

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

کد های نوشته شده برای این دو فایل کاملاً 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 هر کدام 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. دو لیست در اختیار هر کاربر است: لیست پیام های ارسالی و لیست پیام 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.

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.

```
1 import socket
2 import threading
```

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 تعریف شده اند. همچنین اطلاعات طرف مقابل که شامل ip و پورت شنونده طرف مقابل است هم در destinationAddressPort نوشته است.

```

4     localIP = '127.0.0.1'
5     sourcePort, listenPort = 65432, 65438
6     destinationAddressPort = ("127.0.0.1", 20001)
7
8     bufferSize = 1024
9     messages_received, messages_sent = [], []
10

```

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

```

11     # Create a datagram socket (UDP)
12     UDPSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
13     UDPSocket.bind((localIP, sourcePort))
14
15     print("Client 1 up and listening")
16
17

```

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.

```

18     # Resend messages when needed
19     def resend():
20         sock = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
21         for message in messages_sent:
22             resend_message = str.encode(message + ' -resend')
23             sock.sendto(resend_message, destinationAddressPort)
24
25

```

بعد از آن، تابع `listen` را نوشتیم که بر روی `listenPort` همیشه در حال گوش دادن است. در همین تابع است که با دریافت پیام، عدد ارسالی هم چک میشود تا در صورت 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 تگ `"resend"` داشته باشد کاری با آن پیام ندارد و فقط آن را نمایش میدهد.

```
26 # Listen for incoming datagrams
27 def listen():
28     sock = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
29     sock.bind(('127.0.0.1', listenPort))
30
31     while True:
32         data = sock.recv(1024)
33         print('\rClient2: {} \n> '.format(data.decode()), end='')
34
35         data_split = data.decode().split('-')
36         if data_split[1] == "resend":
37             continue
38         messages_received.append(data.decode())
39         if int(data_split[1]) != len(messages_sent):
40             resend_thread = threading.Thread(target=resend)
41             resend_thread.start()
42
```

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

```
44 # New Thread for listening
45 listener = threading.Thread(target=listen, daemon=True)
46 listener.start()
47
```

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.

```
48 # Send Messages to other client using created UDP socket
49 while True:
50     msg = input('> ')
51     messages_sent.append(msg)
52
53     bytesToSend = str.encode(msg + ' -' + str(len(messages_received)))
54
55     UDPsocket.sendto(bytesToSend, destinationAddressPort)
56
```

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.

```
Client 1 up and listening
> hello client 2
Client2: hello to you too client 1 -1
> |
```

```
Client 2 up and listening
Client1: hello client 2 -0
> hello to you too client 1
> |
```

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:

```
> how are you client 2?  
Client2: hello to you too client 1 -resend  
> |
```

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").

پایان گزارش