附录

1. 系统概述

本实验代码 Python 语言编写，由于 Python 语言的跨平台性，理论上主流的操作系统均可运行，本次试验中的运行环境为我的个人电脑，具体环境如下：

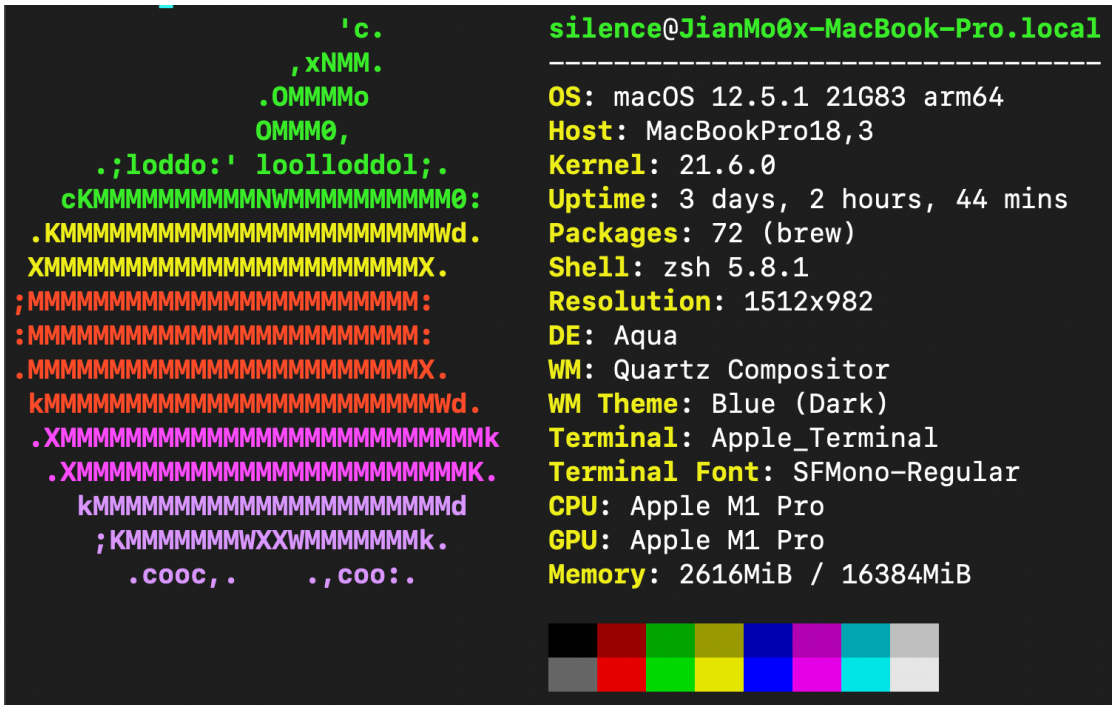

图 1. 运行环境

程序文件列表如下图所示，其中，以.ui 结尾的是 PyQt 的 UI 文件，负责配置图形化界面。gui_client.py 为客户端程序，TCPServer.py\UDPServer.py 分别为 TCP、UDP 的服务器端程序。

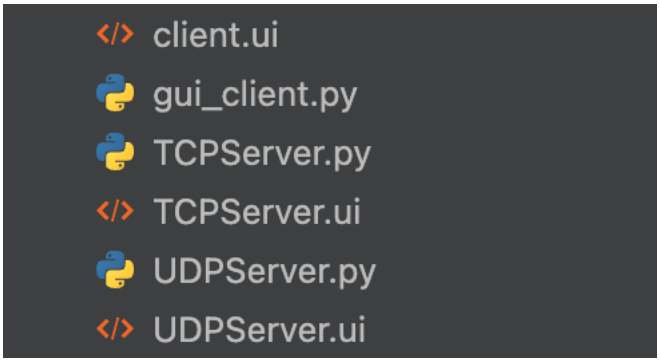

图 2. 程序文件列表

2. 主要数据结构

主要数据结构为 Socket()类，在实验报告中均有介绍。

针对 TCP 的 Socket 为：

socket(socket.AF_INET, socket.SOCK_STREAM)

针对 UDP 的 Socket 为：

socket(socket.AF_INET, socket.SOCK_STREAM)

3. 主要算法描述；

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 3. 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 4. 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 5. UDP 接收数据的步骤

2、TCP 接收数据的步骤

```python
# 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 6. TCP 接收数据的步骤

4. 用户使用手册

运行本程序需要使用 python 分别运行 gui_client.py 文件与 TCPServer.py 或 UDPServer.py 中的任一一个。随后根据图形化界面的提示操作即可。

图 7. 启动客户端程序

图 8. 启动服务端程序