Department of
Computer Engineering

دانشگاه صنعتی امیرکبیر
( پلی تکنیک تهران )

# به نام خالق زیبایی ها

گزارش‌کار پروژه عملی

درس شبکه های کامپیوتری

امیرفاضل کوزه گر کالجی

۹۹۳۱۰۹۹

# STUN server:

در اینجا یک سرورِ استان داریم که با استفاده از کلاینت ردیس، امورر مربوط به ثبت پورت کاربران برنامه را مدیریت می‌کند.

کلاس HTTPRequestHandler، مدیریت امور مربوط به apiرا بر عهده دارد:

```python
class HTTPRequestHandler(BaseHTTPRequestHandler):
    def do_GET(self):
        parsed_url = urlparse(self.path)
        query_params = parse_qs(parsed_url.query)
        if self.path == '/users':
            self.send_response(200)
            self.send_header('Content-type', 'application/json')
            self.end_headers()
            keys = r.keys('*')
            data = {'users': [key.decode('utf-8')for key in keys]}
            json_data = json.dumps(data)
            self.wfile.write(json_data.encode('utf-8'))
        elif self.path.split('?')[0] == '/user':
            # Handle saving data logic here
            id = query_params['id']
            self.send_response(200)
            self.send_header('Content-type', 'application/json')
            self.end_headers()
            data = {'port': r.get(id[0]).decode('utf-8')}
            json_data = json.dumps(data)
```

در متد do_GET، تمامی مسیر های مربوط به درخواست های get مشخص شده اند و برای رخ دادن هر کدام، منطق مربوطه پیاده سازی شده است.

```
def do_POST(self):
    content_length = int(self.headers['Content-Length'])

    body = self.rfile.read(content_length)

    # form_data = parse_qs(body.decode())
    if self.path == '/signup':
        data = json.loads(body)
        username = data['username']
        ip = data['ip']
        self.send_response(200)
        self.send_header('Content-type', 'text/plain')
        self.end_headers()
        data = json.loads(body.decode())
        print(data['username'])
        r.set(data['username'], data['port'])
        self.wfile.write('Done'.encode('utf-8'))
        print('{} was signed up with ip: {}'.format(username, ip))
    else:
        # Send a 404 Not Found response
        self.send_response(404)
        self.send_header('Content-type', 'text/plain')
        self.end_headers()
        self.wfile.write(b'404 Not Found')
```

در متد do_POST نیز، تمامی مسیر های مربوط به متد post مشخص و منطق هر کدام تعیین شده است.

Utils.py:

در این فایل برخی تابع ها ی کارآمد نگه داری می‌شود:

- Generate_random_port

  از این تابع برای تولید پورت رندوم برای هر برنامه جدیدی که ران می شود استفاده میکنیم.

- Print_menu

دستورات قابل استفاده در هر حالت برنامه در اینجا تعیین و چاپ می شود.

- Get_random_quote & get_random_image

از این دو تابع برای استخراج متن یا عکس تصادفی برای ارسال به کاربر هنگام درخواست تبادل داده استفاده می شود.

# Peer.py:

در این برنامه از دو ترد استفاده می شود. یکی ترد اصلی و دیگری تردی برای دریافت درخواست هایی که به سمت همتا ارسال می‌شود.

```python
if __name__ == "__main__":

    loop_condition = True
    print('welcome to TMGE service \nplease pick a choice:')
    while loop_condition:
        if LOCK:
            pass
        else:
            utils.print_menu(connected=connected, registered=registered)

            user_input = utils.get_input()
            if registered:
                if not connected:
                    if user_input == '1':
                            getAllUsers()
                    elif user_input == '2':
                            getUserInfo()
                    elif user_input == '3':
                            connected, connected_to = connect_to_peer()
                else:
                        # print(
                        #     """
                        # 1) Request quote
                        # 2) Request Media
```

ترد مین

```python
def listen_for_requests(port=PORT_NUMBER):
    # Create a UDP socket
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    # Bind the socket to a specific port
    server_socket.bind((ip_address, port))

    print(f"UDP listener started on port {port}")

    while True:
        # Receive data from clients
        data, client_address = server_socket.recvfrom(1024)
        data = pickle.loads(data)
        LOCK = True
        if data.type == RequestTypes.CONNECTION:
            print(f"Received request from {client_address[0]}:{client_address[1
            connection_request_handler(server_socket, client_address)
        elif data.type == RequestTypes.QUOTE:
            print(f"Received request from {client_address[0]}:{client_address[1
            request_quote_handler(server_socket, client_address, data)
        elif data.type == RequestTypes.MEDIA:
            print(f"Received request from {client_address[0]}:{client_address[1
            request_media_handler(server_socket, client_address, data)
```

ترد listen

در ادامه، توابع مربوط به هر درخواست را می‌بینید:

```
def connection_request_handler(server_socket, client_address):...

def request_quote_handler(server_socket, client_address, data):...

def split_into_packets(image_data, packet_size):...

def calculate_checksum(data):...

def request_media_handler(server_socket, client_address, data):...
```

و در اینجا نیز، توابع مربوط به ارسال درخواست ها قابل مشاهده هستند:

```
def connect_to_peer():...

def request_quote():...


def validate_checksum(data, received_checksum):...

def reassemble_packets(received_packets, total_packets):...

def request_media():...
```

در ادامه به بررسی تک تک توابع خواهیم پرداخت:

## توابع سمت کاربر:

- Connect_to_peer:

این تابع وظیفه ارسال درخواست اتصال به یک همتای دیگر را دارد و در ادامه منتظر پاسخ می ماند. در صورت قبول، وضعیت برنامه به connected = true تغییر یافته و همتایی که به آن وصل هستیم، ذخیره می شود تا بعدا از پورت آن استفاده کنیم. لازم به ذکر است که اینجا از پورت udp استفاده می‌کنیم.

```
def connect_to_peer():
    port = input("please enter Port number:\n>")
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    message = pickle.dumps(Request(RequestTypes.CONNECTION))
    sock.sendto(message, (ip_address, int(port)))
    print('Connection request was sent succesfully wait for acception...')
    sock.settimeout(15)
    try:
        response, _ = sock.recvfrom(1024)
        print(f"Connection accepted by {ip_address}:{port}")
        return (True, {
            'ip': ip_address,
            'port': port
        })

    except socket.timeout:
        print(f"No response from {ip_address}:{port}. Peer is not online.")
        return (False, None)

    finally:
        # Close the socket
        sock.close()
```

- Request_quote():

این تابع به قصد ارسال درخواست دریافت متن استفاده می شود. و برای متن دریافتی از نقل قول های افراد مشهور استفاده شده است. در این تابع، ابتدا با udp درخواستی ارسال می شود و در ادامه با tcp متن دریافت می‌شود:

```
def request_quote():
    port = connected_to['port']
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    message = pickle.dumps(Request(RequestTypes.QUOTE, data={'port': TCP_PORT_NUMBER}))
    sock.sendto(message, (ip_address, int(port)))
    print('Quote request was sent succesfully wait for acception...')
    sock.close()
    try:
        tcp_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        tcp_socket.connect((ip_address, TCP_RECIEVE_PORT))
        print(f"Connected to {ip_address}:{TCP_RECIEVE_PORT} via TCP")

        try:
            response = tcp_socket.recv(1024)
            print(f"Quote received from {ip_address}:{port}  :\n>{response.decode()}")

        finally:
            tcp_socket.close()

    except Exception as e:
        print(f"An error occurred while connecting to the target peer via TCP: {e}")
```

- Request_media():

در این تابع، درخواست دریافت عکس ارسال می شود و در ادامه آن، عکس نیز دریافت می شود. هر دو عمل ارسال و دریافت با udp پیاده سازی شده است:

```
def request_media():
    port = connected_to['port']
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    message = pickle.dumps(Request(RequestTypes.MEDIA))
    sock.sendto(message, (ip_address, int(port)))
    print('Media request was sent succesfully wait for acception...')
    sock.settimeout(70)
    try:
        image_size_data, addr = sock.recvfrom(1024)
        image_size = tuple(map(int, image_size_data.decode().split(',')))

        image = Image.new('RGB', image_size)

        received_packets = {}
```

```
while len(received_packets) < image_size[0] * image_size[1]:
    # Receive a packet
    packet_data, addr = sock.recvfrom(1024)

    # Split the packet data into individual values
    x, y, r, g, b = map(int, packet_data.decode().split(','))

    # Set the pixel value in the image
    image.putpixel((x, y), (r, g, b))

    # Send an acknowledgment
    sock.sendto(b'ACK', addr)

    # Store the received packet in the dictionary
    received_packets[(x, y)] = True

    # Check for any missing packets and request retransmission
    missing_packets = [(x, y) for x in range(image_size[0]) for y in range(image_size[1]) if
                       (x, y) not in received_packets]
    for packet in missing_packets:
        request_data = f"REQUEST,{packet[0]},{packet[1]}"
        sock.sendto(request_data.encode(), addr)

    # sock.close()
```

# توابع سروری:

این توابع نقش برآورده کردن درخواست ها و ارسال داده را بر عهده دارند

- Connection_request_handler():

  این تابع وظیفه تایید یا رد درخواست اتصال به یک همتا را دارد:

```python
def connection_request_handler(server_socket, client_address):
    req_accept = input('Do you want to accept this request?\n1)Yes\n2)No\n>')
    if req_accept == '1':
        response_message = "Connection accepted"
        server_socket.sendto(response_message.encode(), client_address)
        print(f"Sent connection acceptance to {client_address[0]}:{client_address[1]}")
    elif req_accept == '2':
        response_message = "Connection rejected"
        server_socket.sendto(response_message.encode(), client_address)
        print(f"Sent connection rejection to {client_address[0]}:{client_address[1]}")
    else:
        print('wrong entry!!')
```

- Request_quote_handler():

این تابع وظیفه تایید یا رد درخواست ارسال یک متن را دارد. این تابع، درخواست را با udp دریافت و پاسخ را با tcp ارسال می‌کند(بخش udp آن در تابع listener مدیریت شده است):

```
def request_quote_handler(server_socket, client_address, data):
    tcp_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    tcp_socket.bind((ip_address, TCP_RECIEVE_PORT))
    tcp_socket.listen(1)
    quote_accept = input('Do you want to accept the quote request?\n1) Yes\n2) No\n>')
    if quote_accept == '1':
        response_message = utils.get_random_quote()
        connection_socket, _ = tcp_socket.accept()
        connection_socket.sendall(response_message.encode())
        print(f"Sent a Random quote to {client_address[0]}:{client_address[1]}")
    elif quote_accept == '2':
        response_message = "random quote request rejected"
        server_socket.sendto(response_message.encode(), client_address)
        print(f"Sent random quote request rejection to {client_address[0]}:{client_address[1]}")
    else:
        print('oops. wrong entry!!')
```

- Request_media_handler():

در این تابع، درخواست ارسال یک عکس تایید یا رد می شود. در این تابع هر دو عمل دریافت درخواست و ارسال نتیجه با udp پیاده سازی شده اند. در این تابع، تصویر را به داده های بایتی در می آوریم و در ادامه آن را packet بندی میکنیم و در هر مرحله یک packet را ارسال میکنیم. از طرف کلاینت نیز با داشتن یک حلقه بی نهایت، packet ها را دریافت میکنیم و کنار هم قرار می‌دهیم تا درنهایت عکس خروجی را تولید کنیم.

```
image = Image.open(image_address)

image_size = image.size
image_size_data = f"{image_size[0]},{image_size[1]}"
server_socket.sendto(image_size_data.encode(), client_address)

# Convert the image to RGB mode
image = image.convert('RGB')

# Iterate over the image and send packets
for y in range(image_size[1]):
    for x in range(image_size[0]):
        # Get the pixel RGB values
        r, g, b = image.getpixel((x, y))

        # Create the packet data
        packet_data = f"{x},{y},{r},{g},{b}"

        # Send the packet
        while True:
            print('boi')
            server_socket.sendto(packet_data.encode(), client_address)
            ack, addr = server_socket.recvfrom(1024)
            if ack == b'ACK':
```