Amirkabir University of
Technology

پروژه نهایی درس شبکه

نام استاد : دکتر صادقیان

نام دانشجو: محمد حسین سلیمانی

شماره دانشجویی : ۹۷۲۳۱۴۷

مقدمه

در این پروژه، هدف ما طراحی یک شبکه است که با استفاده از سرورهای STUN قادر باشد اطلاعات کاربران را ذخیره کرده و سپس با اتصال نقطه به نقطه (P۲P) کاربران را به یکدیگر متصل کند، به گونه‌ای که آن‌ها بتوانند با یکدیگر چت کنند و علاوه بر این، قابلیت ارسال عکس را نیز داشته باشند.

STUN (Session Traversal Utilities for NAT) سرورها را برای اتصال به کاربران در شبکه‌های NAT استفاده می‌کنیم. NAT یک تکنولوژی است که برای مدیریت آدرس‌های IP در شبکه‌ها استفاده می‌شود و می‌تواند باعث محدودیت‌ها در ارتباطات مستقیم بین کاربران شود. با استفاده از سرورهای STUN، می‌توانیم اطلاعات لازم برای اتصال کاربران را در شبکه‌های NAT به دست آوریم.

هدف اصلی این پروژه ایجاد یک سیستم ارتباطی دو نفره است که بدون نیاز به سرور مرکزی عمل کند. در این سیستم، هر کاربر می‌تواند به عنوان یک کلاینت عمل کند و از طریق STUN سرورها اطلاعات مربوط به آدرس IP و پورت خود را دریافت کند. سپس با استفاده از این اطلاعات، کاربران می‌توانند به صورت مستقیم با یکدیگر ارتباط برقرار کنند.

علاوه بر این، این سیستم قابلیت چت و ارسال عکس نیز دارد. کاربران متصل شده می‌توانند به صورت فوق العاده ساده و سریع پیام‌های متنی را با یکدیگر تبادل کنند و همچنین تصاویر را نیز به یکدیگر ارسال کنند.

با استفاده از این سیستم، کاربران می‌توانند به راحتی با یکدیگر در ارتباط باشند و بدون نیاز به سرور مرکزی از امکانات چت و ارسال عکس بهره‌برداری کنند. این پروژه می‌تواند در ایجاد ارتباطات نقطه به نقطه در برنامه‌ها و سرویس‌های مختلف از جمله برنامه‌های مبتنی بر وب، اپلیکیشن‌های موبایل و بازی‌های آنلاین مورد استفاده قرار گیرد.

سرور

در بخش اول کار نیاز بود یک سرور اصلی طراحی شود به شکلی که سه نقطه ی پایانی داشته باشد

۱ . ثبت نام در سیستم
۲ . گرفتن لیست کاربران
۳. گرفتن آی پی یک کاربر خاص

ثبت نام در سیستم

```python
@app.route('/send_peer_profile', methods=['POST'])
def send_peer_profile():
    data = request.get_json()
    username = data.get('username')
    address = data.get('address')
    print(username,address)
    if username and address:
        cache.set(username, address)
        return jsonify({'message': 'User profile saved successfully'}), 200
    else:
        return jsonify({'error': 'Invalid data'}), 400      soleimani101, 50 m
```

این قطعه کد یک مسیر (route) در یک وب سرویس فلسک ایجاد می‌کند. این مسیر با روش HTTP POST قابل دسترسی است و در آن متدهایی برای ذخیره و ارسال اطلاعات کاربر ایجاد شده است.

وقتی درخواستی به آدرس 'send_peer_profile/' ارسال می‌شود، این تابع به عنوان برنامه اصلی برای پاسخ به درخواست فراخوانی می‌شود.

در ابتدای تابع، داده‌های ارسال شده به صورت JSON از درخواست دریافت می‌شوند و در متغیر data قرار می‌گیرند.

سپس، مقادیر 'username' و 'address' از داده‌های دریافت شده خوانده می‌شوند. در خطوط بعدی، نام کاربری (username) و آدرس (address) دریافت شده روی خروجی چاپ می‌شوند (به منظور دیباگ کردن و بررسی صحت داده‌ها).

سپس، با بررسی شرط if، اطمینان حاصل می‌شود که هر دو مقدار نام کاربری و آدرس معتبر و موجود هستند. در این صورت، اطلاعات کاربر در یک حافظه نهان (cache) با استفاده از نام کاربری به عنوان کلید ذخیره می‌شوند. سپس یک پیام JSON به همراه کد وضعیت ۲۰۰ (با محتوای 'User profile saved successfully') برگشت داده می‌شود.

در غیر این صورت (یعنی اگر مقادیر نام کاربری و آدرس وجود نداشته باشند)، یک پیام JSON خطا به همراه کد وضعیت ۴۰۰ (با محتوای 'Invalid data') برگشت داده می‌شود.

گرفتن لیست کاربران

```python
@app.route('/get_all_peers', methods=['GET'])      soleimani101, 52 minutes ago • first commit
def get_all_peers():
    usernames = cache.keys()
    usernames_str = [username.decode() for username in usernames]  # Convert bytes to string
    return jsonify({'peers': usernames_str}), 200
```

این قطعه کد یک مسیر (route) را در یک وب سرویس فلسک تعریف می‌کند. این مسیر با روش HTTP GET قابل دسترسی است و برای دریافت لیستی از نام‌های کاربران موجود در سیستم ایجاد شده است.

وقتی درخواستی به آدرس 'get_all_peers/' ارسال می‌شود، این تابع به عنوان برنامه اصلی برای پاسخ به درخواست فراخوانی می‌شود.

در ابتدای تابع، نام‌های کاربران موجود در حافظه نهان (cache) استخراج می‌شوند. به منظور استخراج نام‌ها، از تابع keys() استفاده می‌شود که نام کلیدهای موجود در حافظه نهان را برمی‌گرداند.

سپس، با استفاده از حلقه for، نام کاربران به ترتیبی در لیست usernames_str قرار می‌گیرند. قبل از قرار دادن نام‌ها در لیست، آنها از بایت به رشته (string) تبدیل می‌شوند (با استفاده از تابع decode()).

در نهایت، لیست نام‌های کاربران به همراه کد وضعیت ۲۰۰ به عنوان یک پیام JSON برگشت داده می‌شود. در محتوای JSON، کلید 'peers' با مقدار لیست نام‌های کاربران قرار می‌گیرد.

گرفتن آی پی یک کاربر خاص

```python
@app.route('/get_peer_info/<username>', methods=['GET'])
def get_peer_info(username):
    print(username)
    address = cache.get(username)
    print(address)
    if address:      soleimani101, 53 minutes ago • first commit
        return ({'address': address.decode()}), 200
    else:
        return ({'error': 'Peer not found'}), 404
```

این قطعه کد یک مسیر (route) در یک وب سرویس فلسک ایجاد می‌کند. این مسیر با روش HTTP GET قابل دسترسی است و از طریق آن اطلاعات کاربران متصل به سرویس دریافت می‌شود.

وقتی درخواستی به آدرس '/get_peer_info/<username>' ارسال می‌شود، این تابع به عنوان برنامه اصلی برای پاسخ به درخواست فراخوانی می‌شود. در اینجا، یک پارامتر متغیر به نام 'username' در آدرس تعیین شده است که برای دریافت اطلاعات یک کاربر خاص استفاده می‌شود.

در ابتدای تابع، نام کاربری (username) را روی خروجی چاپ می‌کند (به منظور دیباگ کردن و بررسی صحت داده‌ها).

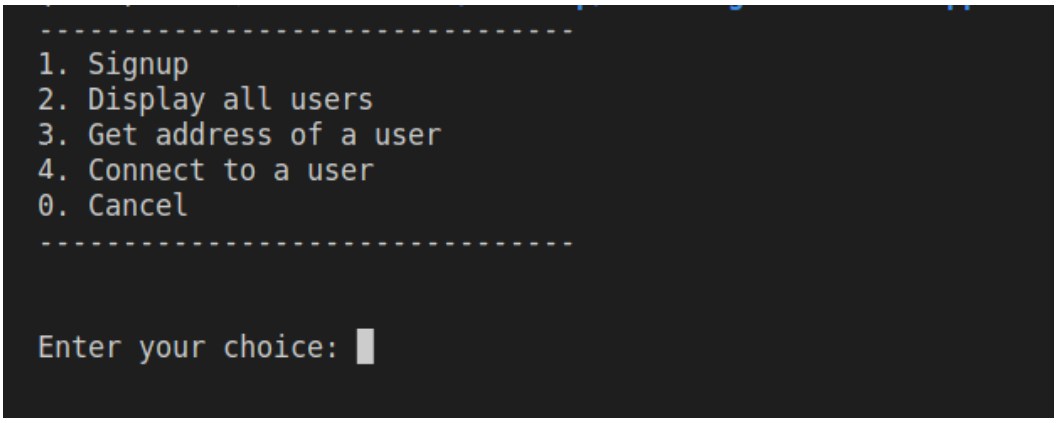
سپس، با استفاده از نام کاربری دریافت شده، آدرس مربوط به آن کاربر از حافظه نهان (cache) استخراج می‌شود و در متغیر address قرار می‌گیرد. سپس آدرس روی خروجی چاپ می‌شود (مجدداً به منظور دیباگ کردن و بررسی صحت داده‌ها).

سپس، با بررسی شرط if، اطمینان حاصل می‌شود که آدرس وجود دارد. در این صورت، یک پیام JSON به همراه کد وضعیت ۲۰۰ و آدرس مورد نظر (به صورت دیکد شده از نوع bytes) برگشت داده می‌شود.

در غیر این صورت (یعنی اگر آدرسی برای کاربر پیدا نشود)، یک پیام JSON خطا به همراه کد وضعیت ۴۰۴ (با محتوای 'Peer not found') برگشت داده می‌شود.

همتا

در ابتدای کار که نرم افزار اجرا میشود
یک منو به شکل زیر به کاربر نمایش داده میشود در این بخش از گزارش هر بخش از منو را توضیح داده و عملکرد آن را بررسی میکنیم .



ثبت نام

```python
def send_peer_profile():      soleimani101, 1 hour ago • first commit
    username = input("Enter your username: ")
    hostname = socket.gethostname()
    ip_address = socket.gethostbyname(hostname)

    Port = random.randint(8001, 8100)
    print(ip_address,Port)
    url = 'http://localhost:8000/send_peer_profile'
    data = {
        'username': username,
        'address':  f'({ip_address},{Port})',
    }
    try:
        response = requests.post(url, json=data)
        response.raise_for_status()  # Raise an exception if an HTTP error occurred
        data = response.json()
        if 'message' in data:
            print(data['message'])
        else:
            print('Error: Unexpected response')
    except requests.exceptions.RequestException as e:
        print(f'Error: {e}')
```

این برنامه یک کلاینت است که از کاربر نام کاربری را دریافت می‌کند و سپس اطلاعات مربوط به خود را به صورت یک درخواست POST به سرور ارسال می‌کند.

در ابتدا، نام کاربری از کاربر با استفاده از تابع input دریافت می‌شود.

سپس، با استفاده از تابع socket.gethostname()، نام میزبانی (hostname) محلی که برنامه در آن اجرا می‌شود، دریافت می‌شود. سپس با استفاده از تابع socket.gethostbyname(hostname)، آدرس IP مربوط به این میزبان را دریافت می‌کنیم و در متغیر ip_address قرار می‌دهیم.

سپس، یک پورت تصادفی در محدوده ۸۰۰۱ تا ۸۱۰۰ با استفاده از تابع random.randint() انتخاب می‌شود و در متغیر Port ذخیره می‌شود.

سپس، آدرس مقصد برای ارسال درخواست POST تعیین می‌شود که در این مثال 'http://localhost:۸۰۰۰/send_peer_profile' است.

در متغیر data، اطلاعات کاربر (نام کاربری و آدرس) در قالب یک دیکشنری ذخیره می‌شود.

در بلوک try-except، درخواست POST با استفاده از تابع requests.post() ارسال می‌شود. سپس با استفاده از تابع response.raise_for_status()، در صورتی که درخواست HTTP خطا داشته باشد، یک استثنا رخ می‌دهد.

سپس، با استفاده از تابع response.json()، پاسخ دریافتی از سرور به صورت JSON تجزیه و تحلیل می‌شود. اگر در پاسخ کلید 'message' وجود داشته باشد، پیام مربوطه چاپ می‌شود. در غیر این صورت، پیام خطای 'Unexpected response' چاپ می‌شود.

در صورت بروز خطا در ارسال درخواست (به طور مثال، اتصال به سرور برقرار نشود)، استثنا requests.exceptions.RequestException صادر می‌شود و پیام خطای مربوطه به همراه جزئیات نمایش داده می‌شود.

و درکل عملکرد این تابع برای ثبت نام کاربر ها است .

نمایش همه ی کاربران

```python
def request_get_all_peers():      soleimani101, 1 hour ago • 
    url = 'http://localhost:8000/get_all_peers'
    response = requests.get(url)
    if response.status_code == 200:
        data = response.json()
        peers = data.get('peers')
        if peers:
            print("List of Peers:")
            for peer in peers:
                print(peer)
        else:
            print("No peers found")
    else:
        print(f"Error: {response.status_code}")
```

این تابع برای دریافت لیستی از تمام همتایان (peers) از سرور استفاده می‌شود.

ابتدا، آدرس مقصد برای ارسال درخواست GET تعیین می‌شود که در این مثال 'http://localhost:۸۰۰۰/get_all_peers' است.

سپس، با استفاده از تابع `requests.get()`، یک درخواست GET به سرور ارسال می‌شود و پاسخ دریافتی در متغیر `response` ذخیره می‌شود.

سپس، با بررسی کد وضعیت پاسخ (status code) از طریق `response.status_code`، بررسی می‌شود که آیا درخواست با موفقیت انجام شده یا خیر.

در صورتی که کد وضعیت ۲۰۰ باشد، یعنی درخواست با موفقیت انجام شده است، اطلاعات دریافتی به صورت JSON تجزیه و تحلیل می‌شوند و لیستی از همتایان از متغیر `data` استخراج می‌شود با استفاده از `data.get('peers')`. در صورت وجود همتایان، آن‌ها چاپ می‌شوند. در غیر این صورت، پیام "No peers found" چاپ می‌شود.

در صورتی که کد وضعیت پاسخ ۲۰۰ نباشد، به این معنی است که درخواست با خطا مواجه شده است و پیام خطا به همراه کد وضعیت پاسخ چاپ می‌شود.

گرفتن آدرس یک کاربر خاص

```python
def request_special_peer():
    username = input("Enter the username: ")
    url = f'http://localhost:8000/get_peer_info/{username}'
    response = requests.get(url)
    if response.status_code == 200:
        data = response.json()
        address = data.get('address')
        print(address)

        if address:
            address = address.strip('()')  # Remove parentheses
            ip, port = address.split(',')  # Split IP and port
            ip = ip.strip()  # Remove whitespace
            port = int(port.strip())  # Convert port to integer
            print(f"Peer {username}: IP={ip}, Port={port}")
            return ip, port

        else:
            print(f"Peer {username} not found")
    else:
        print(f"Peer {username} not found")
```

این تابع برای دریافت اطلاعات مربوط به یک همتا (peer) خاص از سرور استفاده می‌شود.

در ابتدا، نام کاربری مربوط به همتا را از کاربر با استفاده از تابع input دریافت می‌کند.

سپس، با استفاده از نام کاربری دریافت شده، آدرس مقصد برای ارسال درخواست GET تعیین می‌شود. آدرس درخواست به شکل http://localhost:۸۰۰۰/get_peer_info/{username} است.

سپس، با استفاده از تابع requests.get()، یک درخواست GET به سرور ارسال می‌شود و پاسخ دریافتی در متغیر response ذخیره می‌شود.

سپس، با بررسی کد وضعیت پاسخ (status code) از طریق response.status_code، بررسی می‌شود که آیا درخواست با موفقیت انجام شده یا خیر.

در صورتی که کد وضعیت ۲۰۰ باشد، یعنی درخواست با موفقیت انجام شده است، اطلاعات دریافتی به صورت JSON تجزیه و تحلیل می‌شوند و آدرس همتا از متغیر data استخراج می‌شود با استفاده از data.get('address'). آدرس چاپ می‌شود.

سپس، در صورت وجود آدرس، آن را پردازش می‌کنیم. ابتدا پرانتزها را از آدرس حذف می‌کنیم (با استفاده از address.strip('()'))، سپس آدرس را براساس کاما جدا می‌کنیم و به IP و پورت تقسیم می‌کنیم (با

استفاده از address.split(',')، و در نهایت IP و پورت را پردازش می‌کنیم. IP را از فاصله‌های اضافی پاک می‌کنیم و پورت را به صورت عدد صحیح تبدیل می‌کنیم (با استفاده از ip.strip() و int(port.strip())) و آن‌ها را چاپ می‌کنیم. سپس آدرس IP و پورت به عنوان خروجی تابع برگردانده می‌شود.

در صورتی که آدرس وجود نداشته باشد، پیام "Peer {username} not found" چاپ می‌شود.

در صورتی که کد وضعیت پاسخ ۲۰۰ نباشد، به این معنی است که همتا موردنظر پیدا نشده است و پیام "Peer {username} not found" چاپ می‌شود.

اتصال به کاربر های دیگر

پس از انتخاب این گزینه صفحه ی زیر برای شما باز میشود :

```
--------------------------------
1. Signup
2. Display all users
3. Get address of a user
4. Connect to a user
0. Cancel
--------------------------------


Enter your choice: 4
        (1) Do you want to get online and wait for Image files?
        (2) Send Images?
        (3) get online and wait for text?
        (4) send text message?
```

در این بخش شما باید انتخاب کنید که میخواهید تصویر انتقال بدهید یا میخواهید از قابلیت چت برنامه استفاده کنید .

ارسال تصویر

در ابتدا باید با یک کاربر انلاین شویم برای این کار باید گزینه ی اول یعنی :

(۱) Do you want to get online and wait for Image files?

را انتخاب کنیم

پس از این بخش تابع زیر اجرا میشود .

```
def Listen_for_files():

        ############################
        name = input("who are you ???")
        send_peer_profile_inapp(name)
        Ip , port = request_special_peer_in_app(name)
        IP_PORT =(Ip, port)
        globalPort = port
        print("Waiting for connection...")

        your_ip = "127.0.0.1"  # IP address of the receiver
        your_port = globalPort  # Port number to listen on
        receiver_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        receiver_socket.bind((your_ip, your_port))

        image_path = "rec.jpg"  # Replace with the desired path to save the received image

        # Receive the number of packets from the sender
        num_packets_data, sender_address = receiver_socket.recvfrom(1024)
        num_packets = int(num_packets_data.decode())

        # Prompt the receiver to accept or reject the file
        response = input("Do you want to accept the received file? (y/n): ")

        # Send the acceptance response to the sender
        if response.lower() == 'y':
            receiver_socket.sendto('accept'.encode(), sender_address)
        else:
            receiver_socket.sendto('reject'.encode(), sender_address)

        # If accepted, receive and save the image data packets
        if response.lower() == 'y':
            image_data = b''
            for i in range(num_packets):
                packet_data, sender_address = receiver_socket.recvfrom(1024)
                image_data += packet_data
```

این تابع برای گوش دادن به درخواست‌ها و دریافت فایل‌ها از سمت سرور و دیگر کاربران استفاده می‌شود.

ابتدا، نام کاربر را از ورودی کاربر با استفاده از تابع `input` دریافت می‌کند و آن را در متغیر `name` ذخیره می‌کند.

سپس، با استفاده از توابع `send_peer_profile_inapp` و `request_special_peer_in_app`، اطلاعات مربوط به همتا را دریافت می‌کند. IP و پورت همتا به عنوان `ip` و `port` در متغیرهای مربوطه ذخیره می‌شوند و آدرس به صورت `IP_PORT` ترکیب می‌شوند. همچنین، پورت عمومی را در متغیر `globalPort` ذخیره می‌کند.

سپس، آدرس IP و پورت مورد نظر برای گوش دادن و دریافت فایل‌ها را تعیین می‌کند. IP به عنوان `your_ip` ثابت برابر با "۱۲۷.۰.۰.۱" قرار می‌گیرد و پورت برابر با `globalPort` قرار می‌گیرد.

سپس، یک سوکت جدید با استفاده از `socket.socket()` ایجاد می‌شود، که از نوع `socket.AF_INET` است و برای ارتباط UDP استفاده می‌شود. سپس سوکت به آدرس و پورت تعیین شده برای گوش دادن بایند می‌شود با استفاده از `receiver_socket.bind((your_ip, your_port))`.
مسیر فایلی که فایل دریافتی در آن ذخیره می‌شود، در متغیر `image_path` تعیین می‌شود.
سپس، تعداد بسته‌هایی که از سمت فرستنده دریافت خواهد شد را از فرستنده دریافت می‌کند. ابتدا داده تعداد بسته‌ها و آدرس فرستنده دریافت می‌شود با استفاده از `receiver_socket.recvfrom(۱۰۲۴)` و سپس داده را به عدد تبدیل می‌کند.سپس، کاربر را مجاب می‌کند تا فایل دریافتی را قبول یا رد کند با استفاده از `input` و آن را در متغیر `response` ذخیره می‌کند.
سپس، با توجه به پاسخ کاربر، پاسخ قبول یا رد را به فرستنده ارسال می‌کند.
در صورتی که پاسخ کاربر "y" باشد، رشته 'accept' را به صورت بایت و با استفاده از `receiver_socket.sendto('accept'.encode(), sender_address)` ارسال می‌کند. در غیر این صورت، رشته 'reject' را ارسال می‌کند.
در صورتی که پاسخ کاربر "y" باشد و فایل را قبول کرده باشد، بسته‌های داده‌ای که حاوی تصویر دریافتی است را دریافت می‌کند. از آنجایی که هر بسته ۱۰۲۴ بایت است، در هر مرحله بسته‌ها را دریافت کرده و به داده تصویر اضافه می‌کند.
سپس، داده تصویر دریافت شده را در یک فایل با نام 'received_image.jpg' ذخیره می‌کند.
در نهایت، سوکت را می‌بندد با استفاده از `receiver_socket.close()`.

سپس از شما نام کاربریتان را برای اپدیت کردن مقادیر سرور میپرسد و متنی که در ترمینال برای شما نمایش داده میشود به شکل زیر خواهد بود :

```
Enter your choice: 4
        (1) Do you want to get online and wait for Image files?
        (2) Send Images?
        (3) get online and wait for text?
        (4) send text message?
1
who are you ???soli
User profile saved successfully
(127.0.1.1,8010)
Peer soli: IP=127.0.1.1, Port=8010
Waiting for connection...
```

سپس در ترمینال دیگر با کاربر دیگری وارد میشوید و مراحل زیر را طی میکنید

```
Enter your choice: 4
        (1) Do you want to get online and wait for Image files?
        (2) Send Images?
        (3) get online and wait for text?
        (4) send text message?
1
who are you ???soli
User profile saved successfully
(127.0.1.1,8010)
Peer soli: IP=127.0.1.1, Port=8010
Waiting for connection...
Do you want to accept the received file? (y/n): y
--------------------------------
```

```
Enter your choice: 4
        (1) Do you want to get online and wait for Image files?
        (2) Send Images?
        (3) get online and wait for text?
        (4) send text message?
2
Enter the name of the user: soli
Enter the name of the file: img.png
(127.0.1.1,8010)
Peer soli: IP=127.0.1.1, Port=8010
File accepted. Sending...
100%|                    | 52/52 [00:00<00:00, 73410.91packets/s]
File sent successfully.
--------------------------------
```

از شما نام یوزری که میخواهید به او متصل شوید را میپرسد سپس نام فایلی که میخواهید برای او بفرستید را میپرسد .
سپس از طرف گیرنده پیغامی مبنی بر تایید این که میخواهید فایل را دریافت کنید دریافت میشود پس از تایید پروسه ارسال تصویر شروع میشود .

کد بخش ارسال عکس به شکل زیر میباشد .

```python
def SendImage():
    name = input("Enter the name of the user: ")
    file_name = input("Enter the name of the file: ")

    ip, port = request_special_peer_in_app(name)
    IP_PORT =(ip, port)
    globalPort = port
    receiver_ip = "127.0.0.1"  # IP address of the receiver
    receiver_port = globalPort  # Port number of the receiver
    sender_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    image_path = file_name


    # Define the IP address and port to bind the sender


    # Create a socket object
    sender_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    # Read the image file as binary data
    with open(image_path, 'rb') as file:
        image_data = file.read()

    # Get the image size and calculate the number of packets needed
    image_size = len(image_data)
    packet_size = 1024
    num_packets = math.ceil(image_size / packet_size)

    # Send the number of packets to the receiver
    receiver_address = ('127.0.0.1', receiver_port)  # Replace with receiver's IP address and port      soleiman
    sender_socket.sendto(str(num_packets).encode(), receiver_address)

    # Wait for the acceptance response from the receiver
    response, _ = sender_socket.recvfrom(1024)
    response = response.decode()

    if response == 'accept':
```

این تابع برای ارسال یک فایل تصویر از طریق پروتکل UDP به همتای موردنظر استفاده می‌شود.

ابتدا، نام کاربر را از ورودی کاربر با استفاده از `input` دریافت می‌کند و آن را در متغیر `name` ذخیره می‌کند. سپس، نام فایل تصویر را از ورودی کاربر دریافت می‌کند و آن را در متغیر `file_name` ذخیره می‌کند.

سپس، با استفاده از تابع `request_special_peer_in_app`، اطلاعات مربوط به همتای موردنظر را دریافت می‌کند. IP و پورت همتا به عنوان `ip` و `port` در متغیرهای مربوطه ذخیره می‌شوند و آدرس به صورت `IP_PORT` ترکیب می‌شوند. همچنین، پورت عمومی را در متغیر `globalPort` ذخیره می‌کند.

سپس، آدرس IP و پورت فرستنده را تعیین می‌کند. IP فرستنده را به عنوان '۱۲۷.۰.۰.۱' تعیین می‌کند و پورت را برابر با `globalPort` قرار می‌دهد.

سپس، یک سوکت جدید با استفاده از `socket.socket()` ایجاد می‌شود، که از نوع `socket.AF_INET` است و برای ارتباط UDP استفاده می‌شود.

سپس، فایل تصویر را به صورت داده‌های باینری می‌خواند با استفاده از `open` و `read` و آن را در متغیر `image_data` ذخیره می‌کند.

سپس، سایز تصویر را محاسبه کرده و تعداد بسته‌هایی که برای ارسال تصویر نیاز است را محاسبه می‌کند. سایز هر بسته برابر با `packet_size` است که در اینجا ۱۰۲۴ است. تعداد بسته‌ها را با استفاده از `math.ceil` و `image_size` و `packet_size` محاسبه می‌کند.

سپس، تعداد بسته‌ها را به فرستنده ارسال می‌کند. آدرس فرستنده

را به عنوان `receiver_address` مشخص می‌کند و از روی آدرس و پورت مقصد، یک تاپل ایجاد می‌کند. سپس با استفاده از `sender_socket.sendto`، تعداد بسته‌ها را به صورت رشته عددی به فرستنده ارسال می‌کند.

سپس، منتظر پاسخ از فرستنده می‌ماند. با استفاده از `sender_socket.recvfrom`، پاسخ را از فرستنده دریافت می‌کند. پاسخ را به عنوان رشته دریافت کرده و در متغیر `response` ذخیره می‌کند.

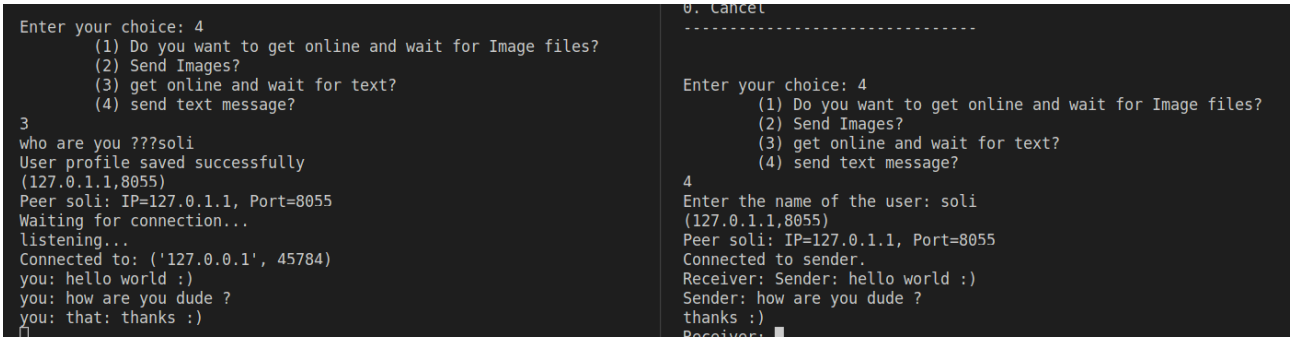
اگر پاسخ "accept" باشد، به کاربر اعلام می‌کند که فایل قبول شده است و شروع به ارسال می‌کند. سپس، داده‌های تصویر را به بسته‌ها تقسیم کرده و آن‌ها را به فرستنده ارسال می‌کند. هر بسته از طریق `sender_socket.sendto` به فرستنده ارسال می‌شود. همچنین، پیشرفت ارسال را با استفاده از `tqdm` نمایش می‌دهد.

در صورتی که پاسخ "reject" باشد، به کاربر اعلام می‌کند که فایل رد شده است.

در نهایت، سوکت را با استفاده از `sender_socket.close()` بسته می‌کند.

ارسال متن

در بخش قبلی برای منتظر ماندن و متصل شدن برای چت کردن گزینه ی ۳ را انتخاب میکنیم و پیغام زیر نمایش داده میشود .



همانطور که مشاهده میشود اتصال به شکل قبل اما با پروتکل tcp برقرار میشود
توضیح کد این بخش :

```python
def SendText():

    name = input("Enter the name of the user: ")

    ip, port = request_special_peer_in_app(name)
    globalPort = port
    receiver_ip = ip  # IP address of the receiver
    receiver_port = globalPort  # Port number of the receiver

    # Create a socket object
    receiver_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    # Connect to the sender      soleimani101, 2 hours ago • first commit
    receiver_socket.connect((receiver_ip, receiver_port))
    print("Connected to sender.")

    # Function to send messages
    def send_messages():
        while True:
            # Send data to the sender
            message = input("Receiver: ")
            receiver_socket.send(message.encode())

            # Check for the disconnect command
            if message == 'disconnect':
                break

    # Function to receive messages
    def receive_messages():
        while True:
            # Receive data from the sender
            data = receiver_socket.recv(1024)
```

این قطعه کد یک تابع به نام "SendText" را تعریف می‌کند که برای برقراری ارتباط بین دو کاربر بر اساس پروتکل TCP استفاده می‌شود. این تابع دارای یک ورودی است که نام کاربر را از ورودی دریافت می‌کند.

ابتدا، تابع از طریق تابع "request_special_peer_in_app" آدرس IP و پورت مقصد را برای برقراری ارتباط با کاربر مشخص شده دریافت می‌کند. سپس، یک اتصال TCP بین کاربر جاری و کاربر مقصد برقرار می‌شود.

سپس، دو تابع دیگر به نام‌های "send_messages" و "receive_messages" تعریف می‌شوند. تابع "send_messages" برای ارسال پیام‌ها به کاربر مقصد استفاده می‌شود و در یک حلقه بی‌نهایت اجرا می‌شود. ابتدا کاربر از ورودی یک پیام را وارد می‌کند و سپس پیام را از طریق اتصال TCP به کاربر مقصد ارسال می‌کند. اگر پیام "disconnect" باشد، حلقه توقف می‌یابد و اجرای تابع به پایان می‌رسد.
تابع "receive_messages" نیز در یک حلقه بی‌نهایت اجرا می‌شود و برای دریافت پیام‌ها از کاربر مقصد استفاده می‌شود. در هر دور از حلقه، پیام از طریق اتصال TCP دریافت می‌شود و سپس نمایش داده می‌شود. اگر پیام "disconnect" باشد، حلقه توقف می‌یابد و اجرای تابع به پایان می‌رسد.

پس از تعریف این دو تابع، دو رشته جداگانه به نام‌های "send_thread" و "receive_thread" ایجاد می‌شوند و هر کدام از این رشته ها به ترتیب توابع "send_messages" و "receive_messages" را اجرا می‌کنند.
در انتها، منتظر شدن برای پایان اجرای هر دو نخ و سپس بستن اتصال TCP صورت می‌گیرد.
بخش دریافت متن هم تقریبا مشابه همین عمل میکند .