Федеральное государственное автономное образовательное учреждение высшего образования**«Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»**

Факультет Институт международного развития и партнерства

Кафедра иностранных языков

Отчет

по дисциплине:

Web_программирование

Лабораторная работа No1

работа с сокетами

Автор: *Хоу Дан*

Группа D34102

Преподаватель:

_____

Санкт-Петербург

2021 г.

Связь с сервером при IPV4

Server:

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.bind(('172.20.10.13', 9090))
sock.listen(1)
conn, addr = sock.accept()
```

Client:

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('172.20.10.13', 9090))#'localhost'
```

Проверить, что успешно ли ссылка выполнена

Клиент отправляет тест на сервер, и сервер возвращает полученную информацию клиенту после ее получения. Если передача и прием совпадают, связь успешна, и, если они отличаются, связь отключена.

Server:

```
test = conn.recv(1024)
conn.send(test)
```

Client:

```
sock.send("Test".encode('utf-8'))
test = sock.recv(1024)
if test.decode('utf-8') == "Test":
    print('链接成功')
    addr = str(sock.recv(1024).decode('utf-8'))
    print(addr)
    # **************链接验证******************
    main(sock)
else:
    print(test.decode('utf-8'))
    sock.close()
```

Функция main

Server:

```
def main(conn,addr):
    while True:
        conn.send("***功能选取模式***\nendserver结束链接，math进入数学模式，autorobot自动俩天机器人，"
                  "Http发送http请求，Chat进入聊天群，UNKWON未知指令".encode('utf-8'))
        r = conn.recv(1024)
        r = r.decode('utf-8')
        if r == "endserver":
            conn.send(r.encode('utf-8'))
            endconn(conn,addr)
            break
        elif r == "math":
            conn.send(r.encode('utf-8'))
            math(conn,addr)
        elif r == "autorobot":
            conn.send(r.encode('utf-8'))
            autorobot(conn,addr)
        elif r == "Http":
            conn.send(r.encode('utf-8'))
            Http(conn,addr)
        elif r == "Chat":
            conn.send(r.encode('utf-8'))
            Chat(conn,addr)
        else:
            conn.send('UNKWON'.encode('utf-8'))
```

Client:

```
def main(sock):
    while True:
        mod = sock.recv(1024)
        print(mod.decode('utf-8'))
        m = input('cliemt:')
        sock.send(m.encode('utf-8'))
        re = sock.recv(1024)
        re = re.decode('utf-8')
        print('------------------model-------------->', re)
        # *************模式的选取与确认**************
        if re == "endserver":
            r = sock.recv(1024)
            print(r.decode('utf-8'))
            sock.close()
            break
        elif re == "autorobot":
            print("进入autorobot模式")
            autorobot(sock)
        elif re == "math":
            print("进入数学模式")
            math(sock)
        elif re == "Http":
            print("进入Http模式")
            Http(sock)
        elif re == "Chat":
            print("进入Chat模式")
            Chat(sock)
```

После успешной проверки сервер отправит введение в службу.

Сервер возвращает команды и вводит различные сервисные функции

Функция endserver

Server:

```
if r == "endserver":
    conn.send(r.encode('utf-8'))
    endconn(conn,addr)
    break
```

```
def endconn(conn,addr):
    conn.send("断开链接".encode('utf-8'))
    conn_list.remove(conn)
    addr_list.remove(addr)
    conn.close()
```

Client:

```
if re == "endserver":
    r = sock.recv(1024)
    print(r.decode('utf-8'))
    sock.close()
    break
```

Сервер получает команду endsever и вызывает функцию endserver. Remove данные клиента и закончить связи с клиентом.

Функция math

Server:

```python
def math(conn,addr):
    addr = addr
    conn.send("说明：math进入计算模式，a进入勾股定理计算，b求解二次方程，"
              "c计算梯形面积，d计算平行四边形面积，end退出该模式".encode('utf-8'))
    while True:
        text = conn.recv(1024)
        text = text.decode('utf-8')
        if text == "end":
            break
        elif text == "math":
            conn.send("***maht中的math计算，完成后自动返回math模式中***".encode('utf-8'))

            text = conn.recv(1024)
            text = text.decode('utf-8')
            r = r'\d+'
            r2 = r'[/+-/*/]'
            ints = re.findall(r, text)
            symbols = re.findall(r2, text)

            count = float(ints.pop(0))

            for i in [0] * len(ints):
                pop = float(ints.pop(i))
                symbol = symbols.pop(i)
                if symbol == '+':
                    count += pop
                elif symbol == '-':
                    count -= pop
                elif symbol == '*':
                    count *= pop
                elif symbol == '/':
                    count /= pop
            conn.send(str(count).encode('utf-8'))
```

```python
elif text == "a":
    # 打印说明
    conn.send("***输入abc三边，以a+b或c-a或c-b形式输入，返回计算结果***".encode('utf-8'))
    text = conn.recv(1024)
    text = text.decode('utf-8')
    # **************************
    r1 = r'\d+'
    r2 = r'[\+-]'
    ints = re.findall(r1, text)
    symbols = re.findall(r2, text)
    result = 0.0
    count = float(ints.pop(0)) ** 2
    for i in [0] * len(ints):
        pop = float(ints.pop(i)) ** 2
        symbol = symbols.pop(i)
        print(symbols)
        if symbol == '+':
            result = (count + pop) ** (1 / 2)
        elif symbol == '-':
            if count > pop:
                result = (count - pop) ** (1 / 2)
            else:
                result = (pop - count) ** (1 / 2)
    conn.send(str(result).encode('utf-8'))
```

```
elif text == "b":
    # 打印说明
    conn.send("**输入二次方程，或者以a=,b=,c=的形式输入**".encode('utf-8'))
    text = conn.recv(1024)
    text = text.decode('utf-8')
    # ************************
    r_a = r'(?<=a=)\d+\.?\d*|\d+\.?\d*(?=[Xx].2)'
    r_b = r'(?<=b=)\d+\.?\d*|\d+\.?\d*(?=[Xx][/+-])'
    r_c = r'(?<=c=)\d+\.?\d*|(?<=\+|-)\d+\.?\d*$'
    a = float(re.findall(r_a, text)[0]) if re.findall(r_a, text) else 1
    b = float(re.findall(r_b, text)[0]) if re.findall(r_b, text) else 1
    c = float(re.findall(r_c, text)[0]) if re.findall(r_c, text) else 0
    D = b ** 2 - 4 * a * c
    if D > 0:
        x1 = (-b + D ** (1 / 2)) / 2 * a
        x2 = (-b * D ** (1 / 2)) / 2 * a
        x = "x1:" + str(x1) + ", x2:" + str(x2)
        conn.send(x.encode('utf-8'))
    elif D == 0:
        x = (-b * D ** (1 / 2)) / 2 * a
        conn.send(str(x).encode('utf-8'))
    else:
        conn.send("无解D<0".encode('utf-8'))
elif text == "c":
    # 打印说明
    conn.send("**根据提示输入梯形参数**".encode('utf-8'))
    # ************************
    conn.send("上底:".encode('utf-8'))
    s = conn.recv(1024).decode('utf-8')

    conn.send("下底:".encode('utf-8'))
    x = conn.recv(1024).decode('utf-8')

    conn.send("高:".encode('utf-8'))
    h = conn.recv(1024).decode('utf-8')

    x = float(x)
    s = float(s)
    h = float(h)
    area = (s + x) * h / 2

    conn.send(str(area).encode('utf-8'))
```

```
elif text == "d":
    # 打印说明
    conn.send("**根据提示输入平行四边形参数**".encode('utf-8'))
    # ************************
    conn.send("底边长:".encode('utf-8'))
    x = conn.recv(1024).decode('utf-8')

    conn.send("高:".encode('utf-8'))
    h = conn.recv(1024).decode('utf-8')

    x = float(x)
    h = float(h)
    area = x * h
    conn.send(str(area).encode('utf-8'))
else:
    conn.send("UNKWON".encode('utf-8'))
```

Client:

```
def math(sock):
    # 接收说明
    illustrate = sock.recv(1024)
    print(illustrate.decode('utf-8'))
    while True:
        m = input('cloemt:')
        sock.send(m.encode('utf-8'))
        print(m == "math")
        if m == "end":
            print('**结束math数学模式**')
            break
        elif m == "math":
            ill = sock.recv(1024)
            print(ill.decode('utf-8'))

            m = input("client:")
            sock.send(m.encode('utf-8'))

            r = sock.recv(1024)
            r = r.decode('utf-8')
            print('server:', r)
```

```
elif m == "a":
    # 接受说明
    ill = sock.recv(1024)
    print(ill.decode('utf-8'))
    # ********
    m = input('cloemt:')
    m = m.encode('utf-8')
    sock.send(m)
    # ********
    result = sock.recv(1024).decode('utf-8')
    print("返回结果:", result)
elif m == "b":
    # 接受说明
    ill = sock.recv(1024)
    print(ill.decode('utf-8'))
    # ********
    m = input('cloemt:')
    m = m.encode('utf-8')
    sock.send(m)
    # ********
```

```
elif m == "c":
    # 接受说明
    ill = sock.recv(1024)
    print(ill.decode('utf-8'))
    # ********
    r1 = sock.recv(1024).decode('utf-8')
    m = input(r1)
    sock.send(m.encode('utf-8'))

    r2 = sock.recv(1024).decode('utf-8')
    m = input(r2)
    sock.send(m.encode('utf-8'))

    r3 = sock.recv(1024).decode('utf-8')
    m = input(r3)
    sock.send(m.encode('utf-8'))
    # ********
    result = sock.recv(1024).decode('utf-8')
    print("梯形面积:", result)
```

```
elif m == "d":
    # 接受说明
    ill = sock.recv(1024)
    print(ill.decode('utf-8'))
    # ********
    r1 = sock.recv(1024).decode('utf-8')
    m = input(r1)
    sock.send(m.encode('utf-8'))

    r2 = sock.recv(1024).decode('utf-8')
    m = input(r2)
    sock.send(m.encode('utf-8'))
    # ********
    result = sock.recv(1024).decode('utf-8')
    print("平行四边形面积:", result)
else:
    r = sock.recv(1024)
    print(r.decode('utf-8'))
```

Входить в различные функции математического расчета в соответствии с параметрами, переданными пользователем. Введить параметры в соответствии с возвращенным описанием.

Функция Http

Server:

```
Http(conn, addr):
addr = addr
conn.send("说明：按照要求输入请求段，hist查看历史，end退出该模式".encode('utf-8'))
while True:

    method = (conn.recv(1024).decode('utf-8')).upper()
    if method == "END":
        print("tiaochu")
        break
    elif method == "HIST":
        with open('Historical_search.csv') as f:
            text = read(f, 5)
        conn.send(text.encode('utf-8'))
        continue
    else:

        host = conn.recv(1024).decode('utf-8')

        Port = int(conn.recv(1024).decode('utf-8'))

        url = conn.recv(1024).decode('utf-8')

        data = conn.recv(1024).decode('utf-8')

        Content_Type = conn.recv(1024).decode('utf-8')

        print(repr(method),repr(host),repr(Port),repr(url),repr(data),repr(Content_Type))
        dataset_pd = pd.read_csv('Historical_search.csv', sep=",")
        dnew_pd = pd.DataFrame([[method,host,Port,url,repr(data),Content_Type,pd.datetime.now()]],columns=['method','host','Port','url'
            ,'data','Content_Type','data_time'])
        dataset_pd = dataset_pd.append(dnew_pd,ignore_index=True)
        dataset_pd =dataset_pd[['method','host','Port','url','data','Content_Type','data_time']]
```

```
        print(repr(method),repr(host),repr(Port),repr(url),repr(data),repr(Content_Type))
        dataset_pd = pd.read_csv('Historical_search.csv', sep=",")
        dnew_pd = pd.DataFrame([[method,host,Port,url,repr(data),Content_Type,pd.datetime.now()]],columns=['method','host','Port','url'
            ,'data','Content_Type','data_time'])
        dataset_pd = dataset_pd.append(dnew_pd,ignore_index=True)
        dataset_pd =dataset_pd[['method','host','Port','url','data','Content_Type','data_time']]
        dataset_pd.to_csv('./Historical_search.csv')

        if method == "HEAD":
            with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
                s.connect((host, Port))
                print(f"与{host}:{Port}建立连接")
                ask = method + f" {url}" + " HTTP/1.1\r\n" + "Host: " + host + "\r\nAccept: " + Content_Type + "\r\nConnection: " + data + "\r\n\r\n"
                s.sendall(ask.encode('utf-8'))
                respones = s.recv(1024)
                conn.send(respones)

        elif method == "GET":
            with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:

                s.connect((host, Port))
                ask = method + f" {url}" + " HTTP/1.1\r\n" + "Host: " + host + "\r\nAccept: " + Content_Type + "\r\nConnection: " + data + "\r\n\r\n"
                s.sendall(ask.encode('utf-8'))

                data = s.recv(1024 * 1024).decode('utf-8')
                respones = ""
                while data:
                    respones = respones + data
                    data = s.recv(1024*1024).decode('utf-8')
                print(respones)
                conn.sendall(respones.encode('utf-8'))
```

Client:

```
def Http(sock):
    illustrate = sock.recv(1024)
    print(illustrate.decode('utf-8'))
    while True:

        m = input("请求方法:")
        sock.send(m.encode('utf-8'))
        if m == "end":
            print('**结束Http模式**')
            break  # 退出检测
        elif m == "hist" or m == "HIST":
            print("输出历史记录:")
            text = sock.recv(1024).decode('utf-8')
            print(text)
            continue
        else:

            m = input("Host:")
            sock.send(m.encode('utf-8'))

            m = input("Porst:")
            if not m:
                m = "80"
            sock.send(m.encode('utf-8'))

            m = input("Url:")
            if not m:
                m = "/"
            sock.send(m.encode('utf-8'))

            m = input("Connection:")
            if not m:
                m = "\r\n"
            sock.send(m.encode('utf-8'))

            m = input("Content-Type:")
            if not m:
                m = "text/html"
            sock.send(m.encode('utf-8'))

            response = sock.recv(1024 * 1024).decode('utf-8')
            print("返回数据: \n", response)
```

Сервер отправит запрос на основе метода запроса и заголовка запроса, переданного пользователем, и вернет возвращенный html-файл клиенту.

Функция Chat

Server:

```python
def broadcast():
    while True:
        # print(monitor_text)
        time.sleep(0.3)
        for x in range(len(monitor_text)):
            info = monitor_text.pop(0)
            if info != False:
                # print(conn_list_chat)
                if info[2] == "《---进入聊天---》\n":
                    # 传送成员列表
                    print("in", addr_list_chat)
                    for client_addr in addr_list_chat:#必须把整张表发给客户端，因为新进入的客户端没有全部数据
                        addr_client = "\\\\addr-in:" + str(client_addr)  # 通过in和out告诉客户端是要加还是减chatclientlist
                        for conn in conn_list_chat:
                            conn.send(addr_client.encode('utf-8'))
                            time.sleep(0.01)
                elif info[2] == "《----退出聊天---》\n":#可以仅发退出者的信息，应为所有客户端进入时已经获得了整表
                    # print("out", addr_list_chat)
                    client_addr = info[1]
                    addr_client = "\\\\addr-out:" + str(client_addr)
                    for conn in conn_list_chat:
                        conn.send(addr_client.encode('utf-8'))
                        time.sleep(0.01)
                for i in range(len(conn_list_chat)):
                    re = f"{info[1]}:"+info[2]
                    # print(re)
                    # time.sleep(0.001)#两个发送相隔时间过短会被一个recv接收
                    conn_list_chat[i].send(re.encode('utf-8'))
```

```python
def Chat(conn,addr):
    addr = addr
    conn.send("说明：进入聊天群，endchat结束模式".encode('utf-8'))
    time.sleep(0.2)
    conn_list_chat.append(conn)
    addr_list_chat.append(addr)
    #将用户进入信息压入广播列表内
    info = [conn, addr, "《---进入聊天---》\n"]
    monitor_text.append(info)
    thread_broadcast = threading.Thread(target=broadcast)#开启广播线程
    thread_broadcast.start()
    while True:
        r = conn.recv(1024)
        r = r.decode('utf-8')
        # print(r == "endchat\n")
        if r == "endchat\n":
            info =[conn,addr,"《----退出聊天---》\n"]
            conn.send("breack".encode('utf-8'))#杀掉客户端监听线程
            monitor_text.append(info)
            conn_list_chat.remove(conn)
            addr_list_chat.remove(addr)
            time.sleep(0.5)
            break
        else:
            list_info = []
            list_info.append(conn)
            list_info.append(addr)
            list_info.append(r)
            monitor_text.append(list_info)
            # time.sleep(0.1)
```

Сервер использует функцию широковещательной передачи для непрерывной трансляции собранного пользовательского ввода через список conn_list_chat.

Если пользователь входит в группу чата, функция чата введет информацию о пользователе в conn_list_chat и добавит "**Пользователь《---входит в группу чата---》" в контент, который будет транслироваться.

Cliet:

```
def TK(sock):
    global listLianxi,txtMsg,txtMsgList
    def Clear_History():...
    def sendMsg(sock):...
    def cancelMsg():...
    def sendMsgEvent(event):...
    def send_close():...
    def close():...
    # <editor-fold desc="TK块">
    t = Tk()# 创建窗口
    t.title('Chat聊天窗口')  # 窗口名称
    t.resizable(0, 0)  # 禁止调整窗口大小
    #########创建frame容器******###
    frmA1 = Frame(width=180, height=300)
    frmB1 = Frame(width=350, height=300)
    frmB2 = Frame(width=350, height=80)
    frmB3 = Frame(width=350, height=25)
    #########创建控件******###
    # 1.Text控件

    txtMsgList = Text(frmB1,width=45)
    txtMsg = Text(frmB2,width=45);
    txtMsg.bind("<KeyPress-Return>", sendMsgEvent)  # 事件绑定，定义快捷键

    btnSend = Button(frmB3, text='发送', width=8, command=lambda: sendMsg(sock))
    btnCancel = Button(frmB3, text='取消', width=8, command=cancelMsg)
    btnCance2 = Button(frmB3, text="关闭", width=8, command=send_close)
    btnCance3 = Button(frmB3, text="清除记录", width=8, command=Clear_History)
    scroLianxi = Scrollbar(frmA1, width=22)#聊天群成员
    listLianxi = Listbox(frmA1, width=24, height=20,
                         yscrollcommand=scroLianxi.set)  # 连接listbox 到 vertical scrollbar
    scroLianxi.config(command=listLianxi.yview)  # scrollbar滚动时listbox同时滚动
```

```
#########窗口布局*******###
frmA1.grid(row=0, column=0)
frmB1.grid(row=0, column=1)
frmB2.grid(row=2, column=1)
frmB3.grid(row=3, column=1)
#########窗口布局*******###
frmA1.grid_propagate(0)
frmB1.grid_propagate(0)
frmB2.grid_propagate(0)
frmB3.grid_propagate(0)
#########控件布局*******###
btnSend.grid(row=0, column=0)
btnCancel.grid(row=0, column=1)
btnCance2.grid(row=0, column=2)
btnCance3.grid(row=0, column=3)

txtMsgList.grid()
txtMsg.grid()

scroLianxi.grid(row=0, column=1, ipady=120)
listLianxi.grid(row=0, column=0)
# </editor-fold>
return t
```

```
def Chat(sock):
    global chat_client_list
    t = TK(sock)
    illustrate = sock.recv(1024)
    print(illustrate.decode('utf-8'))
    chat_client_list = []
    M = threading.Thread(target=monitor)
    M.start()
    t.mainloop()
```

```
def add_chat_client_info():
    listLianxi.delete(0,END)
    # print(chat_client_list)
    for i in range(len(chat_client_list)):
        client_NO = f'Client{i}(ONLINE):'
        client_If = str(chat_client_list[i])
        listLianxi.insert(END, client_NO)
        listLianxi.insert(END, client_If)
```

Пользователь создает окно чата локально с помощью функции TK.

Окно чата состоит из списка участников группы чата слева и отображения истории чата и поля ввода чата справа. Функция add_chat_cient_info будет выводить онлайн-пользователей в режиме реального времени в левом окне комнаты чата на основе списка chat_cient_list.

```
def monitor():
    while 1:
        text = sock.recv(1024)
        text = text.decode('utf-8')
        # print(text == "breack")
        # print(text)
        if text == "breack":
            break
        elif text[:10] == "\\\\addr-in:":
            addr_client = text[10:]
            if addr_client not in chat_client_list:
                chat_client_list.append(addr_client)
                add_chat_client_info()
            continue
        elif text[:11] == "\\\\addr-out:":
            addr_client = text[11:]
            if addr_client in chat_client_list:
                chat_client_list.remove(addr_client)
                add_chat_client_info()
            continue
        txtMsgList.insert(END, text)
```

Функция монитора будет отслеживать данные, отправляемые сервером в режиме реального времени. если это информация о присоединении или выходе участников чата, функция добавит или удалит участников, присоединяющихся или выходящих из списка chat_cient_list. Если пользователь выйдет из чата, функция прослушает команду breack.Если это сообщение чата, распечатайте его в чате.