client_address)
    try:
        # 5.指明一个缓冲区，该缓冲区用来存放recv函数接收到的数据
        data = connection.recv(1024)
        print(data)
        if data:
            # 6.通过sendall()进行回传客户端数据。
            connection.sendall("已接受到数据".encode())
        else:
            print("客户端没有发送数据，不需要传送数据")
```

图 2-7 循环监听

第三章 设计结果

3.1 代码

（1）Client.py

```
import socket
# 获取匹配开头字符串的所有属性值
def getConstants(prefix):
    return {
        getattr(socket, n): n
        for n in dir(socket)
        if n.startswith(prefix)
    }


ipproto_str = getConstants("IPPROTO_")
family_str = getConstants("AF_")
type_str = getConstants("SOCK_")

# 端口号具体看电脑的代理端口号
sock = socket.create_connection(('127.0.0.1', 7897))
print(ipproto_str[sock.proto])
print(family_str[sock.family])
print(type_str[sock.type])

try:
    msg = b"Are you there?"
    sock.sendall(msg)
    data = sock.recv(1024)
    print(data.decode())
finally:
    sock.close()
```

（2）Sever.py

```
import socket

# 1.创建一个套接字，
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# 2.使用 bind()函数将套接字与服务器地址关联
sock.bind(('localhost', 7897))
# 3.调用 listen()函数将套接字设置为服务器模式
sock.listen(1)

while True:
    # 4.调用 accept()等待客户端的消息连接
    # 如果有客户端进行连接，那么 accept()函数会返回一个打开的连接与客户端地址
    connection, client_address = sock.accept()
    print("连接客户端地址：", client_address)
    try:
        # 5.指明一个缓冲区，该缓冲区用来存放 recv 函数接收到的数据
        data = connection.recv(1024)
```

```
        print(data)
        if data:
            # 6.通过 sendall()进行回传客户端数据。
            connection.sendall("已接受到数据".encode())
        else:
            print("客户端没有发送数据，不需要传送数据")
    finally:
        #7.需要使用 close()进行关闭清理
        connection.close()
```

3.2 运行结果

（1）服务端

```
D:\Python\python.exe F:\Pycharm\TCP\Sever.py
连接客户端地址:  ('127.0.0.1', 4948)
b'Are you there?'
```

图 3-1 服务端运行结果

（2）客户端

```
D:\Python\python.exe F:\Pycharm\TCP\Client.py
IPPROTO_IP
AF_INET
SOCK_STREAM
已接受到数据
```

图 3-2 客户端运行结果

第四章 结论展望

4.1 结论

通过 Python 可以很容易地实现客户端-服务器通信。然而，这只是一个基础的示例，实际应用中可能涉及到更多复杂的情况，例如多客户端同时连接、数据处理和安全性等问题。在实际应用中，需要根据具体需求进行更加细致和完善的设计和实现。同时，可以考虑使用更高级的网络通信框架或协议，如 Twisted、Flask 等，来简化开发和提高性能。

4.2 展望及改进

这个代码示例虽然实现了基本的客户端-服务器通信功能，但仍存在一些缺点和可以改进的地方。首先是单线程阻塞，服务器端采用单线程模式处理客户端连接请求，当有多个客户端同时连接时，会导致阻塞，降低服务器的响应速度和并发性能。其次是简单的数据处理，服务器端仅简单地接收客户端发送的数据并回复确认消息，缺乏对数据的解析、处理和存储功能，无法处理复杂的业务逻辑。最后是缺乏错误处理，代码中未对可能出现的异常情况进行充分的错误处理，如网络连接异常、数据传输错误等，缺乏容错机制。

因此有以下改进方向：

（1）多线程处理：服务器端可以采用多线程或多进程的方式处理客户端连接请求，提高并发处理能力，增强系统的稳定性和性能。

（2）数据处理和存储：服务器端可以增加对数据的解析、处理和存储功能，实现更复杂的业务逻辑，如数据库操作、文件处理等。

（3）错误处理和容错机制：添加完善的错误处理机制，捕获并处理可能出现的异常情况，保证系统的稳定性和可靠性。

（4）优化性能：优化网络通信和数据传输方式，减少数据传输延迟，提高系统的响应速度和吞吐量。

通过以上改进方向的实施，可以使通信更加高效，更适用于实际的网络应用场景，并能够应对更复杂的业务需求和挑战。