《网络与通信》课程实验报告

实验 2：Socket 通信编程

姓名	胡才郁	院系	计算机学院	学号	20121034
任课教师	刘通		指导教师	刘通	
实验地点	计 708		实验时间		
实验课表现	出勤、表现得分(10)		实验报告得分(40)		实验总分	
	操作结果得分(50)					

实验目的：

1. 掌握 Socket 编程过程；
2. 编写简单的网络应用程序。

实验内容：

利用你选择的任何一个编程语言，分别基于 TCP 和 UDP 编写一个简单的 Client/Server 网络应用程序。具体程序要求参见《实验指导书》。

要求以附件形式给出：

- 系统概述：运行环境、编译、使用方法、实现环境、程序文件列表等；
- 主要数据结构；
- 主要算法描述；
- 用户使用手册；
- 程序源代码；

实验要求：（学生对预习要求的回答）（10 分）　　得分：

- **Socket编程客户端的主要步骤**

  对于客户端，此处以Python语言的TCP服务器为例：

  1、创建套接字并链接至远端地址

  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

  s.connect()

  2、链接后发送数据和接收数据

  s.sendall()

  s.recv()

  3、传输完毕后，关闭套接字

- **Socket编程服务器端的主要步骤**

  对于服务器端，此处以Python语言的TCP服务器为例：

  1、 创建套接字，绑定套接字到本地IP与端口

  s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

  s.bind()

  2、开始监听链接

  s.listen()

3、进入循环，不断接受客户端的链接请求

While True:

s.accept()

4、接收客户端传来的数据，并且发送给对方发送数据

s.recv()

s.sendall()

5、传输完毕后，关闭套接字

s.close()

实验过程中遇到的问题如何解决的？（10 分）	得分：

问题 1：Client 与 Server 不能使用同一个线程运行。

在此次实验中，由于一台主机将同时启动两个程序，分别作为服务端和客户端，所以需要将 Client 与 Server 分别使用不同的线程进行运行。我同时为 Client 与 Server 使用 PyQt 编写图形化界面，便于展示相关信息与用户交互操作。此处的 Server 端、Client 端可以分别选择启用 TCP 服务或者 UDP 服务进行网络通信。

然而，Server 端与 Clinet 端无法使用同一个线程运行。并且 PyQt5 的 UI 界面运行服务器端程序时，UI 界面需要占用一个线程，还需要开启另一个线程运行 TCP 或者 UDP 服务，当接受到由客户端发来的请求时，需要 TCP 或 UDP 线程将接收到的信息回显到 UI 界面之上，这就涉及到了线程的通信。因此实际操作时对于主类 Status 设置按下按钮后的操作，开启一个新线程运行 TCP 或 UDP。

下图以 TCP 服务器端为例，展示了新线程的开启部分，解决了线程冲突的问题。

```
class Stats:
    def __init__(self) -> None:
        # 从文件中加载UI定义
        self.ui: Any | None = uic.loadUi("TCPServer.ui")
        # 给文本输入框绑定事件
        self.ui.server_ip.setText(server_ip)
        self.ui.server_port.setText(server_port)
        self.ui.server_ip.textChanged.connect(self.handleServerIpChanged)
        self.ui.server_port.textChanged.connect(self.handleServerPortChanged)
        self.ui.start.clicked.connect(self.handleStartButtonClicked)
        # 成员变量
        self.server_message: Literal[''] = ''
        self.tcp: TCPThread = TCPThread()
        self.handleStartButtonClicked()

    # 启动server
    def handleStartButtonClicked(self) -> None:
        self.ui.start.setEnabled(False)
        # 开启一个新线程
        self.tcp.start()
        self.tcp.show.connect(self.display)
```

图 1. TCP 服务器端多线程处理

当在 UI 界面中输入文字，并按下发送按钮时，需要将 UI 界面框中的文字发送，首先开启一个新的线程，创建 Socket 对象，选择相应的协议，并将此线程中处理得到的内容返回给 UI 界面的所在线程。

问题 2：Server 端启用时，出现端口占用问题，导致启动失败。

在本实验中，我将 Server 端的 TCP 或者 UDP 绑定端口 9001，在编写代码调试的过程中，如果上一个 Server 仍未关闭，9001 端口仍被占用时，此时新创建的 Server 无法监听相同的 9001 端口，导致第二个 server 无法运行，并且出现程序卡死的现象。

解决方法：首先在命令行中使用 lsof 指令查询当前 9001 端口的占用情况，并且使用 kill 指令停止原先的进程。由下图可以看出，Python3.9 所在的进程号 pid 为 47642 的进程占用了目前的 9001 端口，所以需要 kill 掉此进程。

```
→  sourde_code lsof -i:9001
COMMAND     PID     USER   FD   TYPE             DEVICE SIZE/OFF NODE NAME
python3.9 47642 silence    8u  IPv4 0xa98467ef16c9ae4f      0t0  TCP *:etlservicemgr (LISTEN)
```

图 2. 查看相应端口号的占用情况

为了保证实验时不出现多次请求绑定相同端口的情况，此处添加代码，添加异常处理部分，对于反复请求绑定相同端口号的行为做出限制。

```
try:
    server_socket.bind(('', int(server_port)))
except OSError:
    server_socket.close()
    self.show.emit('clear')
    self.show.emit('通常每个套接字地址(协议/网络地址/端口)只允许使用一次!' + '\n')
    self.show.emit('请修改端口号后重新启动~' + '\n')
    return
```

图 3. 对于端口占用做出异常处理

最后，当多个服务器端请求使用相同的端口号时，在 GUI 页面上做出相应的用户提示。如下图，1 号为第一个 TCP 服务器端，监听 9001 端口，2 号为第二个 TCP 服务器端，当它也试图监听相同的 9001 端口时，反馈给相关的用户提示。

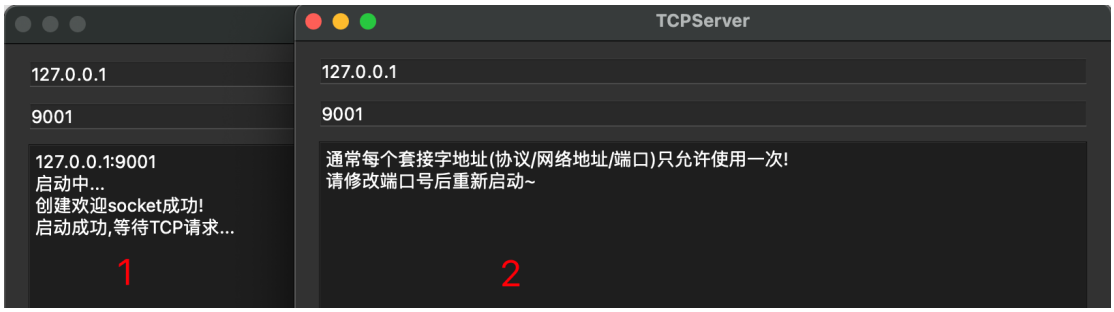

图 4. 图形化界面上对于端口占用的现实提示

问题 3：初始接受到的流信息未经处理导致类型错误，无法返回处理后的信息。

解决方法：将字节流转换为字符流。接受信息代码如下：

message, client_address = server_socket.recvfrom(2048)

以上代码含义为 Server 端的 Socket 接受信息，默认接受到的 message 为字节流，因此，当接受到后需要在 Server 端对其进行类型转换，将字节流转换为字符流后才可以对相应的信息进行处理。下图红框部分对于 message 进行 decode 操作，并将 decode 得到的字符流转换为大写。

```
while True:
    message: bytes, client_address: _RetAddress = server_socket.recvfrom(2048)
    self.show.emit('收到来自:' + str(client_address[0]) + '的信息: ' + message.decode() + '\n')
    modified_message: str = message.decode().upper()
    self.show.emit('返回修改后的信息: ' + modified_message + '\n')
    server_socket.sendto(modified_message.encode(), client_address)
    self.show.emit('---------------UDP请求完成---------------' + '\n')
```

图 5. 字节流转换为字符流

实验结果如下，Client 端发送一串字母 abcdefg 后，Server 端进行字节流处理后，将大写的 ABCDEFG 发送给 Client 端。

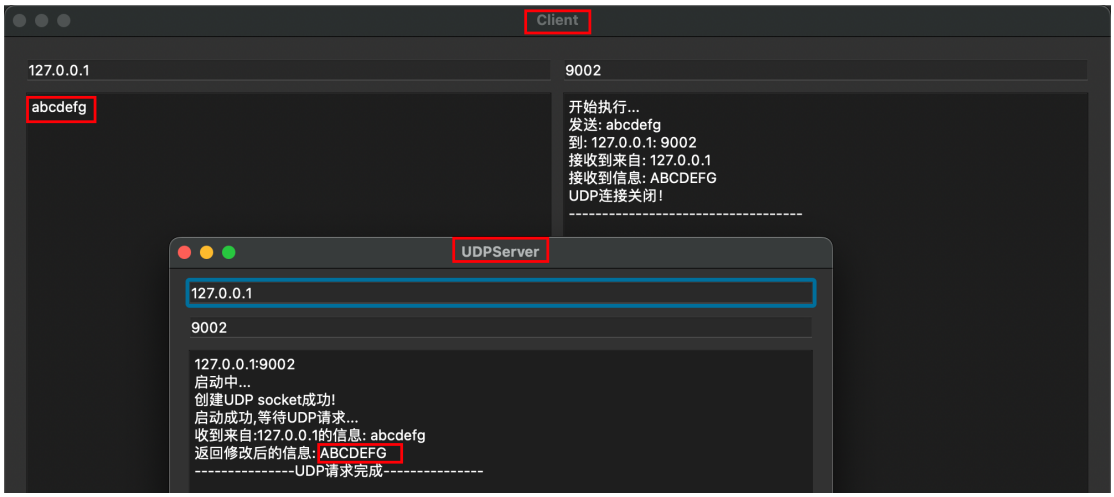

图 6. 实验结果展示

本次实验的体会（结论）（10 分）	得分：

网络通信需要以下三个要素：

1、IP 地址

要想让网络中的计算机能够互相通信，必须为每台计算机指定一个标识号，通过这个标识号来指定要接收数据的计算机和识别发送的计算机，而 IP 地址就是这个标识号。也就是设备的标识。

2、端口

网络的通信，本质上是两个应用程序的通信。每台计算机都有很多的应用程序，那么在网络通信时，如何区分这些应用程序呢？如果说 IP 地址可以唯一标识网络中的设备，那么端口号就可以唯一标识设备中的应用程序了。也就是应用程序的标识。

3、协议

通过计算机网络可以使多台计算机实现连接，位于同一个网络中的计算机在进行连接和通信时需要遵守一定的规则，这就好比在道路中行驶的汽车一定要遵守交通规则一样。在计算机网络中，这些连接和通信的规则被称为网络通信协议，它对数据的传输格式、传输速率、传输步骤等做了统一规定，通信双方必须同时遵守才能完成数据交换。常见的协议有 UDP 协议和 TCP 协议。

TCP 和 UDP 都是传输层协议，概括而言，TCP 对应的是可靠性要求高的应用，而 UDP 对应的则是可靠性要求低、传输经济的应用。

在本次实验中，我使用 Python 语言完成了基于 Scoket 的 TCP\UDP 网络编程，并且使用图形化 UI 界面对于网络通信结果进行了展示。下图展示了 Server 端与 Client 端的基本界面。

图 7. Server 端图形化界面

图 8. Client 端图形化界面

思考题：（10 分）

思考题 1：（4 分）	得分：

你所用的编程语言在 Socket 通信中用到的主要类及其主要作用。

Python 提供了两个级别访问的网络服务：

低级别的网络服务支持基本的 Socket，它提供了标准的 BSD Sockets API，可以访问底层操作系统 Socket 接口的全部方法。

高级别的网络服务模块 SocketServer， 它提供了服务器中心类，可以简化网络服务器的开发。

在 Python 语言中，socket 编程客户端主要分为以下几个步骤，分别将相应的函数接口以及代码如以下所示：

1、UDP 发送数据的步骤

```
# 创建发送端的Socket对象(DatagramSocket)
client_socket = socket(AF_INET, SOCK_DGRAM)
# 向目的主机发送报文,并且将字符串转换为字节
client_socket.sendto(client_message.encode(), (self.target_ip, int(self.target_port)))
#等待接收来自服务器的数据，缓存长度2048,从服务器接收的数据和服务器地址
rec_message, server_address = client_socket.recvfrom(2048)
# 关闭发送端，释放资源
client_socket.close()
```

图 9. UDP 发送数据的步骤

2、TCP 发送数据的步骤

```
# 创建客户端的Socket对象(Socket)
Socket(String host, int port)
# 获取输出流，写数据
OutputStream getOutputStream()
# 关闭发送端，释放资源
client_socket.close()
```

图 10. TCP 发送数据的步骤

在 Python 语言中，socket 编程服务器端主要分为以下几个步骤：

1、UDP 接收数据的步骤

```
# 创建服务器端的Socket对象(DatagramSocket)
server_socket = socket(AF_INET, SOCK_DGRAM)
Socket对象绑定端口
server_socket.bind(('', int(server_port)))
# 接收客户端发送信息与地址
message, client_address = server_socket.recvfrom(2048)
```

图 11. UDP 接收数据的步骤

2、TCP 接收数据的步骤

```
# 创建欢迎socket
server_socket = socket(AF_INET, SOCK_STREAM)
# 绑定相应端口
server_socket.bind(('', int(server_port)))
# 完成握手，创建连接socket
connection_socket, address = server_socket.accept()
# 接收客户传递的信息
message = connection_socket.recv(2048)
# 关闭本次链接，释放资源
connection_socket.close()
```

图 12. TCP 接收数据的步骤

思考题 2：（6 分）　　得分：

说明 TCP 和 UDP 编程的主要差异和特点。

TCP 协议

传输控制协议 (Transmission Control Protocol)

TCP 协议是面向连接的通信协议，即传输数据之前，在发送端和接收端建立逻辑连接，然后再传输数据，它提供了两台计算机之间可靠无差错的数据传输。在 TCP 连接中必须要明确客户端与服务器端，由客户端向服务端发出连接请求，每次连接的创建都需要经过“三次握手”。

三次握手：TCP 协议中，在发送数据的准备阶段，客户端与服务器之间的三次交互，以保证连接的可靠。

第一次握手，客户端向服务器端发出连接请求，等待服务器确认。

第二次握手，服务器端向客户端回送一个响应，通知客户端收到了连接请求。

第三次握手，客户端再次向服务器端发送确认信息，确认连接。

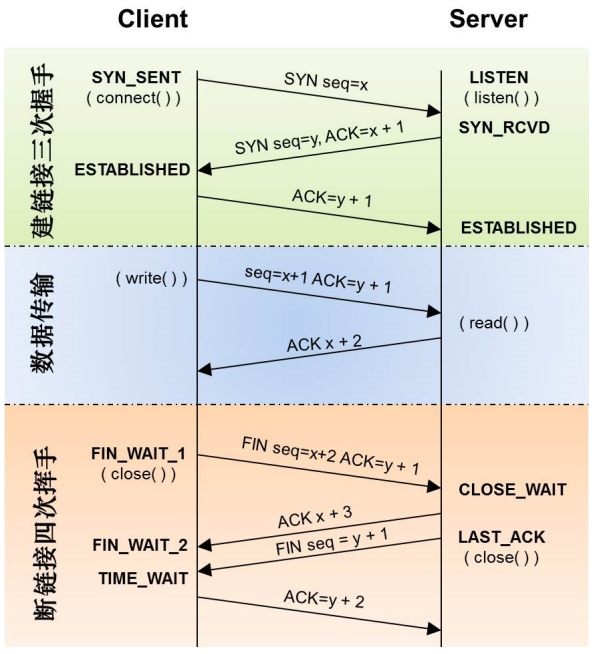

图 13. TCP 三次握手与四次挥手

UDP 协议

用户数据报协议(User Datagram Protocol)

UDP 是无连接通信协议，即在数据传输时，数据的发送端和接收端不建立逻辑连接。简单来说，当一台计算机向另外一台计算机发送数据时，发送端不会确认接收端是否存在，就会发出数据，同样接收端在收到数据时，也不会向发送端反馈是否收到数据。

由于使用 UDP 协议消耗资源小，通信效率高，所以通常都会用于音频、视频和普通数据的传输。

例如视频会议通常采用 UDP 协议，因为这种情况即使偶尔丢失一两个数据包，也不会对接收结果产生太大影响。但是在使用 UDP 协议传送数据时，由于 UDP 的面向无连接性，不能保证数据的完整性，因此在传输重要数据时不建议使用 UDP 协议。

以下是 TCP 与 UDP 两者之间的主要差异：

表 1. TCP 与 UDP 的主要差异

	UDP	TCP
连接方式	无连接	面向连接
可靠性	不可靠，不使用流量控制和拥塞控制	可靠，使用流量控制和拥塞控制
对象个数	支持一对一，一对多，多对一和多对多交互通信	只能一对一
传输方式	报文	字节流
头部长度	首部开销小，仅 8 字节	首部最小 20 字节，最大 60 字节
使用场景	适用于实时应用，流媒体服务等	用于要求可靠传输的应用

指导教师评语：

日期：