گزارش تمرین اول – مهندسی اینترنت

رامتین احسانی 97521018

فایل های حاوی کد این برنامه نوشته شده: Client1.py – Client2.py

کد های نوشته شده برای این دو فایل کاملا شبیه هم هستند با این تفاوت که برای راحت کردن فرآیند اجرای کد ها، دو کلاینت از هم جدا شده اند تا شماره port های منحصر به فردی به هرکدام تعلق بگیرد.

منطق برنامه به این صورت است که هر دو کلاینت در ابتدا بر روی دو سوکت شروع به کار میکنند. یک سوکت اختصاصی برای ارسال پیام به طرف مقابل در thread اصلی. یک سوکت برای گوش دادن به پیام های ارسالی در thread و پورت جداگانه. سه لیست در اختیار هر کاربر است: لیست پیام های ارسالی و لیست پیام های دریافتی و لیست ack های دریافتی. هر بار که پیامی برای طرف مقابل ارسال میکنند، لیست پیام های ارسالی آن ها افزایش میابد و هر بار پیامی دریافت میکنند لیست پیام های دریافتی افزایش میابد.

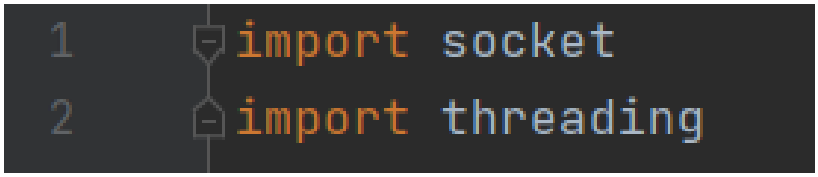
با ارسال پیام، علاوه بر پیام اصلی، عددی هم همراه با آن ارسال میشود که نشان دهنده تعداد پیام هایی است که با **موفقیت** از طرف مقابل دریافت کرده است. طرف مقابل(کاربر B) با دریافت پیام از کاربر A، عدد همراه با پیام را چک میکند. در صورتی که **تعداد** پیام های ارسالی که طرف مقابل(کاربر A) دریافت کرده است با تعداد پیام های نوشته شده توسط کاربر B برابر نباشد، کاربر B تمام پیام های نوشته شده را در سوکت و thread جدید برای کاربر A میفرستد.

همچنین، با دریافت هر پیام، آن کاربر باید برای طرف مقابل خود یک ack هم بفرستد. این ack ها برای هر کاربر در لیست های متناظرشون ذخیره میشود.

با ارسال هر پیام با استفاده از timer یک زمان برای 10 ثانیه بعد ست میشود که بعد از گذشت آن زمان **اگر** تعداد ack های دریافتی با تعداد پیام های ارسالی برابر نباشد پیام های خود را دوباره برای طرف مقابل میفرستد.

پیام های دوباره ارسال شده با تگ "resend" فرستاده میشوند تا کاربر مقابل متوجه بشود.

برنامه با استفاده از زبان python و با استفاده از پکیج های threading و socket نوشته شده است.



در قسمت اول کد، اطلاعات کلاینت مورد نظر مانند شماره پورت های لازم در ادامه برنامه، ip، bufferSize و لیست هایی برای ذخیره پیام های ارسالی و دریافتی و ack ها تعریف شده اند. همچنین اطلاعات طرف مقابل که شامل ip و پورت **شنونده** طرف مقابل است هم در destinationAddressPort نوشته است.

```
localIP = '127.0.0.1'
sourcePort, listenPort = 65432, 65438
destinationAddressPort = ("127.0.0.1", 20001)

bufferSize = 1024
messages_received, messages_sent, acks_received = [], [], []

```

در قسمت بعدی، سوکت برای کلاینت تعریف کردم که از این سوکت برای فرستادن پیام به طرف مقابل استفاده میشود. سوکت تعریف شده UPD است. (SOCK_DGRAM)

```
# Create a datagram socket (UDP)
UDPSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
UDPSocket.bind((localIP, sourcePort))

print("Client 1 up and listening")


```

برای قسمت بعدی، تابع resend را نوشتم که در آن سوکت جدیدی تعریف شده است و در صورت فراخوانی **تمام** پیام های **نوشته** شده به طرف مقابل را دوباره ارسال میکند.

```
# Resend messages when needed
def resend():
    sock = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
    for message in messages_sent:
        resend_message = str.encode(message + ' -resend')
        sock.sendto(resend_message, destinationAddressPort)


```

تابع resend_timeout هم مانند resend است با این تفاوت که توسط هر پیام ارسال شده بعد از گذشت timeout فراخوانی شده و با چک کردن تعداد پیام های ارسالی با تعداد ack های دریافتی، اگر برابر **نباشند** پیام ها را دوباره ارسال میکند.

```
# Resend messages in case of timeout
def resend_timeout():
    sock = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
    if len(acks_received) != len(messages_sent):
        for message in messages_sent:
            resend_message = str.encode(message + ' -resend')
            sock.sendto(resend_message, destinationAddressPort)

```

بعد از آن، تابع listen را نوشتم که بر روی listenPort همیشه در حال گوش دادن است. در همین تابع است که با دریافت پیام، عدد ارسالی هم چک میشود تا درصورت مغایرت، تابع resend را در thread جداگانه ای فراخوانی کند. پیام های دریافتی در صورتی که resend نباشند در لیست پیام های دریافتی ذخیره میشوند و نمایش هم داده میشوند.

در صورتی که ack باشند هم در لیست ack ها آن را اضافه کرده و دوباره به حالت گوش دادن می رود.

در صورتی که تگ "resend" داشته باشد کاری با آن پیام ندارد و فقط آن را نمایش میدهد.

```python
# Listen for incoming datagrams
def listen():
    sock = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
    sock.bind(('127.0.0.1', listenPort))

    while True:
        data = sock.recv(1024)

        if data.decode() == "ack":
            acks_received.append(data.decode())
            continue
        print('\rClient2: {}\n> '.format(data.decode()), end='')
        data_split = data.decode().split('-')
        if data_split[1] == "resend":
            continue
        messages_received.append(data.decode())
        UDPSocket.sendto(str.encode("ack"), destinationAddressPort)
        if int(data_split[1]) != len(messages_sent):
            resend_thread = threading.Thread(target=resend)
            resend_thread.start()

```

کار گوش دادن هم همانگونه که اشاره شد در thread جدایی انجام میگیرد.

```python
# New Thread for listening
listener = threading.Thread(target=listen, daemon=True)
listener.start()

```

در thread اصلی هم با استفاده از سوکتی که در اول تعریف کردم به پورتی که طرف مقابل در حال **گوش دادن** است پیام میفرستیم. پیام ها هم علاوه بر متن اصلی، حاوی تعداد پیام های دریافتی که موفق به کاربر رسیده اند نیز می باشد. با ارسال پیام تایمر هم شروع به کار میکند.

```
61      # Send Messages to other client using created UDP socket
62      while True:
63          msg = input('> ')
64          messages_sent.append(msg)
65
66          bytesToSend = str.encode(msg + ' -' + str(len(messages_received)))
67
68          UDPSocket.sendto(bytesToSend, destinationAddressPort)
69          timer = threading.Timer(10, resend_timeout)
70          timer.start()
71
```

اجرای برنامه: بعد از اجرا هر دو کاربر آماده ارسال و دریافت پیام هستند.

```
Client 1 up and listening
>
```

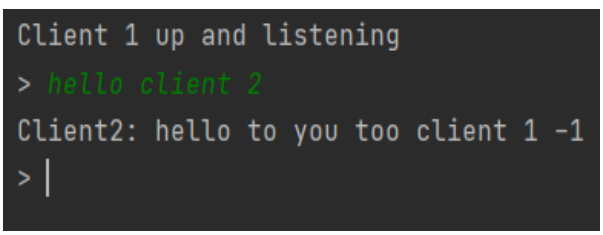
```
Client 2 up and listening
>
```

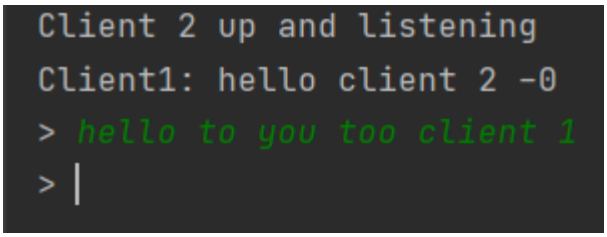
ارسال پیام:

```
Client 1 up and listening
> hello client 2
>
```

```
Client 2 up and listening
Client1: hello client 2 -0
>
```

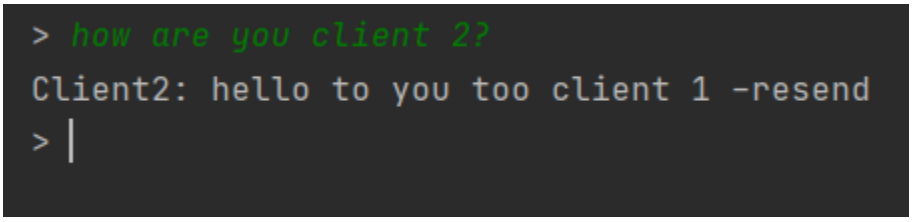
کلاینت 1 به 2 پیام ارسال کرده و برای کلاینت 2 به نمایش درآمده است. 0 یعنی هنوز از تو پیامی نگرفتم.





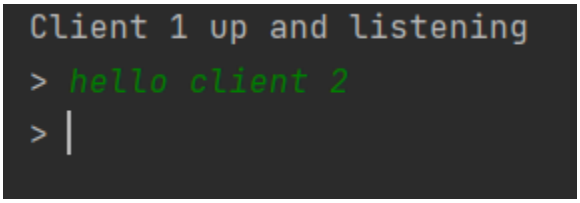
هر دو یک پیام ارسال کرده و یک پیام دریافت کرده اند.

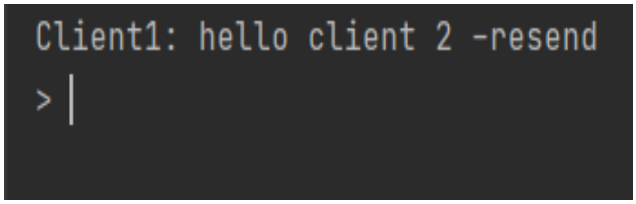
برای تست قابلیت resend شرط چک کردن تعداد پیام ها را تغییر میدهم تا قطعا دوباره پیام ها را ارسال کند تا مطمئن بشویم که برنامه درست کار میکند:



با ارسال این پیام از 1 به 2، کلاینت 2 با توجه به عددی که دریافت میکند متوجه میشود که کلاینت 1 پیام قبلی را دریافت نکرده است. پس دوباره پیام ها را برای 1 میفرستد (”resend“).

اگر timeout هم اتفاق بیفتد و تعداد ack های دریافتی با پیام های ارسالی برابر نباشد، کلاینت 1 دوباره پیام را برای 2 میفرستد:





پایان گزارش