华东师范大学软件工程学院实验报告

实验课程：计算机网络实践	姓名：余明蕊	学号：10204500101
实验名称：Lab7 Socket	实验日期：2021.12.17	指导老师：刘献忠

实验目的

- 熟悉socket编程的基本原理
- 掌握简单的套接字编程
- 掌握通过socket编程实现C/S程序的基本方法
- 了解应用层和运输层的作用及相关协议的工作原理和机制

实验内容与实验步骤

server

- 能在标准输出打印客户端发送的消息
- 支持5个以上客户端同时发送消息并逐一打印
- 绑定至错误的端口号时能提示出错信息

client

- 能从标准输入或文件接收消息
- 标准输入消息以两次回车作为结束标志
- 连接至错误的IP地址/端口号时能提示出错信息

整体

- 支持在localhost及在两台不同机器上运行
- 支持长文本消息 （不少于 20KB），有缓冲区管理
- 容错性好，无闪退

加分项

- 支持双工通信

所有得分点将在代码分析中加粗体现。

实验环境

- Windows
- Python

实验过程与分析

服务端源代码

```
import socket
import threading

clients = []
lock = threading.Lock()


def broadcast():
    while True:
        message = input().rstrip()
        # if the message is empty, ignore it
        if message == '':
            continue
        for client in clients:  # for each client
            a = 0  # start at the beginning of the message
            while len(message) - a >= 8192:  # send message in 8192 bytes
                client[2].sendall(message[a:a + 8192].encode())
                a += 8192  # move to the next 8192 bytes
            client[2].sendall(message[a:].encode())  # send the rest of the message


def reception():
    client = (address[0], address[1], connection)
    clients.append(client)  # add the client to the list
    print(f">>>{client[0:2]} have connected.")
    while True:
        # receive message
        try:
            message = connection.recv(8192).decode()
        except Exception:
            # remove the client from the list
            lock.acquire()  # lock the list
            clients.remove(client)
            lock.release()
            print(f"{client[0:2]} have disconnected.")
        # print message
        if message == '':
            print(f"{client[0:2]} have disconnected.")
            break
        else:
            print("----------------------")
            print(f"{client[0:2]}:\n{message}")
            print("----------------------")
    lock.acquire()
    clients.remove(client)
    lock.release()
    connection.close()  # close the connection


port = input("Input the port you want to bind.\n")
try:
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    host = ('', int(port))  # Bind to all address
```

```
    s.bind(host)
    s.listen(10)  # 10 is the max number of connections
except socket.error as msg:
    print(msg)  # print(f"ERROR:Timed Out")
    exit(1)
print("Server start!")-
print('Waiting connection...')============================9=

threading.Thread(target=broadcast, args=()).start()  # Start the broadcast thread
while True:
    (connection, address) = s.accept()  # Accept the connection
    threading.Thread(target=reception, args=()).start()  # Start the reception thread
```

`port`是端口号，服务端在该端口创建监听，**最大可连接10个**。监听时进行**异常捕获并打印**对应出错信息`socket.error`，例如端口号占用或者超时等，此时程序将中止。

`reception`是为服务端接受客户端信息所用的线程，每当一个客户端与服务端连接时，服务端都会创建一个`reception`线程，传入的参数为地址（包含端口）与socket对象，这样子服务端就可以**异步接受到每一个线程的消息并打印**。客户端连接或者断开时，在服务端也会进行对应的提示打印，并对连接列表`clients`进行修改。

`clients`是包含当前所有已连接的请求的列表，内部元素是`ip地址`、`端口号`、`socket对象`。在连接成功后会客户端会加入`clients`队列，在连接断开后对应客户端也会从队列中删除。

`broadcast`是服务端给所有客户端广播信息所用的线程。服务端会给`clients`列表里的每一个客户端发送消息，**实现双工通信**。

客户端源代码

```
import socket
import threading


def transfer():
    if way == "file":
        print("Now the way is file,input nothing to stop.")
        while True:
            path = input("Please input the file path:\n").rstrip()
            # enter twice to stop
            if path == "":
                if input("Enter again to close") == "":
                    break
                else:
                    continue
            # send file
            file = open(path, encoding="UTF-8")
            message = file.readlines()
            file.close()
            for line in message:  # for each line
                # if the line is empty, ignore it
                if line == "":
                    continue
                a = 0
                while len(line) - a >= 8192:
```

```
                    s.sendall(line[a:a + 8192].encode())
                    a += 8192
                s.sendall(line[a:].encode())
            else:
                continue
    elif way == "keyboard":
        print("Now the way is standard input")
        while True:
            message = input("Please input the message:\n").rstrip()
            # enter twice to stop
            if message == "":
                if input("Enter again to close") == "":
                    break
                else:
                    continue
            # send message
            a = 0
            while len(message) - a >= 8192:
                s.sendall(message[a:a + 8192].encode())
                a += 8192
            s.sendall(message[a:].encode())
    else:
        print("ERROR:Wrong way!")
        exit(1)
    s.close()


def receive():
    while True:
        try:
            s.settimeout(1000)
            message = s.recv(8192).decode()
            print("----------------------")
            print(f"Broadcast:\n{message}")
            print("----------------------")
        except:
            break


way = input("The way you want to input is file or keyboard?\n")
address = input("The address you want to connect is?\n")
port = input("The port you want to connect is?\n")
host = (address, int(port))
try:
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.settimeout(3)
    s.connect(host)  # connect to the server
except socket.error as msg:
    print(msg)  # print(f"ERROR:Timed Out")
    exit()
print("Connected successfully.")
threading.Thread(target=transfer, args=()).start()
threading.Thread(target=receive, args=()).start()
```

`way`是**输入模式**，可以为`keyboard`或者是`file`。

`address`和`port`分别是待连接服务端的IP地址与端口号，之后就连接服务端，进行**异常捕获并打印**对应出错信息`socket.error`，例如连接超时，此时该客户端程序将中止，不影响服务端运行。客户端与服务端之间可以为**localhost**，或者是**在同一局域网内的不同机器。**

连接成功后，客户端会创建两个线程，`transfer`线程负责给服务端发送信息，`receive`线程负责**接收服务端的信息并打印。**

`transfer`线程根据`way`有两种方式发送信息，从标准输入(keyboard)读入直接输入即可，从文件中读入(file)需要输入文件的绝对路径，两者都是使用一次回车键确认发送，**两次回车键结束程序。**由于Socket的传输每次最大8192Bytes，于是将字符串总是分为8192Bytes的段进行依次的传输，**支持不少于20kb的长文本消息**发送。

实验结果总结

- 在测试过程中程序容错性好，对于各种连接异常都作**异常处理**，例如某一客户端异常不影响服务端与其他客户端，例如异常后都会打印出错信息并退出程序。测试较长时间都**无闪退情况。**
- 已实现Socket编程中所有评分标准，包括双工通信。
- 对于socket通信流程有更深入的理解。