به نام خدا

دانشگاه صنعتی امیرکبیر

دانشکده مهندسی کامپیوتر

گزارش پروژه شبکه های کامپیوتری

نام و نام خانوادگی	
شماره دانشجویی	

فهرست گزارش

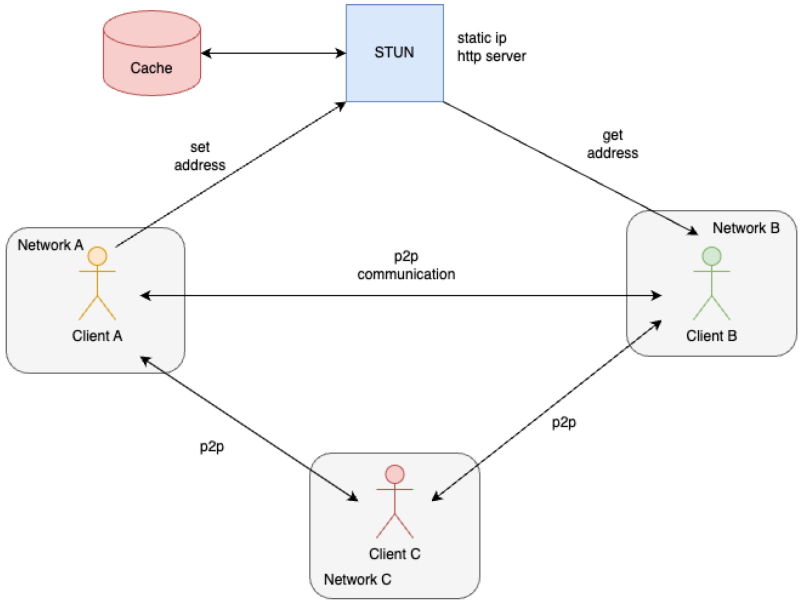
معماری سیستم

هدف این پروژه پیاده سازی یک سیستم برای اشتراک گذاری فایل به صورت نظیر به نظیر است.

هر کاربر ابتدا به سرور اصلی متصل شده و نام یکتای خود را وارد کرده و آدرس و شماره پورت خود را به سرور می فرستد. سرور نیز این مقادیر را برای هر کاربر ذخیره می کند. سپس یک لیست از اسامی کاربران ثبت شده در حافظه (Redis) به کاربر می فرستد تا کاربر یکی از آن ها را برای گرفتن فایل انتخاب کند. بعد از انتخاب کاربر مورد نظر، سرور اطلاعات لازم برای ارتباط با آن کاربر انتخابی را به کاربر انتخاب کننده می فرستد.

در این پروژه فرض شده هر کاربر دارای یک فایل my_file.txt و یک فایل my_file.jpg است که می تواند با دیگر کاربران به اشتراک بگذارد. هر کاربر پس از گرفتن اطلاعات لازم برای ارتباط با آن کاربر انتخابی اش، یکی از این فایل ها را از آن کاربردرخواست می کند. فایل متنی با tcp و فایل تصویری با udp تبادل می شود.

شکل زیر شمای معماری کلی سیستم را نشان می دهد:



سرور مدیریت آدرس همتا ها (STUN server)

این سرور، یک سرور HTTP می باشد که برای ذخیره اطلاعات از حافظه Redis استفاده می کند. در این حافظه نام کاربر به عنوان کلید و آدرس و پورت کاربر به عنوان مقدار ذخیره می شود. ارتباطات سرور با کاربران به فرمت JSON انجام می شود. همچنین مقدار ها در Redis به فرمت JSON ذخیره می شوند.

این سرور در فایل stun.py پیاده شده است.

```
from http.server import HTTPServer, BaseHTTPRequestHandle
import json
import redis
import socket

server_ip = socket.gethostbyname(socket.gethostname())
server_port = 8000
redis_ip = '127.0.0.1'
redis_port = 6379

r = redis.Redis(host=redis_ip, port=redis_port, db=1)
```

در این فایل از کتابخانه socket ، json ، redis و http.servrer استفاده شده است.

از socket تنها برای گرفتن آدرس سرور، از json برای تبدیل مقادیر از و به فرمت JSON و از redis برای برقراری ارتباط و درخواست از Redis استفاده شده است.

ماژول BaseHTTPRequestHandler یک کلاس برای مدیریت درخواست های HTTP و تولید پاسخ های متناسب آن است که کلاس Handler ما با ارث بری از این کلاس، پاسخ های متناسب هر درخواست را تولید می کند. سپس این کلاس به HTTPServer داده می شود تا یک سرور HTTP با توجه به Handler بسازد.

همچنین در این قسمت به حافظه Redis متصل می شویم تا بتوانیم به آن درخواست بفرستیم.

```python
def register_user(name, ip, port):
    if r.exists(name) != 0:
        return False
    else:
        d = {"ip": ip, "port": port}
        j = json.dumps(d)
        r.set(name, j)
        return True


def get_user_data(name):
    j = r.get(name)
    return j


def get_list():
    list_users = r.keys()
    if list_users is None:
        return None
    list_users=[x.decode('utf-8') for x in list_users]
    j = json.dumps(list_users)
    return j
```

در این فایل توابعی برای ارتباط با Redis پیاده شده اند. برای ثبت نام یک کاربری بررسی می شود در صورتی که نامش قبلا در حافظه وجود داشته باشد، اجازه استفاده از این نام داده نمی شود. برای گرفتن اطلاعات ارتباطی یک کاربر، مقدار ذخیره شده در حافظه را برای آن کاربر بر می گردانیم. همچنین برای گرفتن لیست کاربران، همه کلید ها را گرفته و به فرمت JSON در می آوریم و بر می گردانیم.

دو نوع درخواست GET و یک نوع در خواست POST به سرور می تواند فرستاده شود و سرور متناسب با هر درخواست پاسخ می دهد.

```python
class Handler(BaseHTTPRequestHandler):
    def do_GET(self):
        if self.path == 'list':
            list_users = get_list()
            if list_users is not None:
                data = list_users.encode('utf-8')
                self.send_response(200)
                self.send_header("Content-Length", str(len(data)))
                self.end_headers()
                self.wfile.write(data)
                return
            else:
                self.send_response(400)
                self.end_headers()
                return

        elif self.path.startswith('user'):
            target_user_name = self.path[5:]
            user_data = get_user_data(target_user_name)
            if user_data is not None:
                data = user_data
                self.send_response(200)
                self.send_header("Content-Length", str(len(data)))
                self.end_headers()
                self.wfile.write(data)
                return
            else:
                self.send_response(400)
                self.end_headers()
                return
```

یک نوع از درخواست های GET ، برای گرفتن لیست تمام کاربران است که در path این درخواست مشخص می شود. در صورتی که سرور بطور موفقیت آمیز لیست کاربران را بگیرد، آن لیست را در قسمت داده پاسخ خود قرار داده و با کد 200 پاسخ را می فرستد. در غیر اینصورت، پاسخ با کد 400 می فرستد که بیانگر وجود خطا است.

نوع دیگر GET، گرفتن اطلاعات ارتباطی یک کاربر است که با ‘user/UserName’ در path مشخص می شود. در صورتی که آن کاربر در حافظه موجود باشد، داده های مربوطه به همراه کد 200 و در صورت عدم وجود کاربر، کد 400 پاسخ داده می شود.

```python
def do_POST(self):
    content_len = int(self.headers.get('Content-Length'))
    post_body = self.rfile.read(content_len)
    user_data = json.loads(post_body.decode('utf-8'))
    name = user_data["name"]
    ip = user_data["ip"]
    port = user_data["port"]
    successful_register = register_user(name, ip, port)
    if successful_register:
        self.send_response(200)
        self.end_headers()
        return
    else:
        self.send_response(400)
        self.end_headers()
        return
```

یک نوع در خواست POST نیز سرور ما پاسخ می دهد که کاربر برای ثبت نام، نام خود به همراه آدرس و پورت خود را در body پیام می فرستد. سپس سرور این مقادیر را در حافظه ذخیره می کند. در صورت موفقیت آمیز بودن این کار، پاسخ با کد 200 و در غیر اینصورت، با کد 400 فرستاده می شود.

```python
def run(server_class=HTTPServer, handler_class=Handler):
    server_address = (server_ip, server_port)
    httpd = server_class(server_address, handler_class)
    httpd.serve_forever()


if __name__ == "__main__":
    print(server_ip)
    run()
```

برای اجرا این سرور، یک نمونه از آن را با آدرس و پورت مشخص شده و کلاس Handler تعریف شده می سازیم و در تابع main، آن را اجرا می کنیم.

همتا (Peer)

هر کاربر دو بخش ارتباط با سرور و ارتباط با همتا دارد. کاربر ابتدا به سرور متصل شده و بعد از ثبت نام و گرفتن اطلاعات کاربر انتخابی، به آن کاربر (همتا) متصل شده و داده می گیرد.

این بخش در فایل client.py پیاده سازی شده است.

ارتباط با سرور

```
class ServerConnection:

    def __init__(self):
        self.conn = None

    def connect(self):
        try:
            print(Fore.RED+"Connecting to server...")
            conn = http.client.HTTPConnection(server_ip, server_port, timeout=5)
            print(Fore.RED+"Connection succeed")
            self.conn = conn
            return True
        except:
            print(Fore.RED+"Can not connect to server")
            return False
```

برای ارتباط با سرور، یک کلاس ServerConnection را تعریف کرده ایم. در قسمت connect، تلاش می کنیم به سرور متصل شویم. برای این کار از ماژول http.clinet.HTTPConnection استفاده می کنیم که یک ارتباط با سرور مورد نظر با timeout مشخص شده برای ما برقرار می کند. در صورت موفقیت آمیز بودن ارتباط مقدار True و در غیر این صورت، مقدار False بر می گردانیم.

```python
def post_user_data(self, name, ip, port):
    user_dict = {'name': name, 'ip': ip, 'port': port}
    user_dict = json.dumps(user_dict)
    data = user_dict.encode('utf-8')
    headers = {'Content-Length': str(len(data))}
    try:
        self.conn.request("POST", "name", data, headers)
        response = self.conn.getresponse()
        if response.status == 200:
            print(Fore.RED+"Name accepted")
            return True
        else:
            print(Fore.RED+"cannot use this name")
            return "retry_name"

    except:
        print(Fore.RED+"Connection failed")
        return False

def get_list(self):
    try:
        print(Fore.RED+"Getting list of users...")
        self.conn.request("GET", "list")
        response = self.conn.getresponse()
        if response.status == 200:
            rj = response.read()
            r = json.loads(rj.decode("utf-8"))
            return r
        else:
            print(Fore.RED+"No user yet")
            return "no_user_yet"

    except:
        print(Fore.RED+"Connection failed")
        return False

def get_user_data(self, name):
    try:
        print(Fore.RED+"Getting user data")
        self.conn.request("GET", "user/" + name)
        response = self.conn.getresponse()
        if response.status == 200:
            rj = response.read()
            r = json.loads(rj.decode("utf-8"))
            return r
        else:
            print(Fore.RED+"User is not valid")
            return None

    except :
        print(Fore.RED+"Connection failed")
        return False
```

سه تابع برای ارسال درخواست HTTP به سرور تعریف شده است. در تابع اول؛ نام، آدرس و شماره پورت کاربر را گرفته، آن را به فرمت JSON در می آوریم و بعد از مقدار دهی header های لازم، با نوع POST و path:’post’ آن را به سرور ارسال می کنیم. در صورت موفقیت آمیز بودن ثبت اطلاعات در سرور، پاسخ با کد 400 و در غیر اینصورت پاسخ با کد 200 دریافت می کنیم.

در تابع دوم، با فرستادن پیغام مناسب، لیست کاربران را در خواست می کنیم. در صورتی که کد پاسخ 400 باشد، پاسخ دارای لیست در قسمت body است. در غیر اینصورت به این معنی است که هنوز کاربری ثبت نشده است.

در تابع آخر نیز اطلاعات یک کاربر مشخص را به فرم توضیح داده شده در قسمت مربوط به سرور درخواست می کنیم. در صورتی که کد پاسخ 400 باشد، پاسخ دارای اطلاعات ارتباطی آن کاربر در قسمت body است. در غیر اینصورت به این معنی است که هنوز کاربری با آن نام ثبت نشده است.

ارتباط با همتا

برای تبادل داده با همتا، نیازمند طراحی پروتکل هستیم.

```
class ClientSession:
    def __init__(self):
        self.name = None
        self.ip = socket.gethostbyname(socket.gethostname())
        self.welcome_port = random.randint(49152,65535)
        self.welcome_sock = None
```

برقراری ارتباط با همتا در کلاس ClientSession انجام می شود. در ابتدا برای ساخت یک کاربر آدرس و شماره پورت آن مقدار دهی شده تا به سرور فرستاده شود. نام کاربر نیز بعد از تایید سرور مقداردهی می شود.

بعد از دریافت اطلاعات ارتباطی همتا، کاربر یک درخواست برای گرفتن عکس یا متن می فرستد.

```
def initialize(self):
    self.welcome_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    self.welcome_sock.bind((self.ip, self.welcome_port))
    t = threading.Thread(target=self.wait_for_peers)
    t.start()

def wait_for_peers(self):
    while True:
        msg, adr = self.welcome_sock.recvfrom(1024)
        if msg is not None and msg.decode("utf-8") == 'IMG':
            self.welcome_sock.sendto('ACK'.encode("utf-8"),adr)
            t = threading.Thread(target=self.send_img, args=(adr,))
            t.start()
        elif msg is not None and msg.decode("utf-8") == 'TXT':
            self.welcome_sock.sendto('ACK'.encode("utf-8"),adr)
            t = threading.Thread(target=self.send_txt, args=(adr,))
            t.start()
```

به این منظور، هر کاربر بعد از ثبت نام در سرور، یک ریسمان می سازد و در آن ریسمان منتظر اتصال باقی همتا ها به او برای گرفتن داده می ماند. در صورتی که همتایی درخواست اتصال به کاربر بدهد، کاربر برای همتا پیغام ACK فرستاده و در ریسمانی جداگانه قطعه کد مربوط به کنترل ارسال تصویر یا متن را اجرا می کند.

تبادل متن

برای تبادل متن از tcp استفاده می شود.

```
def req_txt(self, target_ip, target_port, retry_attemps = 0):
    if retry_attemps == 3:
        print(Fore.YELLOW+"Text file transfer was unsuccessful.")
        return False

    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    req_port = random.randint(49152,65535)
    sock.bind((self.ip, req_port))  #just a random port
    sock.sendto('TXT'.encode("utf-8"), (target_ip, target_port))
    sock.settimeout(1.0)
    # waiting for ack
    try:
        ack , adr = sock.recvfrom(1024)
        if ack.decode("utf-8") == "ACK":
            pass
        else:
            return self.req_txt(target_ip,target_port,retry_attemps= retry_attemps+1)
    except:
        return self.req_txt(target_ip,target_port,retry_attemps= retry_attemps+1)

    # waiting for other peer to connect
    serv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    serv.bind((self.ip, req_port))
    serv.listen(1)
    serv.settimeout(1.0)
    try:
        data_sock, addr = serv.accept()
        serv.close()
        try:
            print(Fore.YELLOW+"Recieving text from peer...")
            data_sock.settimeout(1.0)
            data = data_sock.recv(1048576)
        except:
            data_sock.close()
            return self.req_txt(target_ip, target_port,retry_attemps=retry_attemps+1)

        try:
            file = open(target_ip + '.txt', 'wb')
            file.write(data)
            file.close()
            print(Fore.YELLOW+"Text file transfer was done successfully.")
            return True
        except:
            print(Fore.YELLOW+"File error in client.")
            file.close()
            return False


    except:
        print(Fore.YELLOW+"Peer is not online.")
        serv.close()
        return False
```

هر کاربر برای درخواست متن، تابع فوق را اجرا می کند که آدرس و شماره پورت مقصد را به عنوان ورودی می گیرد. در طول اجرای این تابع در صورتی که ارتباط قطع شده و یا خطایی پیش بیاید، این تابع تا سقف حداکثر ۳ بار، مجدد صدا زده می شود. در این تابع ابتدا درخواست گرفتن متن به آدرس و پورت همتا فرستاده می شود و متنظر پیام ACK می ماند(در صورت عدم دریافت این پیغام تا ۱ ثانیه مجدد تا سقف ۳ بار درخواست ارسال می شود). بعد از دریافت پیام ACK، کاربر یک سوکت tcp می سازد و منتظر می ماند تا همتا برای ارسال متن به او وصل شود. در صورتی که بعد از ۱ ثانیه همتا وصل نشد؛ دریافت متن ناموفق در مظر گرفته می شود.

بعد از اتصال همتا به سوکت tcp، تمام داده متنی به صورت یک ارسال، فرستاده می شود (در صورت بروز مشکل در این حین نیز تا سقف ۳ بار تلاش مجدد می شود). سپس کاربر یک فایل به اسم ip همتا باز کرده و متن را در آن ذخیره می کند.

```
def send_txt(self, adr):
    try:
        data_sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        data_sock.settimeout(1.0)
        data_sock.connect(adr)
        file = open('my_file.txt', 'rb')
        data = file.read()
        data_sock.send(data)
        file.close()
        data_sock.close()

    except :
        print(Fore.YELLOW+"Unable to send data to peer")
        return
```

در سمت فرستنده نیز برای ارسال متن، ابتدا تلاش می شود که به سوکت tcp بوجود آمده در سمت گیرنده وصل شود. سپس فایل را باز کزده و تمام فایل را بصورت یکجا برای گیرنده می فرستد. در صورتی که در این بین هر مشکلی پیش بیاید، اجرای این تابع متوقف شده و فرستده باید دوباره درخواست دهد تا فایل برایش فرستاده شود.

تبادل تصویر

داده های تصویری باید با استفاده از سوکت udp منتقل شوند؛ با این حال پروتکل ما باید به طور صحیح همه داده های تصویری را دریافت کند. برای تبدیل داده های تصویری به بسته و تبدیل آن به رشته بایت، و همچنین پیاده سازی مکانیزم کنترل خطا، یک کلاس تعریف می کنیم

```python
class DecEnc :

    def encode_wh_im_to_bytes(w,h):
        b = w.to_bytes(4,'big')
        b += h.to_bytes(4,'big')
        c = DecEnc.checksum(b)
        return b + c


    def decode_wh_im_to_int(b:bytes):
        if DecEnc.checksum(b[0:8]) != b[8:]:
            return False
        w = int.from_bytes(b[:4], "big")
        h = int.from_bytes(b[4:8], "big")
        return w, h

    def encode_im_chunk_to_bytes(pix,hi,width):
        by = hi.to_bytes(4,'big')
        for wi in range(width):
            r,g,b = pix[wi,hi]
            by += r.to_bytes(1,'big')
            by += g.to_bytes(1,'big')
            by += b.to_bytes(1,'big')
        c = DecEnc.checksum(by)
        return by + c

    def decode_im_chunk_to_int(by:bytes,pix,width):
        sc = width*3+4
        if DecEnc.checksum(by[0:sc]) != by[sc:]:
            return False
        hi = int.from_bytes(by[:4], "big")
        for wi in range(width):
            index  = 3 * wi
            r = int.from_bytes(by[index+4:index+4+1], "big")
            g = int.from_bytes(by[index+4+1:index+4+1+1], "big")
            b = int.from_bytes(by[index+4+2:index+4+2+1], "big")
            pix[wi,hi] = (r,g,b)
        return hi

    def encode_im_hi(hi):
        by = hi.to_bytes(4,'big')
        c = DecEnc.checksum(by)
        return by + c

    def decode_im_hi(by:bytes):
        if DecEnc.checksum(by[0:4]) != by[4:]:
            return False
        hi = int.from_bytes(by[:4], "big")
        return hi


    def checksum(data):
        """
        Calculate the 1-Byte checksum for an array of bytes.
        """
        return bytes([sum(data) & 0xFF])
```

این کلاس سه نوع تابع encode, decde ,checksum دارد. تابع checksum یک رشته بایت را به عنوان ورودی می گیرد و یک checksum ۱ بایتی با جمع تمام بایت های رشته بایت به عنوان خروجی می دهد.

۳ نوع بسته برای ارسال تصویر وجود دارند که باید encode و decode شوند. در encode، در انتها checksum محاسبه شده و به انتهای بسته اضافه می شود و در decode، مجدد روی قسمت داده بسته checksum محاسبه می شود و در صورت عدم تساوی checksum محاسبه شده با checksum درون بسته، بسته قبول نمی شود.

یک نوع بسته، شامل weight و height تصویر است که به صورت دو عدد int داده می شود و به بایت تبدیل می شود. یک نوع بسته شامل یک ردیف از ماتریس مربوط به تصویر است. نوع دیگر بسته تنها شامل یک شماره ردیف است.

```
def req_img(self, target_ip, target_port,retry_attemps = 0):
    num = 0
    if retry_attemps == 3:
        print(Fore.YELLOW+"Image file transfer was unsuccessful.")
        return False

    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    req_port = random.randint(49152,65535)
    sock.bind((self.ip, req_port))  #just a random port
    sock.sendto('IMG'.encode("utf8"), (target_ip, target_port))
    sock.settimeout(1.0)
    # waiting for ack
    try:
        ack , adr = sock.recvfrom(1024)
        if ack.decode("utf-8") == "ACK":
            pass
        else:
            return self.req_img(target_ip,target_port,retry_attemps= retry_attemps+1)
    except:
        return self.req_img(target_ip,target_port,retry_attemps= retry_attemps+1)

    # recieving image
    sock.settimeout(10.0)
    # getting image size
    try:
        msg , adr = sock.recvfrom(1024)
    except:
        return self.req_img(target_ip, target_port,retry_attemps = retry_attemps+1)

    result = DecEnc.decode_wh_im_to_int(msg)
    if result is False:
        return self.req_img(target_ip, target_port,retry_attemps = retry_attemps+1)
    else:
        sock.sendto("ACK_SIZE".encode("utf-8"), adr)
        width , height = result
        im = Image.new(mode="RGB" , size=(width,height))
        pix = im.load()

    # getting image pixels:
    pix_status = numpy.full((height), False, dtype=bool)
    print(Fore.YELLOW+"Getting image from peer...")
    while True:
        try:
            msg , a = sock.recvfrom(4096)
        except:
            return self.req_img(target_ip, target_port,retry_attemps = retry_attemps+1)

        try:
            text = msg.decode("utf-8")
        except:
            text = None

        if text is not None and text == "FIN":
            all_true = True
            for hi in range(height):
                if pix_status[hi] == False:
                    all_true = False
                    B = DecEnc.encode_im_hi(hi)
                    sock.sendto(B,adr)

            if all_true is False :
                sock.sendto("FIN".encode("utf-8"),adr)
            else:
                sock.sendto("COMPLETE".encode("utf-8"),adr)
                break


        else:
            result = DecEnc.decode_im_chunk_to_int(msg,pix,width)
            if result is not False:
                pix_status[result] = True
            num+=1


    im.save(target_ip+'.jpg')
    print(Fore.YELLOW+"Image file transfer was done successfully.")
    sock.close()
    return True
```

قسمت ابتدایی درخواست تصویر مشابه است. بعد از دریافت ACK از همتا، کاربر منتظر می ماند تا ابعاد تصویر را دریافت کند (در غیر اینصورت دوباره درخواست می کند). پس از دریافت ابعاد تصویر پیغام ACK_SIZE را به همتا می فرستد. سپس کاربر یک لیست به اندازه ارتفاع تصویر درست می کند و تمامی مقادیر آن را در ابتدا False قرار می دهد. سپس منتظر می ماند تا همتا، بسته های حاوی اطلاعات تصویر را ارسال کند. هر بسته شامل یک سطر از ماتریس تصویر است. در صورتی که بسته توسط کاربر پذیرفته شود، مقدار مربوط به آن سطر در لیست True می شود. فرستنده (همتا) بعد اتمام ارسال تمام بسته ها پیغام FIN می فرستد. در صورت مشاهده این پیغام توسط گیرنده (کاربر) روی لیست حرکت کرده و شماره ردیف هایی که هنوز دریافت نکرده را به فرستده می فرستد تا مجدد ارسال شوند؛ و در انتها پیغام FIN را

می فرستد. در صورتی که تمام بسته ها دریافت شده بود نیز پیغام COMPLETE فرستاده می شود و پس ذخیره تصویر، فرایند متوقف می شود.

```
def send_img(self, adr):
    try:
        data_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        data_sock.settimeout(10.0)
        im = Image.open('my_file.jpg')
    except Exception as e:
        print(Fore.YELLOW+"Unable to send data to peer")
        data_sock.close()
        return False

    pix = im.load()  # getting pixel values
    width, height = im.size  # Get the width and height of the image for iterating over
    B = DecEnc.encode_wh_im_to_bytes(width,height)
    try:
        data_sock.sendto(B,adr)
        msg , a = data_sock.recvfrom(1024)
        if msg.decode("utf-8") != "ACK_SIZE":
            return False
    except:
        data_sock.close()
        return False


    for hi in range(height):
        B = DecEnc.encode_im_chunk_to_bytes(pix,hi,width)
        data_sock.sendto(B,adr)


    data_sock.sendto("FIN".encode("utf-8"),adr)

    while True:
        try:
            msg , a = data_sock.recvfrom(1024)
        except:
            data_sock.close()
            return False

        try:
            text = msg.decode("utf-8")
        except:
            text = None

        if text is not None and text == "COMPLETE":
            data_sock.close()
            return True
        elif text is not None and text == "FIN":
            pass
        else:
            result = DecEnc.decode_im_hi(msg)
            if result is not False:
                hi = result
                B = DecEnc.encode_im_chunk_to_bytes(pix,hi,width)
                data_sock.sendto(B,adr)
```

در سمت فرستنده نیز ابتدا فایل تصویر باز می شود. سپس ابعاد تصویر به گیرنده فرستاده می شود و منتظر پیغام ACK_SIZE می ماند. سپس ابتدا هر سطر ماتریس تصویر را به عنوان یک بسته به گیرنده می فرستد و در آخر پیغام FIN را می فرستد. سپس منتظر ارسال داده توسط گیرنده می ماند. در صورتی که پیغام COMPLETE دریافت شد، ارسال داده موفقیت آمیز بوده و فرایند متوقف می شود. در غیر اینصورت گیرنده شماره ردیف هایی که دریافت نکرده را ارسال می کند و فرستده نیز مجدد این ردیف ها را می فرستد.

توجه شود در صورت بروز مشکل در ارسال داده، این تابع متوقف شده و منتظر درخواست مجدد می ماند

رابط کاربری

برای این بخش از رابط کاربری متنی استفاده شده است.

```python
def run():
    print(Fore.GREEN+"Press cntl + C whenever you want to exit")
    server_connection = ServerConnection()
    status = server_connection.connect()
    if status is False:
        return

    client = ClientSession()

    # setting name and address in server DB
    while True:
        print(Fore.GREEN+"Please enter your unique username:")
        username = input(Fore.WHITE)
        status = server_connection.post_user_data(username, client.ip, client.welcome_port)
        if status is True:
            client.name = username
            break
        elif status is False:
            return


    # now we can initialize client's welcome thread
    client.initialize()

    # getting list of users
    while True:
        users = server_connection.get_list()
        if users is False:
            return
        elif users == "no_user_yet":
            print(Fore.GREEN+"Press U to update users or any other key to exit")
            command = input(Fore.WHITE)
            if command != "U":
                return
        else:
            for name in users:
                print(Fore.BLUE+name)
            break

    # getting user data:
    while True:
        print(Fore.GREEN+"Please enter the name of the user you want to get files from:")
        target_name = input(Fore.WHITE)
        if target_name == client.name:
            print(Fore.GREEN+"You cannot get files from yourself.")
        else:
            user_data = server_connection.get_user_data(target_name)
            if user_data is False:
                return
            elif user_data is None:
                pass
            else:
                target_ip = user_data["ip"]
                target_port = user_data["port"]
                break

    # getting text file or image file
    while True:
        print(Fore.GREEN+"Enter I to get image file or T to get text file:")
        s = input(Fore.WHITE)
        if s == 'I':
            client.req_img(target_ip, target_port)
            return
        elif s == 'T':
            client.req_txt(target_ip, target_port)
            return
        else:
            print(Fore.GREEN+"Invalid command.")


if __name__ == "__main__":
    run()
```

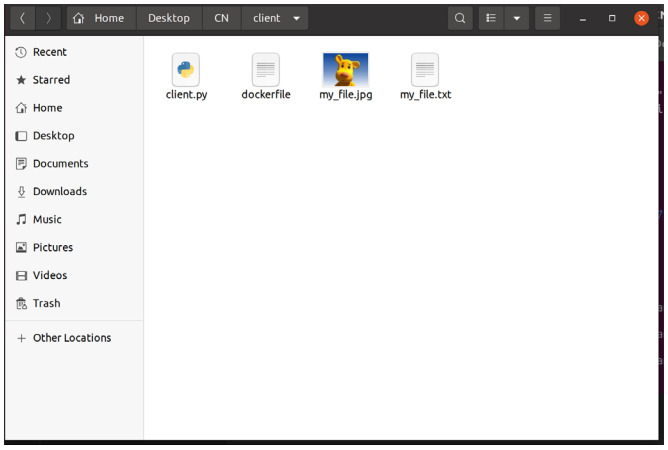
در این بخش ابتدا تلاش می شود به سرور متصل شد. سپس یک نمونه از ClientSession تولید می شود و از کاربر نام کاربری اش را درخواست می کند. در صورتی که نام کاربری یکتا نباشد، از کاربر مجدد درخواست می شود تا نام کاربری انتخاب کند. سپس تابع initialize روی نمونه کاربر صدا زده می شود تا ریسمان مربوط به قبول درخواست تبادل داده اجرا شود. سپس لیستی از همتا ها به کاربر نشان داده می شود و از کاربر نام همتایی که می خواهد از او داده بگیرد را ورودی می گیرد. در صورتی که نام معتبر نبود یا نام خود کاربر بود، دوباره از کاربر نام همتا را می گیرد. سپس از کاربر نوع داده درخواستی را می پرسد و شروع به تبادل پیام می کند.

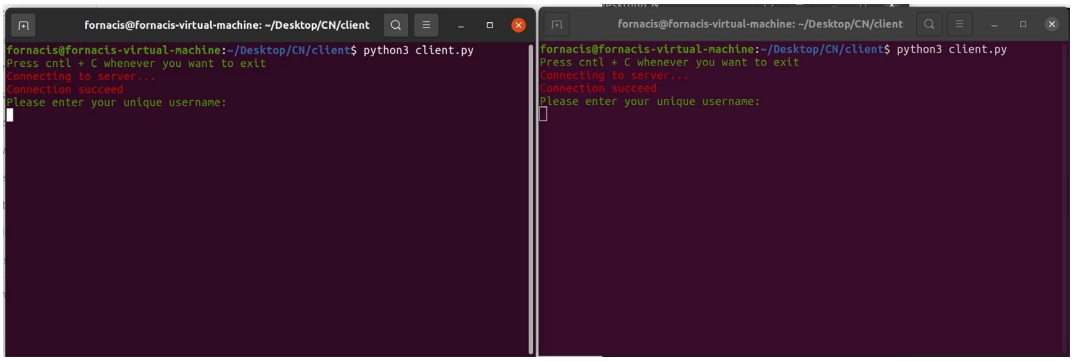
اجرای سیستم

بعد از نصب و راه اندازی redis، فایل مربوط به server را اجرا می کنیم

```
fornacis@fornacis-virtual-machine:~/Desktop/CN/server$ python3 stun.py
127.0.1.1
```

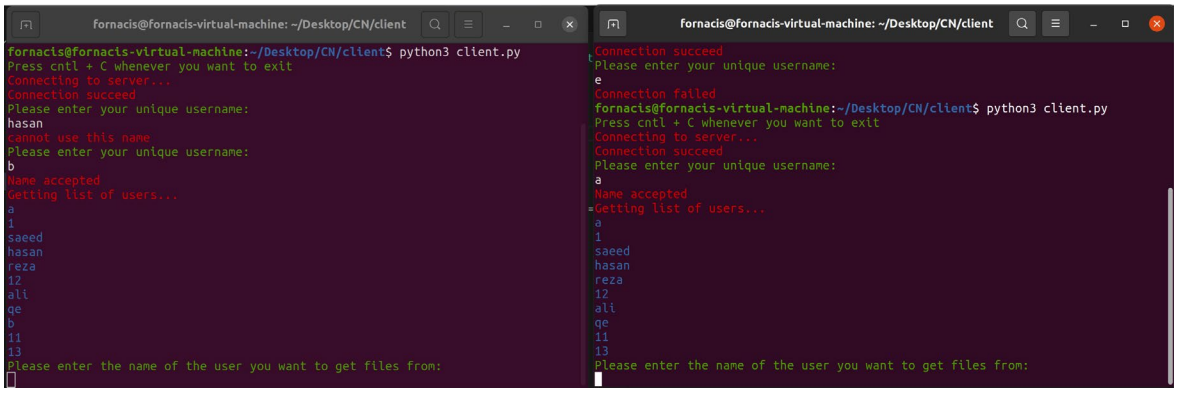
حال در دو ترمینال جدا فایل مربوط به client را دوبار اجرا می کنیم





در نظر داشته باشید در صورتی که server اجرا نشده باشد و ما برنامه client را اجرا کنیم؛ خطا دریافت می کنیم.

برای کاربران نام انتخاب می کنیم. در صورتی که نام قبلا انتخاب شده باشد، باید نامی دیگر انتخاب کنیم:



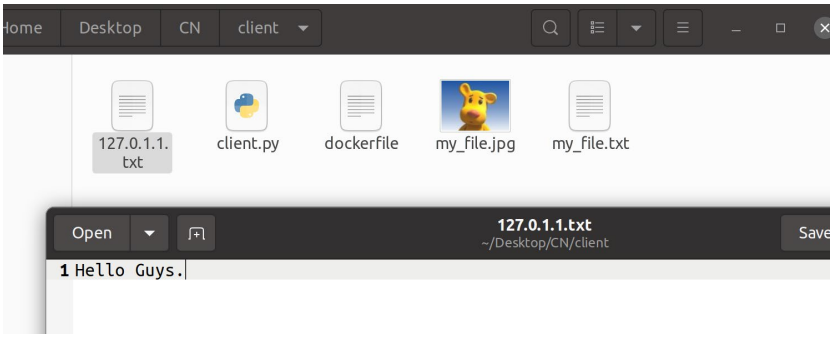
حال باید نام کاربری را که می خواهیم از او فایل دریافت کنیم را انتخاب کنیم.

در این مثال برای پوشش همه حالات ممکن، کاربر a ابتدا نامی وارد می کند که در سرور وجود ندارد و پیغام متناسب دریافت می کند. سپس کاربری را انتخاب می کند که online نیست. بعد از گرفتن اطلاعات آن کاربر، بین تصویر و متن انتخاب می کند. در نهایت چون کاربر انتخابی اش آنلاین نیست، پیغام مناسب نمایش داده می شود

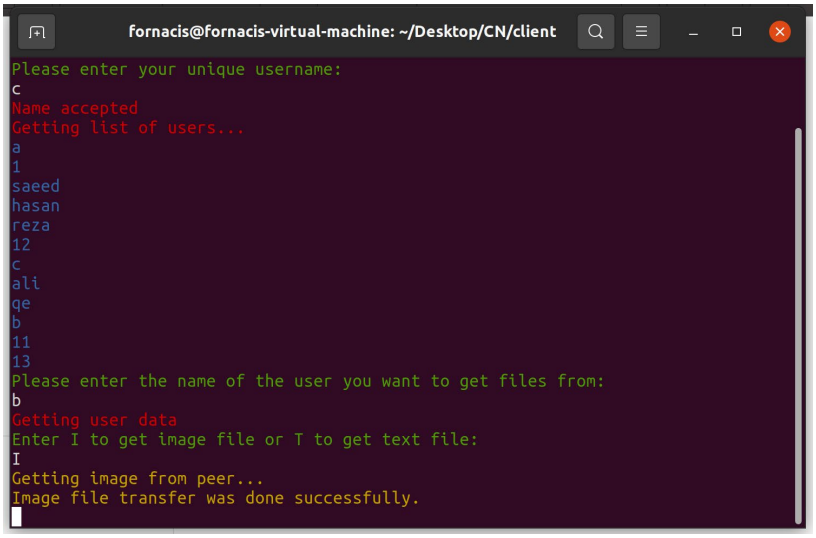
```
Please enter the name of the user you want to get files from:
nist
Getting user data
User is not valid
Please enter the name of the user you want to get files from:
hasan
Getting user data
Enter I to get image file or T to get text file:
T
Text file transfer was unsuccessful.
```

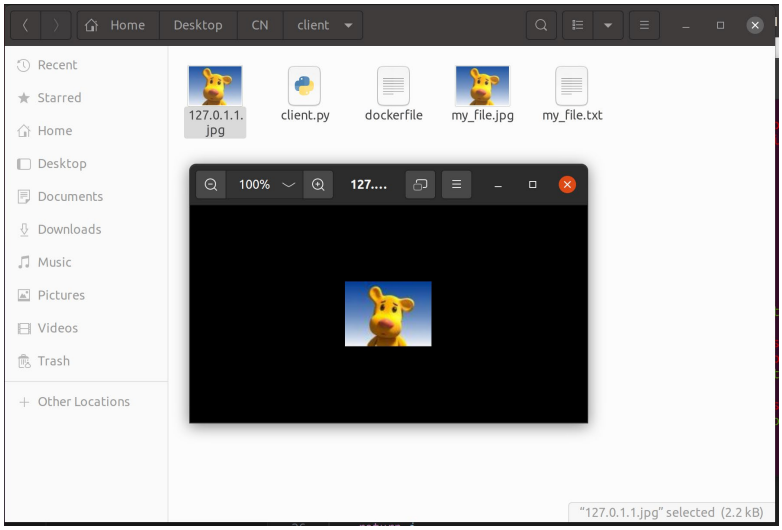
با این حال اجرای برنامه او متوقف نمی شود تا دیگران بتوانند از او فایل درخواست کنند. حال کاربر b از a درخواست فایل متنی می کند که این ارسال موفقیت آمیز است.

```
Please enter the name of the user you want to get files from:
a
Getting user data
Enter I to get image file or T to get text file:
T
Recieving text from peer...
Text file transfer was done successfully.
```



درخواست فایل تصویری نیز به این صورت انجام می شود. در این سناریو کاربر c از b درخواست تصویر می کند.





همچنین در صورت پیش آمدن مشکلی در حین دریافت اطلاعات، پس از ۳ تلاش ناموفق، دریافت متوقف می شود.