# پروژه هفتم :

سیدمحمدمهدی مصطفوی غمام

برای پاسخ به سوالات و پروژه‌های ارائه‌شده در فایل شما، به ترتیب به هر کدام از موارد پرداخته می‌شود. هر پروژه به‌طور جداگانه توضیح داده شده و راهنمایی لازم برای پیاده‌سازی ارائه خواهد شد.

### **پروژه 1: پیاده‌سازی سیستم اشتراک فایل FTP با پایتون**

#### **توضیح:**

شما باید با استفاده از زبان پایتون، یک سیستم اشتراک فایل مبتنی بر پروتکل FTP طراحی کنید. این سیستم باید امکان آپلود و دانلود فایل‌ها را برای کاربران فراهم کند.

#### **راهنمایی:**

1. از کتابخانه‌های استاندارد پایتون مانند ftplib برای پیاده‌سازی استفاده کنید.
2. یک سرور FTP راه‌اندازی کنید که کاربران بتوانند با آن ارتباط برقرار کنند.
3. امکان ورود کاربر (Authentication) را اضافه کنید.
4. قابلیت‌های زیر باید فراهم باشد:
   * آپلود فایل به سرور
   * دانلود فایل از سرور
   * نمایش لیست فایل‌های موجود در سرور
5. برای امنیت بیشتر می‌توانید از پروتکل‌های رمزنگاری مانند TLS/SSL استفاده کنید.

### **پروژه 2: پیاده‌سازی مدل Client-Server برای اجرای دستورات از راه دور**

#### **توضیح:**

یک برنامه Client-Server طراحی کنید که کلاینت بتواند دستورات را به سرور ارسال کند و سرور این دستورات را اجرا کرده و خروجی را به کلاینت بازگرداند.

#### **راهنمایی:**

1. از کتابخانه socket در پایتون برای پیاده‌سازی ارتباط Client-Server استفاده کنید.
2. سرور باید:
   * دستورات ارسالی از کلاینت را دریافت کند.
   * دستور را در سیستم اجرا کند (مثلاً با استفاده از کتابخانه subprocess).
   * خروجی دستور را به کلاینت ارسال کند.
3. کلاینت باید:
   * دستورات را از کاربر دریافت کند.
   * دستور را به سرور ارسال کند.
   * خروجی را از سرور دریافت کرده و نمایش دهد.

### **پروژه 3: پیاده‌سازی رصد لحظه‌ای قیمت ارز (دلار، یورو و …)**

#### **توضیح:**

یک مدل Client-Server طراحی کنید که سرور قیمت لحظه‌ای ارزها (مانند دلار و یورو) را به کلاینت‌ها ارسال کند.

#### **راهنمایی:**

1. سرور باید:
   * قیمت ارزها را از یک منبع آنلاین (مانند APIهای رایگان) دریافت کند.
   * قیمت‌ها را به‌صورت لحظه‌ای به کلاینت‌ها ارسال کند.
2. کلاینت باید:
   * درخواست قیمت ارز خاصی را به سرور ارسال کند.
   * قیمت لحظه‌ای را از سرور دریافت و نمایش دهد.
3. برای دریافت قیمت‌ها می‌توانید از APIهایی مانند [Exchange Rates API](https://exchangeratesapi.io/) استفاده کنید.

### **پروژه 4: ابزار نگهداری کلید و مقدار (Key-Value Store)**

#### **توضیح:**

یک ابزار Client-Server طراحی کنید که بتواند مقادیر را با استفاده از کلید ذخیره و بازیابی کند.

#### **راهنمایی:**

1. از مدل Client-Server و سوکت‌ها استفاده کنید.
2. سرور باید:
   * مقادیر کلید-مقدار را در یک فایل یا پایگاه داده ذخیره کند.
   * امکان اضافه کردن، به‌روزرسانی و حذف مقادیر را فراهم کند.
3. کلاینت باید:
   * درخواست‌های ذخیره، بازیابی و حذف مقادیر را ارسال کند.
   * پاسخ سرور را دریافت و نمایش دهد.

### **پروژه 5: اسکن پورت‌های باز**

#### **توضیح:**

یک برنامه Client-Server طراحی کنید که کلاینت بتواند پورت‌های باز سرور را شناسایی کند.

#### **راهنمایی:**

1. سرور باید:
   * به درخواست‌های کلاینت برای بررسی وضعیت پورت‌ها پاسخ دهد.
2. کلاینت باید:
   * لیستی از پورت‌ها را به سرور ارسال کند.
   * وضعیت پورت‌ها (باز یا بسته) را از سرور دریافت کند.
3. از کتابخانه socket برای این کار استفاده کنید.

### **پروژه 6: تشخیص حملات DDOS**

#### **توضیح:**

یک برنامه Client-Server طراحی کنید که بتواند حملات DDOS را شناسایی کند.

#### **راهنمایی:**

1. سرور باید:
   * تعداد درخواست‌های ورودی را مانیتور کند.
   * در صورت تشخیص تعداد زیاد درخواست‌های غیرعادی، ارتباطات مشکوک را مسدود کند.
2. از الگوریتم‌های تشخیص ساده مانند بررسی تعداد درخواست‌ها در یک بازه زمانی مشخص استفاده کنید.
3. کلاینت باید:
   * درخواست‌های عادی یا غیرعادی را ارسال کند.
   * پاسخ سرور را دریافت کند.

### **پروژه 7: چت روم ایمن با TLS/SSL**

#### **توضیح:**

یک چت روم ایمن طراحی کنید که از پروتکل‌های TLS/SSL برای رمزنگاری استفاده کند.

#### **راهنمایی:**

1. از کتابخانه ssl در پایتون استفاده کنید.
2. سرور باید:
   * پیام‌های کلاینت‌ها را دریافت کرده و به سایر کلاینت‌ها ارسال کند.
   * ارتباطات را رمزنگاری کند.
3. کلاینت باید:
   * پیام‌ها را ارسال و دریافت کند.
   * ارتباط ایمن با سرور برقرار کند.

### **پروژه 8: پیاده‌سازی سرور DNS**

#### **توضیح:**

یک سرور DNS طراحی کنید که بتواند IP سایت‌های معروف را بازگرداند.

#### **راهنمایی:**

1. سرور باید:
   * لیستی از سایت‌های معروف و IPهای آن‌ها را داشته باشد.
   * درخواست‌های کلاینت را پردازش کرده و IP مربوطه را بازگرداند.
2. کلاینت باید:
   * نام سایت را به سرور ارسال کند.
   * IP سایت را از سرور دریافت کند.

### **پروژه 9: طراحی سرور DHCP ساده**

#### **توضیح:**

یک سرور DHCP طراحی کنید که بتواند IPها را به کلاینت‌ها اختصاص دهد.

#### **راهنمایی:**

1. سرور باید:
   * یک Pool از IPها داشته باشد.
   * IPها را بر اساس فرآیند DORA (Discover, Offer, Request, Acknowledge) به کلاینت‌ها اختصاص دهد.
2. کلاینت باید:
   * درخواست IP را به سرور ارسال کند.
   * IP اختصاص داده شده را دریافت کند.

### **پروژه 10: بازی Tic Tac Toe با مدل Client-Server**

#### **توضیح:**

یک بازی Tic Tac Toe طراحی کنید که دو کاربر به‌صورت همزمان بتوانند بازی کنند.

#### **راهنمایی:**

1. سرور باید:
   * ارتباط دو کلاینت را مدیریت کند.
   * وضعیت بازی را ذخیره کند.
   * برنده را مشخص کند.
2. کلاینت‌ها باید:
   * حرکات خود را به سرور ارسال کنند.
   * وضعیت بازی را از سرور دریافت کنند.
3. در صورتی که تعداد کلاینت‌ها بیشتر از دو نفر باشد، آن‌ها را در یک صف (Queue) قرار دهید.

### **نتیجه‌گیری:**

هر پروژه نیازمند آشنایی با مفاهیم پایه شبکه، سوکت‌ها و برنامه‌نویسی پایتون است. برای هر پروژه، از کتابخانه‌های استاندارد پایتون مانند socket، ssl و سایر ابزارهای مرتبط استفاده کنید. اگر به توضیحات بیشتری برای هر پروژه نیاز دارید، اطلاع دهید تا جزئیات بیشتری ارائه شود.

## **پروژه 1: پیاده‌سازی سیستم اشتراک فایل FTP با پایتون**

### **توضیح پروژه:**

سیستم اشتراک فایل FTP به کاربران اجازه می‌دهد فایل‌ها را روی یک سرور آپلود یا از آن دانلود کنند. در این پروژه، از پروتکل FTP استفاده می‌کنیم و یک سرور ساده با قابلیت آپلود و دانلود فایل پیاده‌سازی می‌کنیم.

### **پیاده‌سازی:**

#### **1. نصب کتابخانه مورد نیاز**

برای این پروژه از کتابخانه pyftpdlib استفاده می‌کنیم. این کتابخانه به شما اجازه می‌دهد تا یک سرور FTP ساده راه‌اندازی کنید.

ابتدا کتابخانه را نصب کنید:

pip install pyftpdlib

#### **2. کد سرور FTP**

کد زیر یک سرور FTP ساده را راه‌اندازی می‌کند که کاربران می‌توانند به آن متصل شوند و فایل‌ها را آپلود یا دانلود کنند.

from pyftpdlib.authorizers import DummyAuthorizer

from pyftpdlib.handlers import FTPHandler

from pyftpdlib.servers import FTPServer

def run\_ftp\_server():

# ایجاد مجوزها برای کاربران

authorizer = DummyAuthorizer()

# افزودن کاربر با دسترسی کامل (خواندن و نوشتن)

# username: user | password: 12345

authorizer.add\_user("user", "12345", "./ftp\_files", perm="elradfmw")

# کاربر ناشناس فقط دسترسی خواندن دارد

authorizer.add\_anonymous("./ftp\_files", perm="elr")

# تنظیمات هندلر

handler = FTPHandler

handler.authorizer = authorizer

# راه‌اندازی سرور

server = FTPServer(("127.0.0.1", 2121), handler)

print("FTP Server is running on 127.0.0.1:2121...")

server.serve\_forever()

if \_\_name\_\_ == "\_\_main\_\_":

run\_ftp\_server()

#### **3. توضیحات کد:**

1. **ایجاد مجوزها:**
   * از کلاس DummyAuthorizer برای مدیریت کاربران استفاده می‌کنیم.
   * یک کاربر با نام کاربری user و رمز عبور 12345 ایجاد شده است که به پوشه ./ftp\_files دسترسی کامل دارد.
   * کاربر ناشناس (بدون نیاز به ورود) فقط اجازه خواندن فایل‌ها را دارد.
2. **هندلر FTP:**
   * FTPHandler مسئول مدیریت درخواست‌های FTP است.
3. **راه‌اندازی سرور:**
   * سرور روی آدرس 127.0.0.1 و پورت 2121 اجرا می‌شود.
   * پوشه ./ftp\_files باید از قبل ایجاد شود و فایل‌ها در این پوشه ذخیره می‌شوند.

#### **4. اجرای سرور:**

فایل را اجرا کنید:  
 bash

python ftp\_server.py

* سرور روی 127.0.0.1:2121 اجرا می‌شود.

#### **5. اتصال به سرور:**

برای اتصال به سرور FTP می‌توانید از ابزارهای زیر استفاده کنید:

* **مرورگر فایل (File Explorer):**
  + آدرس ftp://127.0.0.1:2121 را وارد کنید.
  + نام کاربری: user
  + رمز عبور: 12345

**ابزار خط فرمان (Command Line):** در خط فرمان دستور زیر را وارد کنید:  
  
 bash

ftp 127.0.0.1 2121

* سپس نام کاربری و رمز عبور را وارد کنید.
* **کتابخانه پایتون (ftplib):** می‌توانید از کد زیر برای اتصال به سرور و آپلود/دانلود فایل استفاده کنید:

python

from ftplib import FTP

# اتصال به سرور

ftp = FTP()

ftp.connect("127.0.0.1", 2121)

ftp.login("user", "12345")

# لیست فایل‌ها در سرور

print("Files on server:")

ftp.retrlines('LIST')

# آپلود فایل

filename = "upload\_test.txt"

with open(filename, "rb") as file:

ftp.storbinary(f"STOR {filename}", file)

print(f"File '{filename}' uploaded successfully!")

# دانلود فایل

download\_file = "download\_test.txt"

with open(download\_file, "wb") as file:

ftp.retrbinary(f"RETR {download\_file}", file.write)

print(f"File '{download\_file}' downloaded successfully!")

# بستن اتصال

ftp.quit()

### **نتیجه:**

* سرور FTP شما آماده است و می‌توانید فایل‌ها را آپلود یا دانلود کنید.
* این سیستم ساده است و می‌توانید ویژگی‌های بیشتری مانند رمزنگاری TLS/SSL یا مدیریت کاربران پیشرفته به آن اضافه کنید.

### **پروژه 2: پیاده‌سازی مدل Client-Server برای اجرای دستورات از راه دور**

### **توضیح پروژه:**

در این پروژه، یک مدل Client-Server طراحی می‌کنیم که در آن کلاینت می‌تواند دستورات را به سرور ارسال کند. سرور این دستورات را اجرا کرده و خروجی را به کلاینت بازمی‌گرداند. این پروژه از سوکت‌ها برای برقراری ارتباط بین کلاینت و سرور استفاده می‌کند.

### **پیاده‌سازی:**

#### **1. کد سرور**

سرور به درخواست کلاینت گوش می‌دهد، دستورات را دریافت می‌کند، آن‌ها را اجرا می‌کند و خروجی را به کلاینت بازمی‌گرداند.

python

import socket

import subprocess

def run\_server():

# آدرس و پورت سرور

HOST = '127.0.0.1'

PORT = 12345

# ایجاد سوکت سرور

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.bind((HOST, PORT))

server\_socket.listen(1)

print(f"Server is running on {HOST}:{PORT}...")

# انتظار برای اتصال کلاینت

conn, addr = server\_socket.accept()

print(f"Connected by {addr}")

while True:

# دریافت دستور از کلاینت

command = conn.recv(1024).decode()

if not command or command.lower() == "exit":

print("Connection closed by client.")

break

print(f"Received command: {command}")

# اجرای دستور با استفاده از subprocess

try:

output = subprocess.check\_output(command, shell=True, stderr=subprocess.STDOUT, text=True)

except subprocess.CalledProcessError as e:

output = f"Error:\n{e.output}"

# ارسال خروجی به کلاینت

conn.send(output.encode())

conn.close()

server\_socket.close()

if \_\_name\_\_ == "\_\_main\_\_":

run\_server()

#### **2. کد کلاینت**

کلاینت دستورات را از کاربر دریافت می‌کند، آن‌ها را به سرور ارسال می‌کند و خروجی را دریافت و نمایش می‌دهد.

python

import socket

def run\_client():

# آدرس و پورت سرور

HOST = '127.0.0.1'

PORT = 12345

# ایجاد سوکت کلاینت

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client\_socket.connect((HOST, PORT))

print(f"Connected to server {HOST}:{PORT}")

while True:

# دریافت دستور از کاربر

command = input("Enter command to execute (or 'exit' to quit): ")

if command.lower() == "exit":

client\_socket.send(command.encode())

print("Exiting...")

break

# ارسال دستور به سرور

client\_socket.send(command.encode())

# دریافت خروجی از سرور

output = client\_socket.recv(4096).decode()

print(f"Output:\n{output}")

client\_socket.close()

if \_\_name\_\_ == "\_\_main\_\_":

run\_client()

### **توضیحات کد:**

#### **سرور:**

1. **ایجاد سوکت سرور:**
   * سرور یک سوکت TCP با آدرس 127.0.0.1 و پورت 12345 ایجاد می‌کند.
   * از متد bind برای اتصال به آدرس و پورت استفاده می‌شود.
   * از متد listen برای انتظار اتصال کلاینت استفاده می‌شود.
2. **دریافت دستورات:**
   * سرور دستورات را از کلاینت دریافت می‌کند.
   * اگر دستور exit دریافت شود، ارتباط را قطع می‌کند.
3. **اجرای دستورات:**
   * از کتابخانه subprocess برای اجرای دستورات سیستم استفاده می‌شود.
   * خروجی دستور (یا خطا) به کلاینت ارسال می‌شود.

#### **کلاینت:**

1. **ایجاد سوکت کلاینت:**
   * کلاینت به سرور متصل می‌شود.
   * از متد connect برای اتصال به آدرس و پورت سرور استفاده می‌شود.
2. **ارسال دستورات:**
   * کلاینت دستورات را از کاربر دریافت می‌کند و به سرور ارسال می‌کند.
3. **دریافت خروجی:**
   * کلاینت خروجی دستور را از سرور دریافت کرده و نمایش می‌دهد.

### **اجرای پروژه:**

#### **1. اجرای سرور:**

ابتدا سرور را اجرا کنید:

bash

python server.py

سرور آماده است و به درخواست‌های کلاینت گوش می‌دهد.

#### **2. اجرای کلاینت:**

سپس کلاینت را اجرا کنید:

bash

python client.py

کلاینت به سرور متصل می‌شود و شما می‌توانید دستورات را وارد کنید.

#### **3. تست دستورات:**

* در کلاینت دستورات زیر را امتحان کنید:
  + ls (در لینوکس) یا dir (در ویندوز): نمایش فایل‌های موجود در پوشه فعلی.
  + pwd (در لینوکس) یا cd (در ویندوز): نمایش مسیر فعلی.
  + هر دستور دیگری که توسط سیستم شما پشتیبانی می‌شود.

### **نمونه اجرا:**

#### **سرور:**

bash

Server is running on 127.0.0.1:12345...

Connected by ('127.0.0.1', 54321)

Received command: ls

Received command: pwd

Connection closed by client.

#### **کلاینت:**

bash

Connected to server 127.0.0.1:12345

Enter command to execute (or 'exit' to quit): ls

Output:

file1.txt

file2.txt

Enter command to execute (or 'exit' to quit): pwd

Output:

/home/user/project

Enter command to execute (or 'exit' to quit): exit

Exiting...

### **نکات امنیتی:**

1. اجرای دستورات از راه دور ممکن است خطرناک باشد، زیرا کاربران می‌توانند دستورات مخرب ارسال کنند. برای افزایش امنیت:
   * دستورات مجاز را محدود کنید.
   * دسترسی کاربران را کنترل کنید.
   * از رمزنگاری (مانند TLS/SSL) برای ارتباطات استفاده کنید.

### **نتیجه:**

این پروژه یک سیستم ساده برای اجرای دستورات از راه دور است. با این حال، می‌توانید قابلیت‌های بیشتری مانند احراز هویت کاربران، رمزنگاری ارتباطات، و مدیریت دستورات مجاز را به آن اضافه کنید.

### **پروژه 3: پیاده‌سازی رصد لحظه‌ای قیمت ارز (دلار، یورو و …)**

### **توضیح پروژه:**

در این پروژه، یک سیستم Client-Server طراحی می‌کنیم که سرور قیمت لحظه‌ای ارزها (مانند دلار و یورو) را از یک API دریافت کرده و به کلاینت ارسال می‌کند. کلاینت می‌تواند درخواست قیمت ارز خاصی را به سرور ارسال کند و قیمت لحظه‌ای آن را دریافت کند.

### **پیاده‌سازی:**

#### **1. استفاده از API برای دریافت قیمت ارز**

برای دریافت قیمت ارزها از یک API رایگان استفاده می‌کنیم. یکی از APIهای مناسب، [Exchange Rates API](https://exchangeratesapi.io/) یا [Open Exchange Rates](https://openexchangerates.org/) است. برای این مثال، از یک API رایگان استفاده می‌کنیم.

ابتدا کتابخانه requests را نصب کنید (برای ارتباط با API):

bash

pip install requests

#### **2. کد سرور**

سرور به درخواست‌های کلاینت گوش می‌دهد، قیمت ارزها را از API دریافت می‌کند و آن را به کلاینت ارسال می‌کند.

python

import socket

import requests

def get\_exchange\_rate(currency):

"""

دریافت قیمت لحظه‌ای ارز از API

"""

API\_URL = f"https://api.exchangerate-api.com/v4/latest/USD" # نرخ ارزها بر اساس دلار

try:

response = requests.get(API\_URL)

data = response.json()

rate = data['rates'].get(currency.upper())

if rate:

return f"1 USD = {rate} {currency.upper()}"

else:

return f"Currency '{currency}' not found."

except Exception as e:

return f"Error fetching exchange rate: {e}"

def run\_server():

# آدرس و پورت سرور

HOST = '127.0.0.1'

PORT = 12345

# ایجاد سوکت سرور

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.bind((HOST, PORT))

server\_socket.listen(1)

print(f"Server is running on {HOST}:{PORT}...")

# انتظار برای اتصال کلاینت

conn, addr = server\_socket.accept()

print(f"Connected by {addr}")

while True:

# دریافت درخواست ارز از کلاینت

currency = conn.recv(1024).decode()

if not currency or currency.lower() == "exit":

print("Connection closed by client.")

break

print(f"Received request for currency: {currency}")

# دریافت نرخ ارز

exchange\_rate = get\_exchange\_rate(currency)

# ارسال نرخ ارز به کلاینت

conn.send(exchange\_rate.encode())

conn.close()

server\_socket.close()

if \_\_name\_\_ == "\_\_main\_\_":

run\_server()

#### **3. کد کلاینت**

کلاینت نام ارز (مانند EUR برای یورو یا GBP برای پوند) را از کاربر دریافت می‌کند، آن را به سرور ارسال می‌کند و نرخ ارز را دریافت و نمایش می‌دهد.

python

import socket

def run\_client():

# آدرس و پورت سرور

HOST = '127.0.0.1'

PORT = 12345

# ایجاد سوکت کلاینت

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client\_socket.connect((HOST, PORT))

print(f"Connected to server {HOST}:{PORT}")

while True:

# دریافت نام ارز از کاربر

currency = input("Enter currency code (e.g., USD, EUR, GBP) or 'exit' to quit: ")

if currency.lower() == "exit":

client\_socket.send(currency.encode())

print("Exiting...")

break

# ارسال درخواست ارز به سرور

client\_socket.send(currency.encode())

# دریافت نرخ ارز از سرور

exchange\_rate = client\_socket.recv(1024).decode()

print(f"Exchange Rate: {exchange\_rate}")

client\_socket.close()

if \_\_name\_\_ == "\_\_main\_\_":

run\_client()

### **توضیحات کد:**

#### **سرور:**

1. **دریافت نرخ ارز:**
   * سرور از API https://api.exchangerate-api.com/v4/latest/USD استفاده می‌کند تا نرخ ارزها را بر اساس دلار دریافت کند.
   * ارز مورد نظر کلاینت (مانند EUR یا GBP) را از داده‌های API استخراج می‌کند.
2. **مدیریت ارتباط:**
   * سرور به درخواست‌های کلاینت گوش می‌دهد.
   * اگر کلاینت درخواست exit ارسال کند، سرور ارتباط را می‌بندد.

#### **کلاینت:**

1. **ارسال درخواست:**
   * کلاینت کد ارز (مانند EUR یا GBP) را از کاربر دریافت می‌کند و به سرور ارسال می‌کند.
2. **دریافت پاسخ:**
   * کلاینت نرخ ارز را از سرور دریافت کرده و نمایش می‌دهد.

### **اجرای پروژه:**

#### **1. اجرای سرور:**

ابتدا سرور را اجرا کنید:

bash

python server.py

سرور آماده است و به درخواست‌های کلاینت گوش می‌دهد.

#### **2. اجرای کلاینت:**

سپس کلاینت را اجرا کنید:

bash

python client.py

کلاینت به سرور متصل می‌شود و شما می‌توانید نام ارزها را وارد کنید.

#### **3. تست درخواست‌ها:**

* در کلاینت کد ارزهای مختلف را وارد کنید، مثلاً:
  + EUR (یورو)
  + GBP (پوند)
  + JPY (ین ژاپن)
  + هر ارز دیگر که توسط API پشتیبانی می‌شود.

### **نمونه اجرا:**

#### **سرور:**

bash

Server is running on 127.0.0.1:12345...

Connected by ('127.0.0.1', 54321)

Received request for currency: EUR

Received request for currency: GBP

Connection closed by client.

#### **کلاینت:**

bash

Connected to server 127.0.0.1:12345

Enter currency code (e.g., USD, EUR, GBP) or 'exit' to quit: EUR

Exchange Rate: 1 USD = 0.85 EUR

Enter currency code (e.g., USD, EUR, GBP) or 'exit' to quit: GBP

Exchange Rate: 1 USD = 0.75 GBP

Enter currency code (e.g., USD, EUR, GBP) or 'exit' to quit: exit

Exiting...

### **نکات:**

1. **API رایگان:**
   * API مورد استفاده ممکن است محدودیت‌هایی داشته باشد (مانند تعداد درخواست‌ها در هر ساعت). برای استفاده بیشتر، می‌توانید از APIهای پولی استفاده کنید.
   * اگر API دیگری استفاده می‌کنید، ممکن است نیاز باشد کد را کمی تغییر دهید.
2. **مدیریت خطا:**
   * اگر ارز وارد شده توسط کاربر نامعتبر باشد، سرور پیام خطا ارسال می‌کند.
   * اگر اتصال به API با خطا مواجه شود، سرور پیام خطای مناسب ارسال می‌کند.

### **نتیجه:**

این سیستم به شما اجازه می‌دهد نرخ لحظه‌ای ارزها را از طریق یک مدل Client-Server مشاهده کنید. می‌توانید قابلیت‌های بیشتری مانند رمزنگاری ارتباطات، مدیریت کاربران، یا پشتیبانی از چندین کلاینت را به این پروژه اضافه کنید.

### **پروژه 3: پیاده‌سازی رصد لحظه‌ای قیمت ارز (دلار، یورو و …)**

### **توضیح پروژه:**

در این پروژه، یک سیستم Client-Server طراحی می‌کنیم که سرور قیمت لحظه‌ای ارزها (مانند دلار و یورو) را از یک API دریافت کرده و به کلاینت ارسال می‌کند. کلاینت می‌تواند درخواست قیمت ارز خاصی را به سرور ارسال کند و قیمت لحظه‌ای آن را دریافت کند.

### **پیاده‌سازی:**

#### **1. استفاده از API برای دریافت قیمت ارز**

برای دریافت قیمت ارزها از یک API رایگان استفاده می‌کنیم. یکی از APIهای مناسب، [Exchange Rates API](https://exchangeratesapi.io/) یا [Open Exchange Rates](https://openexchangerates.org/) است. برای این مثال، از یک API رایگان استفاده می‌کنیم.

ابتدا کتابخانه requests را نصب کنید (برای ارتباط با API):

bash

pip install requests

#### **2. کد سرور**

سرور به درخواست‌های کلاینت گوش می‌دهد، قیمت ارزها را از API دریافت می‌کند و آن را به کلاینت ارسال می‌کند.

python

import socket

import requests

def get\_exchange\_rate(currency):

"""

دریافت قیمت لحظه‌ای ارز از API

"""

API\_URL = f"https://api.exchangerate-api.com/v4/latest/USD" # نرخ ارزها بر اساس دلار

try:

response = requests.get(API\_URL)

data = response.json()

rate = data['rates'].get(currency.upper())

if rate:

return f"1 USD = {rate} {currency.upper()}"

else:

return f"Currency '{currency}' not found."

except Exception as e:

return f"Error fetching exchange rate: {e}"

def run\_server():

# آدرس و پورت سرور

HOST = '127.0.0.1'

PORT = 12345

# ایجاد سوکت سرور

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.bind((HOST, PORT))

server\_socket.listen(1)

print(f"Server is running on {HOST}:{PORT}...")

# انتظار برای اتصال کلاینت

conn, addr = server\_socket.accept()

print(f"Connected by {addr}")

while True:

# دریافت درخواست ارز از کلاینت

currency = conn.recv(1024).decode()

if not currency or currency.lower() == "exit":

print("Connection closed by client.")

break

print(f"Received request for currency: {currency}")

# دریافت نرخ ارز

exchange\_rate = get\_exchange\_rate(currency)

# ارسال نرخ ارز به کلاینت

conn.send(exchange\_rate.encode())

conn.close()

server\_socket.close()

if \_\_name\_\_ == "\_\_main\_\_":

run\_server()

#### **3. کد کلاینت**

کلاینت نام ارز (مانند EUR برای یورو یا GBP برای پوند) را از کاربر دریافت می‌کند، آن را به سرور ارسال می‌کند و نرخ ارز را دریافت و نمایش می‌دهد.

python

import socket

def run\_client():

# آدرس و پورت سرور

HOST = '127.0.0.1'

PORT = 12345

# ایجاد سوکت کلاینت

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client\_socket.connect((HOST, PORT))

print(f"Connected to server {HOST}:{PORT}")

while True:

# دریافت نام ارز از کاربر

currency = input("Enter currency code (e.g., USD, EUR, GBP) or 'exit' to quit: ")

if currency.lower() == "exit":

client\_socket.send(currency.encode())

print("Exiting...")

break

# ارسال درخواست ارز به سرور

client\_socket.send(currency.encode())

# دریافت نرخ ارز از سرور

exchange\_rate = client\_socket.recv(1024).decode()

print(f"Exchange Rate: {exchange\_rate}")

client\_socket.close()

if \_\_name\_\_ == "\_\_main\_\_":

run\_client()

### **توضیحات کد:**

#### **سرور:**

1. **دریافت نرخ ارز:**
   * سرور از API https://api.exchangerate-api.com/v4/latest/USD استفاده می‌کند تا نرخ ارزها را بر اساس دلار دریافت کند.
   * ارز مورد نظر کلاینت (مانند EUR یا GBP) را از داده‌های API استخراج می‌کند.
2. **مدیریت ارتباط:**
   * سرور به درخواست‌های کلاینت گوش می‌دهد.
   * اگر کلاینت درخواست exit ارسال کند، سرور ارتباط را می‌بندد.

#### **کلاینت:**

1. **ارسال درخواست:**
   * کلاینت کد ارز (مانند EUR یا GBP) را از کاربر دریافت می‌کند و به سرور ارسال می‌کند.
2. **دریافت پاسخ:**
   * کلاینت نرخ ارز را از سرور دریافت کرده و نمایش می‌دهد.

### **اجرای پروژه:**

#### **1. اجرای سرور:**

ابتدا سرور را اجرا کنید:

bash

python server.py

سرور آماده است و به درخواست‌های کلاینت گوش می‌دهد.

#### **2. اجرای کلاینت:**

سپس کلاینت را اجرا کنید:

bash

python client.py

کلاینت به سرور متصل می‌شود و شما می‌توانید نام ارزها را وارد کنید.

#### **3. تست درخواست‌ها:**

* در کلاینت کد ارزهای مختلف را وارد کنید، مثلاً:
  + EUR (یورو)
  + GBP (پوند)
  + JPY (ین ژاپن)
  + هر ارز دیگر که توسط API پشتیبانی می‌شود.

### **نمونه اجرا:**

#### **سرور:**

bash

Server is running on 127.0.0.1:12345...

Connected by ('127.0.0.1', 54321)

Received request for currency: EUR

Received request for currency: GBP

Connection closed by client.

#### **کلاینت:**

bash

Connected to server 127.0.0.1:12345

Enter currency code (e.g., USD, EUR, GBP) or 'exit' to quit: EUR

Exchange Rate: 1 USD = 0.85 EUR

Enter currency code (e.g., USD, EUR, GBP) or 'exit' to quit: GBP

Exchange Rate: 1 USD = 0.75 GBP

Enter currency code (e.g., USD, EUR, GBP) or 'exit' to quit: exit

Exiting...

### **نکات:**

1. **API رایگان:**
   * API مورد استفاده ممکن است محدودیت‌هایی داشته باشد (مانند تعداد درخواست‌ها در هر ساعت). برای استفاده بیشتر، می‌توانید از APIهای پولی استفاده کنید.
   * اگر API دیگری استفاده می‌کنید، ممکن است نیاز باشد کد را کمی تغییر دهید.
2. **مدیریت خطا:**
   * اگر ارز وارد شده توسط کاربر نامعتبر باشد، سرور پیام خطا ارسال می‌کند.
   * اگر اتصال به API با خطا مواجه شود، سرور پیام خطای مناسب ارسال می‌کند.

### **نتیجه:**

این سیستم به شما اجازه می‌دهد نرخ لحظه‌ای ارزها را از طریق یک مدل Client-Server مشاهده کنید. می‌توانید قابلیت‌های بیشتری مانند رمزنگاری ارتباطات، مدیریت کاربران، یا پشتیبانی از چندین کلاینت را به این پروژه اضافه کنید.

### **پروژه 4: ابزار نگهداری کلید و مقدار (Key-Value Store)**

### **توضیح پروژه:**

در این پروژه، یک سیستم Client-Server طراحی می‌کنیم که به کاربران اجازه می‌دهد مقادیر را با استفاده از کلید ذخیره، به‌روزرسانی، بازیابی یا حذف کنند. این سیستم شبیه یک پایگاه داده ساده Key-Value عمل می‌کند.

### **پیاده‌سازی:**

#### **1. کد سرور**

سرور درخواست‌های کلاینت را دریافت می‌کند و بر اساس آن‌ها عملیات ذخیره‌سازی، بازیابی، به‌روزرسانی یا حذف را انجام می‌دهد.

python

import socket

import json

# یک دیکشنری برای ذخیره مقادیر کلید-مقدار

key\_value\_store = {}

def process\_request(request):

"""

پردازش درخواست کلاینت و اجرای عملیات

"""

try:

# تبدیل درخواست از JSON به دیکشنری

data = json.loads(request)

# بررسی نوع عملیات

operation = data.get("operation")

key = data.get("key")

value = data.get("value")

if operation == "set":

# ذخیره یا به‌روزرسانی مقدار

key\_value\_store[key] = value

return f"Key '{key}' set to '{value}'."

elif operation == "get":

# بازیابی مقدار

if key in key\_value\_store:

return f"Value for key '{key}': {key\_value\_store[key]}"

else:

return f"Key '{key}' not found."

elif operation == "delete":

# حذف مقدار

if key in key\_value\_store:

del key\_value\_store[key]

return f"Key '{key}' deleted."

else:

return f"Key '{key}' not found."

elif operation == "list":

# لیست کل مقادیر

return f"Current store: {key\_value\_store}"

else:

return "Invalid operation. Use 'set', 'get', 'delete', or 'list'."

except Exception as e:

return f"Error processing request: {e}"

def run\_server():

# آدرس و پورت سرور

HOST = '127.0.0.1'

PORT = 12345

# ایجاد سوکت سرور

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.bind((HOST, PORT))

server\_socket.listen(1)

print(f"Server is running on {HOST}:{PORT}...")

# انتظار برای اتصال کلاینت

conn, addr = server\_socket.accept()

print(f"Connected by {addr}")

while True:

# دریافت درخواست از کلاینت

request = conn.recv(1024).decode()

if not request or request.lower() == "exit":

print("Connection closed by client.")

break

print(f"Received request: {request}")

# پردازش درخواست

response = process\_request(request)

# ارسال پاسخ به کلاینت

conn.send(response.encode())

conn.close()

server\_socket.close()

if \_\_name\_\_ == "\_\_main\_\_":

run\_server()

#### **2. کد کلاینت**

کلاینت درخواست‌های مربوط به ذخیره‌سازی، بازیابی، به‌روزرسانی یا حذف مقادیر را به سرور ارسال می‌کند.

python

import socket

import json

def run\_client():

# آدرس و پورت سرور

HOST = '127.0.0.1'

PORT = 12345

# ایجاد سوکت کلاینت

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client\_socket.connect((HOST, PORT))

print(f"Connected to server {HOST}:{PORT}")

while True:

# دریافت نوع عملیات از کاربر

operation = input("Enter operation (set, get, delete, list, exit): ").lower()

if operation == "exit":

client\_socket.send(operation.encode())

print("Exiting...")

break

# دریافت کلید و مقدار (در صورت نیاز)

key = input("Enter key: ") if operation in ["set", "get", "delete"] else None

value = input("Enter value: ") if operation == "set" else None

# ساخت درخواست

request = {

"operation": operation,

"key": key,

"value": value

}

# ارسال درخواست به سرور

client\_socket.send(json.dumps(request).encode())

# دریافت پاسخ از سرور

response = client\_socket.recv(1024).decode()

print(f"Response: {response}")

client\_socket.close()

if \_\_name\_\_ == "\_\_main\_\_":

run\_client()

### **توضیحات کد:**

#### **سرور:**

1. **ذخیره‌سازی کلید-مقدار:**
   * مقادیر کلید-مقدار در یک دیکشنری (key\_value\_store) ذخیره می‌شوند.
   * عملیات set یک مقدار جدید را اضافه یا مقدار موجود را به‌روزرسانی می‌کند.
   * عملیات get مقدار مربوط به یک کلید را بازمی‌گرداند.
   * عملیات delete یک کلید را حذف می‌کند.
   * عملیات list کل مقادیر موجود را نمایش می‌دهد.
2. **پردازش درخواست:**
   * درخواست‌های کلاینت به صورت JSON دریافت می‌شوند و به دیکشنری تبدیل می‌شوند.
   * بر اساس نوع عملیات، پاسخ مناسب تولید می‌شود.

#### **کلاینت:**

1. **ارسال درخواست:**
   * کلاینت نوع عملیات (set, get, delete, list) را از کاربر دریافت می‌کند.
   * در صورت نیاز، کلید و مقدار را نیز دریافت می‌کند.
   * درخواست به صورت JSON به سرور ارسال می‌شود.
2. **دریافت پاسخ:**
   * کلاینت پاسخ سرور را دریافت کرده و نمایش می‌دهد.

### **اجرای پروژه:**

#### **1. اجرای سرور:**

ابتدا سرور را اجرا کنید:

bash

python server.py

سرور آماده است و به درخواست‌های کلاینت گوش می‌دهد.

#### **2. اجرای کلاینت:**

سپس کلاینت را اجرا کنید:

bash

python client.py

کلاینت به سرور متصل می‌شود و شما می‌توانید عملیات مختلف را انجام دهید.

#### **3. تست عملیات:**

* **ذخیره مقدار:**

در کلاینت:  
 bash

Enter operation (set, get, delete, list, exit): set

Enter key: name

Enter value: John

پاسخ سرور:  
 bash

Response: Key 'name' set to 'John'.

* **بازیابی مقدار:**

در کلاینت:  
 bash

Enter operation (set, get, delete, list, exit): get

Enter key: name

پاسخ سرور:  
 bash

Response: Value for key 'name': John

* **حذف مقدار:**

در کلاینت:  
 bash

Enter operation (set, get, delete, list, exit): delete

Enter key: name

پاسخ سرور:  
 bash

Response: Key 'name' deleted.

* **نمایش کل مقادیر:**

در کلاینت:  
 bash

Enter operation (set, get, delete, list, exit): list

پاسخ سرور:  
 bash

Response: Current store: {}

### **نکات:**

1. **ذخیره‌سازی پایدار:**
   * در حال حاضر، مقادیر فقط در حافظه ذخیره می‌شوند. برای ذخیره‌سازی پایدار، می‌توانید از یک فایل یا پایگاه داده استفاده کنید.
2. **مدیریت چند کلاینت:**
   * این سرور فقط یک کلاینت را پشتیبانی می‌کند. برای پشتیبانی از چند کلاینت، می‌توانید از threading یا asyncio استفاده کنید.
3. **امنیت:**
   * در این پروژه، هیچ مکانیزم امنیتی برای احراز هویت کاربران یا رمزنگاری داده‌ها وجود ندارد. برای افزایش امنیت، می‌توانید از TLS/SSL استفاده کنید.

### **نتیجه:**

این پروژه یک سیستم ساده Key-Value Store است که می‌تواند مقادیر را ذخیره، بازیابی، به‌روزرسانی یا حذف کند. این سیستم می‌تواند به عنوان پایه‌ای برای یک پایگاه داده توزیع‌شده یا یک سرویس ذخیره‌سازی داده استفاده شود.

### **پروژه 5: اسکن پورت‌های باز**

### **توضیح پروژه:**

در این پروژه، یک سیستم Client-Server طراحی می‌کنیم که کلاینت بتواند پورت‌های باز سرور را شناسایی کند. سرور به درخواست کلاینت گوش می‌دهد و وضعیت پورت‌های مشخص‌شده (باز یا بسته) را به کلاینت گزارش می‌دهد.

### **پیاده‌سازی:**

#### **1. کد سرور**

سرور به درخواست‌های کلاینت گوش می‌دهد و اطلاعات مربوط به وضعیت پورت‌ها را بازمی‌گرداند.

python

import socket

def check\_port(host, port):

"""

بررسی وضعیت پورت (باز یا بسته)

"""

try:

with socket.socket(socket.AF\_INET, socket.SOCK\_STREAM) as s:

s.settimeout(1) # تنظیم زمان انتظار

result = s.connect\_ex((host, port))

if result == 0:

return True # پورت باز است

else:

return False # پورت بسته است

except Exception as e:

return False

def run\_server():

# آدرس و پورت سرور

HOST = '127.0.0.1'

PORT = 12345

# ایجاد سوکت سرور

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.bind((HOST, PORT))

server\_socket.listen(1)

print(f"Server is running on {HOST}:{PORT}...")

# انتظار برای اتصال کلاینت

conn, addr = server\_socket.accept()

print(f"Connected by {addr}")

while True:

# دریافت لیست پورت‌ها از کلاینت

data = conn.recv(1024).decode()

if not data or data.lower() == "exit":

print("Connection closed by client.")

break

print(f"Received request: {data}")

# پردازش لیست پورت‌ها

ports = map(int, data.split(","))

results = {}

for port in ports:

results[port] = check\_port(HOST, port)

# ارسال نتایج به کلاینت

conn.send(str(results).encode())

conn.close()

server\_socket.close()

if \_\_name\_\_ == "\_\_main\_\_":

run\_server()

#### **2. کد کلاینت**

کلاینت لیستی از پورت‌ها را به سرور ارسال می‌کند و وضعیت آن‌ها را دریافت و نمایش می‌دهد.

python

import socket

def run\_client():

# آدرس و پورت سرور

HOST = '127.0.0.1'

PORT = 12345

# ایجاد سوکت کلاینت

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client\_socket.connect((HOST, PORT))

print(f"Connected to server {HOST}:{PORT}")

while True:

# دریافت لیست پورت‌ها از کاربر

ports = input("Enter ports to scan (comma-separated, e.g., 80,443,22) or 'exit' to quit: ")

if ports.lower() == "exit":

client\_socket.send(ports.encode())

print("Exiting...")

break

# ارسال لیست پورت‌ها به سرور

client\_socket.send(ports.encode())

# دریافت نتایج از سرور

results = client\_socket.recv(1024).decode()

print(f"Scan results: {results}")

client\_socket.close()

if \_\_name\_\_ == "\_\_main\_\_":

run\_client()

### **توضیحات کد:**

#### **سرور:**

1. **بررسی وضعیت پورت‌ها:**
   * سرور با استفاده از متد connect\_ex از کتابخانه socket وضعیت هر پورت را بررسی می‌کند.
   * اگر اتصال موفق باشد، پورت باز است؛ در غیر این صورت، پورت بسته است.
2. **مدیریت ارتباط:**
   * سرور لیست پورت‌ها را از کلاینت دریافت می‌کند (به صورت جداشده با کاما).
   * وضعیت هر پورت را بررسی کرده و نتایج را به کلاینت ارسال می‌کند.

#### **کلاینت:**

1. **ارسال لیست پورت‌ها:**
   * کلاینت لیستی از پورت‌ها (مانند 80,443,22) را از کاربر دریافت می‌کند و به سرور ارسال می‌کند.
2. **دریافت نتایج:**
   * کلاینت نتایج بررسی وضعیت پورت‌ها را از سرور دریافت کرده و نمایش می‌دهد.

### **اجرای پروژه:**

#### **1. اجرای سرور:**

ابتدا سرور را اجرا کنید:

bash

python server.py

سرور آماده است و به درخواست‌های کلاینت گوش می‌دهد.

#### **2. اجرای کلاینت:**

سپس کلاینت را اجرا کنید:

bash

python client.py

کلاینت به سرور متصل می‌شود و می‌توانید لیست پورت‌ها را وارد کنید.

#### **3. تست اسکن پورت‌ها:**

در کلاینت، لیست پورت‌هایی که می‌خواهید بررسی کنید را وارد کنید، مثلاً:  
 bash

Enter ports to scan (comma-separated, e.g., 80,443,22) or 'exit' to quit: 80,443,22

خروجی کلاینت:  
 bash

Scan results: {80: True, 443: True, 22: False}

* در این مثال، پورت‌های 80 و 443 باز هستند، اما پورت 22 بسته است.

### **نمونه اجرا:**

#### **سرور:**

bash

Server is running on 127.0.0.1:12345...

Connected by ('127.0.0.1', 54321)

Received request: 80,443,22

#### **کلاینت:**

bash

Connected to server 127.0.0.1:12345

Enter ports to scan (comma-separated, e.g., 80,443,22) or 'exit' to quit: 80,443,22

Scan results: {80: True, 443: True, 22: False}

Enter ports to scan (comma-separated, e.g., 80,443,22) or 'exit' to quit: exit

Exiting...

### **نکات:**

1. **زمان انتظار (Timeout):**
   * زمان انتظار برای بررسی هر پورت به صورت پیش‌فرض 1 ثانیه تنظیم شده است. می‌توانید این مقدار را در متد settimeout تغییر دهید.
2. **امنیت:**
   * این سیستم فقط برای استفاده در شبکه‌های محلی طراحی شده است. برای استفاده در شبکه‌های عمومی، باید از رمزنگاری (مانند TLS/SSL) استفاده کنید.
3. **پشتیبانی از چند کلاینت:**
   * در حال حاضر، سرور فقط یک کلاینت را پشتیبانی می‌کند. برای پشتیبانی از چند کلاینت، می‌توانید از threading یا asyncio استفاده کنید.
4. **پورت‌های قابل بررسی:**
   * می‌توانید لیستی از پورت‌های رایج (مانند 80، 443، 22 و …) را برای تست وارد کنید.

### **نتیجه:**

این پروژه یک ابزار ساده برای اسکن پورت‌های باز در سرور است. می‌توانید از آن برای بررسی وضعیت پورت‌های خاص یا عیب‌یابی شبکه استفاده کنید. همچنین می‌توانید قابلیت‌های بیشتری مانند نمایش اطلاعات بیشتر درباره پورت‌ها یا ذخیره نتایج را به این پروژه اضافه کنید.

### **پروژه 6: تشخیص حملات DDOS**

### **توضیح پروژه:**

در این پروژه، یک سرور طراحی می‌کنیم که بتواند تعداد درخواست‌های ورودی را مانیتور کند. اگر تعداد درخواست‌های ورودی در یک بازه زمانی مشخص از حد مجاز بیشتر شود، سرور آن ارتباط را به عنوان یک حمله DDOS تشخیص داده و مسدود می‌کند.

### **پیاده‌سازی:**

#### **1. کد سرور**

سرور درخواست‌های ورودی را بررسی می‌کند و اگر تعداد درخواست‌ها از یک کلاینت خاص در بازه زمانی مشخص بیش از حد مجاز باشد، آن کلاینت را مسدود می‌کند.

python

import socket

import threading

import time

# تنظیمات سرور

HOST = '127.0.0.1'

PORT = 12345

MAX\_REQUESTS = 5 # حداکثر تعداد درخواست‌ها در بازه زمانی مشخص

TIME\_WINDOW = 10 # بازه زمانی به ثانیه

# دیکشنری برای ذخیره تعداد درخواست‌های هر کلاینت

client\_requests = {}

blocked\_clients = set()

def handle\_client(conn, addr):

global client\_requests, blocked\_clients

# بررسی اینکه آیا کلاینت مسدود شده است یا خیر

if addr[0] in blocked\_clients:

print(f"Blocked client {addr[0]} tried to connect.")

conn.send("You are blocked.".encode())

conn.close()

return

print(f"Connected by {addr}")

while True:

try:

# دریافت داده از کلاینت

data = conn.recv(1024).decode()

if not data or data.lower() == "exit":

print(f"Connection closed by {addr}")

break

print(f"Received from {addr}: {data}")

# ثبت درخواست کلاینت

current\_time = time.time()

if addr[0] not in client\_requests:

client\_requests[addr[0]] = []

client\_requests[addr[0]].append(current\_time)

# حذف درخواست‌های قدیمی‌تر از بازه زمانی مشخص

client\_requests[addr[0]] = [

req\_time for req\_time in client\_requests[addr[0]]

if current\_time - req\_time <= TIME\_WINDOW

]

# بررسی تعداد درخواست‌ها

if len(client\_requests[addr[0]]) > MAX\_REQUESTS:

print(f"DDOS detected from {addr[0]}. Blocking client.")

blocked\_clients.add(addr[0])

conn.send("You are blocked due to suspicious activity.".encode())

conn.close()

break

# ارسال پاسخ به کلاینت

conn.send("Request received.".encode())

except Exception as e:

print(f"Error with client {addr}: {e}")

break

conn.close()

def run\_server():

# ایجاد سوکت سرور

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.bind((HOST, PORT))

server\_socket.listen(5)

print(f"Server is running on {HOST}:{PORT}...")

while True:

# انتظار برای اتصال کلاینت

conn, addr = server\_socket.accept()

# ایجاد یک ترد برای مدیریت کلاینت

client\_thread = threading.Thread(target=handle\_client, args=(conn, addr))

client\_thread.start()

if \_\_name\_\_ == "\_\_main\_\_":

run\_server()

#### **2. کد کلاینت**

کلاینت درخواست‌هایی را به سرور ارسال می‌کند و پاسخ سرور را دریافت و نمایش می‌دهد.

python

import socket

import time

def run\_client():

# آدرس و پورت سرور

HOST = '127.0.0.1'

PORT = 12345

# ایجاد سوکت کلاینت

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client\_socket.connect((HOST, PORT))

print(f"Connected to server {HOST}:{PORT}")

while True:

# ارسال درخواست به سرور

message = input("Enter a message to send (or 'exit' to quit): ")

if message.lower() == "exit":

client\_socket.send(message.encode())

print("Exiting...")

break

client\_socket.send(message.encode())

# دریافت پاسخ از سرور

response = client\_socket.recv(1024).decode()

print(f"Response from server: {response}")

# شبیه‌سازی ارسال سریع درخواست‌ها

time.sleep(1)

client\_socket.close()

if \_\_name\_\_ == "\_\_main\_\_":

run\_client()

### **توضیحات کد:**

#### **سرور:**

1. **مدیریت درخواست‌ها:**
   * هر کلاینت با استفاده از آدرس IP خود شناسایی می‌شود.
   * درخواست‌های هر کلاینت در دیکشنری client\_requests ذخیره می‌شوند.
   * درخواست‌هایی که خارج از بازه زمانی مشخص (مثلاً 10 ثانیه) هستند، حذف می‌شوند.
2. **تشخیص حمله DDOS:**
   * اگر تعداد درخواست‌های یک کلاینت در بازه زمانی مشخص از مقدار MAX\_REQUESTS بیشتر شود، آن کلاینت به عنوان مشکوک شناسایی شده و مسدود می‌شود.
3. **مدیریت چند کلاینت:**
   * برای مدیریت چند کلاینت به صورت همزمان، از threading استفاده می‌شود.
4. **مسدود کردن کلاینت‌ها:**
   * آدرس IP کلاینت‌های مسدودشده در مجموعه blocked\_clients ذخیره می‌شود.
   * اگر کلاینت مسدودشده دوباره سعی کند متصل شود، سرور به او پاسخ می‌دهد که مسدود شده است.

#### **کلاینت:**

1. **ارسال درخواست:**
   * کلاینت پیام‌هایی را به سرور ارسال می‌کند.
   * می‌توانید با ارسال سریع درخواست‌ها، رفتار سرور در تشخیص حمله DDOS را تست کنید.
2. **دریافت پاسخ:**
   * کلاینت پاسخ سرور را دریافت کرده و نمایش می‌دهد.

### **اجرای پروژه:**

#### **1. اجرای سرور:**

ابتدا سرور را اجرا کنید:

bash

python server.py

سرور آماده است و به درخواست‌های کلاینت گوش می‌دهد.

#### **2. اجرای کلاینت:**

سپس کلاینت را اجرا کنید:

bash

python client.py

کلاینت به سرور متصل می‌شود و می‌توانید پیام‌هایی را به سرور ارسال کنید.

#### **3. تست حمله DDOS:**

* چندین درخواست سریع به سرور ارسال کنید (به صورت دستی یا با اجرای چند کلاینت به صورت همزمان).
* اگر تعداد درخواست‌ها از حد مجاز بیشتر شود، سرور کلاینت را مسدود می‌کند.

### **نمونه اجرا:**

#### **سرور:**

bash

Server is running on 127.0.0.1:12345...

Connected by ('127.0.0.1', 54321)

Received from ('127.0.0.1', 54321): Hello

Received from ('127.0.0.1', 54321): Test

Received from ('127.0.0.1', 54321): Another request

DDOS detected from 127.0.0.1. Blocking client.

Blocked client 127.0.0.1 tried to connect.

#### **کلاینت:**

bash

Connected to server 127.0.0.1:12345

Enter a message to send (or 'exit' to quit): Hello

Response from server: Request received.

Enter a message to send (or 'exit' to quit): Test

Response from server: Request received.

Enter a message to send (or 'exit' to quit): Another request

Response from server: You are blocked due to suspicious activity.

### **نکات:**

1. **پیکربندی آستانه‌ها:**
   * می‌توانید مقادیر MAX\_REQUESTS و TIME\_WINDOW را بر اساس نیاز تغییر دهید.
   * برای مثال، اگر سرور شما باید درخواست‌های زیادی را پردازش کند، این مقادیر را افزایش دهید.
2. **امنیت:**
   * این سیستم فقط برای شناسایی حملات ساده طراحی شده است. برای حملات پیچیده‌تر (مانند حملات توزیع‌شده)، باید از ابزارهای پیشرفته‌تری مانند فایروال‌ها یا سرویس‌های مدیریت DDOS استفاده کنید.
3. **پشتیبانی از چند کلاینت:**
   * با استفاده از threading، سرور می‌تواند چندین کلاینت را به صورت همزمان مدیریت کند.

### **نتیجه:**

این پروژه یک سیستم ساده برای شناسایی و مسدود کردن حملات DDOS است. می‌توانید این سیستم را بهبود دهید و قابلیت‌هایی مانند گزارش‌گیری، اعلان‌ها یا تنظیمات پیشرفته‌تر را به آن اضافه کنید.

### **پروژه 7: چت روم ایمن با TLS/SSL**

### **توضیح پروژه:**

در این پروژه، یک سیستم چت روم طراحی می‌کنیم که کاربران بتوانند به آن متصل شوند و پیام‌های خود را ارسال کنند. برای افزایش امنیت، ارتباطات بین کلاینت و سرور با استفاده از پروتکل TLS/SSL رمزنگاری می‌شوند.

### **پیاده‌سازی:**

#### **1. تنظیمات اولیه**

برای استفاده از TLS/SSL، باید یک گواهی SSL و کلید خصوصی ایجاد کنید. برای این کار می‌توانید از ابزار openssl استفاده کنید:

bash

openssl req -new -x509 -days 365 -nodes -out cert.pem -keyout key.pem

این دستور دو فایل ایجاد می‌کند:

* **cert.pem**: گواهی SSL
* **key.pem**: کلید خصوصی

#### **2. کد سرور**

سرور پیام‌های کلاینت‌ها را دریافت کرده و به سایر کلاینت‌ها ارسال می‌کند. ارتباطات با استفاده از TLS/SSL رمزنگاری می‌شوند.

python

import socket

import ssl

import threading

# تنظیمات سرور

HOST = '127.0.0.1'

PORT = 12345

# لیست کلاینت‌های متصل

clients = []

def broadcast(message, sender\_conn):

"""

ارسال پیام به تمام کلاینت‌ها به جز فرستنده

"""

for client in clients:

if client != sender\_conn:

try:

client.send(message)

except:

clients.remove(client)

def handle\_client(conn, addr):

"""

مدیریت ارتباط با کلاینت

"""

print(f"New connection from {addr}")

clients.append(conn)

while True:

try:

# دریافت پیام از کلاینت

message = conn.recv(1024)

if not message:

break

print(f"Message from {addr}: {message.decode()}")

# ارسال پیام به سایر کلاینت‌ها

broadcast(message, conn)

except:

break

print(f"Connection closed by {addr}")

clients.remove(conn)

conn.close()

def run\_server():

# ایجاد سوکت سرور

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.bind((HOST, PORT))

server\_socket.listen(5)

# افزودن TLS/SSL به سوکت

context = ssl.SSLContext(ssl.PROTOCOL\_TLS\_SERVER)

context.load\_cert\_chain(certfile="cert.pem", keyfile="key.pem")

secure\_server\_socket = context.wrap\_socket(server\_socket, server\_side=True)

print(f"Secure Chat Server is running on {HOST}:{PORT}...")

while True:

# پذیرش اتصال کلاینت

conn, addr = secure\_server\_socket.accept()

client\_thread = threading.Thread(target=handle\_client, args=(conn, addr))

client\_thread.start()

if \_\_name\_\_ == "\_\_main\_\_":

run\_server()

#### **3. کد کلاینت**

کلاینت پیام‌های خود را به سرور ارسال کرده و پیام‌های سایر کاربران را دریافت و نمایش می‌دهد.

python

import socket

import ssl

import threading

# تنظیمات سرور

HOST = '127.0.0.1'

PORT = 12345

def receive\_messages(secure\_socket):

"""

دریافت پیام‌ها از سرور

"""

while True:

try:

message = secure\_socket.recv(1024).decode()

if not message:

break

print(f"\n{message}")

except:

print("Disconnected from server.")

break

def run\_client():

# ایجاد سوکت کلاینت

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# افزودن TLS/SSL به سوکت

context = ssl.SSLContext(ssl.PROTOCOL\_TLS\_CLIENT)

context.load\_verify\_locations("cert.pem")

secure\_socket = context.wrap\_socket(client\_socket, server\_hostname=HOST)

# اتصال به سرور

secure\_socket.connect((HOST, PORT))

print("Connected to secure chat server!")

# ایجاد ترد برای دریافت پیام‌ها

receive\_thread = threading.Thread(target=receive\_messages, args=(secure\_socket,))

receive\_thread.start()

while True:

# ارسال پیام به سرور

message = input()

if message.lower() == "exit":

secure\_socket.close()

break

secure\_socket.send(message.encode())

if \_\_name\_\_ == "\_\_main\_\_":

run\_client()

### **توضیحات کد:**

#### **سرور:**

1. **رمزنگاری ارتباطات:**
   * از کلاس ssl.SSLContext برای ایجاد یک سوکت امن استفاده می‌شود.
   * گواهی SSL (cert.pem) و کلید خصوصی (key.pem) برای رمزنگاری استفاده می‌شوند.
2. **مدیریت کلاینت‌ها:**
   * لیست clients برای نگهداری سوکت‌های کلاینت‌های متصل استفاده می‌شود.
   * پیام‌های دریافتی از یک کلاینت به سایر کلاینت‌ها ارسال می‌شوند.
3. **مدیریت چند کلاینت:**
   * برای هر کلاینت یک ترد جداگانه ایجاد می‌شود تا ارتباطات به صورت همزمان مدیریت شوند.

#### **کلاینت:**

1. **رمزنگاری ارتباطات:**
   * کلاینت از گواهی SSL (cert.pem) برای اعتبارسنجی سرور استفاده می‌کند.
   * ارتباطات با استفاده از TLS/SSL رمزنگاری می‌شوند.
2. **دریافت و ارسال پیام‌ها:**
   * یک ترد جداگانه برای دریافت پیام‌ها از سرور ایجاد می‌شود.
   * پیام‌های کاربر به سرور ارسال می‌شوند.

### **اجرای پروژه:**

#### **1. اجرای سرور:**

ابتدا سرور را اجرا کنید:

bash

python server.py

سرور آماده است و به درخواست‌های کلاینت گوش می‌دهد.

#### **2. اجرای کلاینت:**

سپس کلاینت را اجرا کنید:

bash

python client.py

کلاینت به سرور متصل می‌شود و می‌توانید پیام‌ها را ارسال و دریافت کنید.

#### **3. تست چت روم:**

* چندین کلاینت را اجرا کنید و پیام‌هایی را ارسال کنید.
* پیام‌های هر کلاینت برای سایر کلاینت‌ها نیز نمایش داده می‌شود.

### **نمونه اجرا:**

#### **سرور:**

bash

Secure Chat Server is running on 127.0.0.1:12345...

New connection from ('127.0.0.1', 54321)

Message from ('127.0.0.1', 54321): Hello everyone!

Message from ('127.0.0.1', 54322): Hi there!

#### **کلاینت 1:**

bash

Connected to secure chat server!

Hello everyone!

Hi there!

#### **کلاینت 2:**

bash

Connected to secure chat server!

Hi there!

Hello everyone!

### **نکات:**

1. **گواهی SSL:**
   * گواهی SSL برای رمزنگاری استفاده می‌شود. در محیط‌های تولیدی، از گواهی‌های معتبر استفاده کنید.
2. **امنیت:**
   * ارتباطات بین کلاینت و سرور کاملاً رمزنگاری شده‌اند، بنابراین پیام‌ها در مسیر قابل شنود نیستند.
3. **مدیریت چند کلاینت:**
   * سرور با استفاده از threading می‌تواند چندین کلاینت را به صورت همزمان مدیریت کند.

### **نتیجه:**

این پروژه یک چت روم ایمن با استفاده از TLS/SSL است که ارتباطات بین کلاینت و سرور را رمزنگاری می‌کند. می‌توانید قابلیت‌های بیشتری مانند احراز هویت کاربران، ذخیره پیام‌ها یا مدیریت گروه‌های چت را به این پروژه اضافه کنید.

### **پروژه 8: سیستم مدیریت صف (Queue Management System)**

### **توضیح پروژه:**

در این پروژه، یک سیستم مدیریت صف طراحی می‌کنیم که به کاربران اجازه می‌دهد در یک صف قرار بگیرند، وضعیت خود را مشاهده کنند، یا از صف خارج شوند. این سیستم می‌تواند در محیط‌هایی مانند بانک‌ها، بیمارستان‌ها یا فروشگاه‌ها استفاده شود.

### **پیاده‌سازی:**

#### **1. کد سرور**

سرور درخواست‌های کلاینت‌ها را مدیریت می‌کند و وضعیت صف را به‌روزرسانی یا گزارش می‌دهد.

python

import socket

import threading

# صف برای نگهداری کاربران

queue = []

def handle\_client(conn, addr):

"""

مدیریت درخواست‌های کلاینت

"""

print(f"New connection from {addr}")

while True:

try:

# دریافت درخواست از کلاینت

request = conn.recv(1024).decode()

if not request or request.lower() == "exit":

print(f"Connection closed by {addr}")

break

print(f"Request from {addr}: {request}")

# پردازش درخواست

response = process\_request(request, addr)

conn.send(response.encode())

except Exception as e:

print(f"Error with client {addr}: {e}")

break

conn.close()

def process\_request(request, addr):

"""

پردازش درخواست کلاینت

"""

global queue

if request.lower() == "join":

# اضافه شدن به صف

if addr[0] not in queue:

queue.append(addr[0])

return f"You have joined the queue. Your position: {len(queue)}"

else:

return "You are already in the queue."

elif request.lower() == "leave":

# ترک صف

if addr[0] in queue:

queue.remove(addr[0])

return "You have left the queue."

else:

return "You are not in the queue."

elif request.lower() == "status":

# وضعیت صف

if addr[0] in queue:

position = queue.index(addr[0]) + 1

return f"You are in the queue. Your position: {position}"

else:

return "You are not in the queue."

elif request.lower() == "list":

# لیست کل صف

return f"Current queue: {queue}"

else:

return "Invalid command. Use 'join', 'leave', 'status', or 'list'."

def run\_server():

# تنظیمات سرور

HOST = '127.0.0.1'

PORT = 12345

# ایجاد سوکت سرور

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.bind((HOST, PORT))

server\_socket.listen(5)

print(f"Queue Management Server is running on {HOST}:{PORT}...")

while True:

# پذیرش اتصال کلاینت

conn, addr = server\_socket.accept()

client\_thread = threading.Thread(target=handle\_client, args=(conn, addr))

client\_thread.start()

if \_\_name\_\_ == "\_\_main\_\_":

run\_server()

#### **2. کد کلاینت**

کلاینت درخواست‌هایی مانند پیوستن به صف، ترک صف، مشاهده وضعیت، یا مشاهده لیست کامل صف را به سرور ارسال می‌کند.

python

import socket

def run\_client():

# تنظیمات سرور

HOST = '127.0.0.1'

PORT = 12345

# ایجاد سوکت کلاینت

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client\_socket.connect((HOST, PORT))

print(f"Connected to Queue Management Server at {HOST}:{PORT}")

while True:

# دریافت دستور از کاربر

command = input("Enter command (join, leave, status, list, exit): ").lower()

if command == "exit":

client\_socket.send(command.encode())

print("Exiting...")

break

# ارسال درخواست به سرور

client\_socket.send(command.encode())

# دریافت پاسخ از سرور

response = client\_socket.recv(1024).decode()

print(f"Response from server: {response}")

client\_socket.close()

if \_\_name\_\_ == "\_\_main\_\_":

run\_client()

### **توضیحات کد:**

#### **سرور:**

1. **مدیریت صف:**
   * سرور یک لیست به نام queue نگه می‌دارد که آدرس IP کلاینت‌ها را ذخیره می‌کند.
   * هر کلاینت می‌تواند به صف اضافه شود، از صف خارج شود، یا وضعیت خود را مشاهده کند.
2. **پردازش درخواست‌ها:**
   * join: کلاینت به صف اضافه می‌شود.
   * leave: کلاینت از صف حذف می‌شود.
   * status: موقعیت کلاینت در صف نمایش داده می‌شود.
   * list: کل صف به کلاینت نمایش داده می‌شود.
3. **مدیریت چند کلاینت:**
   * سرور با استفاده از threading می‌تواند چندین کلاینت را به صورت همزمان مدیریت کند.

#### **کلاینت:**

1. **ارسال درخواست:**
   * کلاینت دستورات مختلفی (مانند join, leave, status, list) را به سرور ارسال می‌کند.
   * کلاینت می‌تواند با ارسال دستور exit از سیستم خارج شود.
2. **دریافت پاسخ:**
   * کلاینت پاسخ سرور را دریافت کرده و نمایش می‌دهد.

### **اجرای پروژه:**

#### **1. اجرای سرور:**

ابتدا سرور را اجرا کنید:

bash

python server.py

سرور آماده است و به درخواست‌های کلاینت گوش می‌دهد.

#### **2. اجرای کلاینت:**

سپس کلاینت را اجرا کنید:

bash

python client.py

کلاینت به سرور متصل می‌شود و می‌توانید دستورات مختلف را ارسال کنید.

#### **3. تست دستورات:**

* **پیوستن به صف:**

در کلاینت:  
 bash

Enter command (join, leave, status, list, exit): join

پاسخ سرور:  
 bash

Response from server: You have joined the queue. Your position: 1

* **مشاهده وضعیت:**

در کلاینت:  
 bash

Enter command (join, leave, status, list, exit): status

پاسخ سرور:  
 bash

Response from server: You are in the queue. Your position: 1

* **مشاهده کل صف:**

در کلاینت:  
 bash

Enter command (join, leave, status, list, exit): list

پاسخ سرور:  
 bash

Response from server: Current queue: ['127.0.0.1']

* **ترک صف:**

در کلاینت:  
 bash

Enter command (join, leave, status, list, exit): leave

پاسخ سرور:  
 bash

Response from server: You have left the queue.

### **نمونه اجرا:**

#### **سرور:**

bash

Queue Management Server is running on 127.0.0.1:12345...

New connection from ('127.0.0.1', 54321)

Request from ('127.0.0.1', 54321): join

Request from ('127.0.0.1', 54321): status

Request from ('127.0.0.1', 54321): list

Request from ('127.0.0.1', 54321): leave

Connection closed by ('127.0.0.1', 54321)

#### **کلاینت:**

bash

Connected to Queue Management Server at 127.0.0.1:12345

Enter command (join, leave, status, list, exit): join

Response from server: You have joined the queue. Your position: 1

Enter command (join, leave, status, list, exit): status

Response from server: You are in the queue. Your position: 1

Enter command (join, leave, status, list, exit): list

Response from server: Current queue: ['127.0.0.1']

Enter command (join, leave, status, list, exit): leave

Response from server: You have left the queue.

Enter command (join, leave, status, list, exit): exit

Exiting...

### **نکات:**

1. **مدیریت صف:**
   * در حال حاضر، صف به صورت ساده مدیریت می‌شود. می‌توانید قابلیت‌های بیشتری مانند اولویت‌بندی یا محدودیت تعداد کاربران را اضافه کنید.
2. **امنیت:**
   * برای محیط‌های واقعی، می‌توانید از TLS/SSL برای رمزنگاری ارتباطات استفاده کنید.
3. **ذخیره‌سازی پایدار:**
   * در حال حاضر، صف فقط در حافظه نگهداری می‌شود. می‌توانید از یک پایگاه داده یا فایل برای ذخیره‌سازی پایدار استفاده کنید.

### **نتیجه:**

این پروژه یک سیستم ساده مدیریت صف است که می‌تواند در محیط‌های مختلف استفاده شود. قابلیت‌های بیشتری مانند نمایش زمان انتظار، اولویت‌بندی کاربران، یا ارسال اعلان‌ها را می‌توانید به آن اضافه کنید.

### **پروژه 9: سیستم ذخیره‌سازی و بازیابی فایل (File Storage and Retrieval System)**

### **توضیح پروژه:**

در این پروژه، یک سیستم Client-Server طراحی می‌کنیم که به کاربران اجازه می‌دهد فایل‌های خود را روی سرور آپلود کنند، فایل‌های ذخیره‌شده را مشاهده کنند و در صورت نیاز فایل‌ها را دانلود کنند. این سیستم می‌تواند به عنوان یک سرویس ذخیره‌سازی فایل ساده عمل کند.

### **پیاده‌سازی:**

#### **1. کد سرور**

سرور درخواست‌های کلاینت را مدیریت می‌کند و عملیات آپلود، دانلود و نمایش لیست فایل‌ها را انجام می‌دهد.

python

import socket

import os

# پوشه‌ای برای ذخیره فایل‌ها

STORAGE\_DIR = "server\_storage"

os.makedirs(STORAGE\_DIR, exist\_ok=True)

def handle\_client(conn, addr):

"""

مدیریت درخواست‌های کلاینت

"""

print(f"New connection from {addr}")

while True:

try:

# دریافت دستور از کلاینت

command = conn.recv(1024).decode()

if not command or command.lower() == "exit":

print(f"Connection closed by {addr}")

break

print(f"Command from {addr}: {command}")

# پردازش دستور

if command.lower() == "list":

# ارسال لیست فایل‌ها

files = os.listdir(STORAGE\_DIR)

response = "\n".join(files) if files else "No files available."

conn.send(response.encode())

elif command.lower().startswith("upload"):

# آپلود فایل

\_, filename = command.split()

filepath = os.path.join(STORAGE\_DIR, filename)

with open(filepath, "wb") as f:

while True:

data = conn.recv(1024)

if data == b"EOF":

break

f.write(data)

conn.send(f"File '{filename}' uploaded successfully.".encode())

elif command.lower().startswith("download"):

# دانلود فایل

\_, filename = command.split()

filepath = os.path.join(STORAGE\_DIR, filename)

if os.path.exists(filepath):

with open(filepath, "rb") as f:

while chunk := f.read(1024):

conn.send(chunk)

conn.send(b"EOF")

else:

conn.send(f"File '{filename}' not found.".encode())

else:

conn.send("Invalid command. Use 'list', 'upload <filename>', or 'download <filename>'.".encode())

except Exception as e:

print(f"Error with client {addr}: {e}")

break

conn.close()

def run\_server():

# تنظیمات سرور

HOST = '127.0.0.1'

PORT = 12345

# ایجاد سوکت سرور

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.bind((HOST, PORT))

server\_socket.listen(5)

print(f"File Storage Server is running on {HOST}:{PORT}...")

while True:

# پذیرش اتصال کلاینت

conn, addr = server\_socket.accept()

handle\_client(conn, addr)

if \_\_name\_\_ == "\_\_main\_\_":

run\_server()

#### **2. کد کلاینت**

کلاینت می‌تواند فایل‌ها را آپلود کند، فایل‌های موجود در سرور را مشاهده کند و فایل‌ها را دانلود کند.

python

import socket

import os

def run\_client():

# تنظیمات سرور

HOST = '127.0.0.1'

PORT = 12345

# ایجاد سوکت کلاینت

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client\_socket.connect((HOST, PORT))

print(f"Connected to File Storage Server at {HOST}:{PORT}")

while True:

# دریافت دستور از کاربر

command = input("Enter command (list, upload <filename>, download <filename>, exit): ").lower()

if command == "exit":

client\_socket.send(command.encode())

print("Exiting...")

break

# ارسال دستور به سرور

client\_socket.send(command.encode())

if command.startswith("upload"):

# آپلود فایل

\_, filename = command.split()

if os.path.exists(filename):

with open(filename, "rb") as f:

while chunk := f.read(1024):

client\_socket.send(chunk)

client\_socket.send(b"EOF")

response = client\_socket.recv(1024).decode()

print(f"Response from server: {response}")

else:

print(f"File '{filename}' not found locally.")

elif command.startswith("download"):

# دانلود فایل

\_, filename = command.split()

with open(filename, "wb") as f:

while True:

data = client\_socket.recv(1024)

if data == b"EOF":

break

f.write(data)

print(f"File '{filename}' downloaded successfully.")

else:

# دریافت پاسخ از سرور

response = client\_socket.recv(1024).decode()

print(f"Response from server: {response}")

client\_socket.close()

if \_\_name\_\_ == "\_\_main\_\_":

run\_client()

### **توضیحات کد:**

#### **سرور:**

1. **مدیریت فایل‌ها:**
   * فایل‌ها در پوشه server\_storage ذخیره می‌شوند.
   * اگر پوشه وجود نداشته باشد، به صورت خودکار ایجاد می‌شود.
2. **دستورات پشتیبانی‌شده:**
   * list: لیست فایل‌های موجود در سرور را به کلاینت ارسال می‌کند.
   * upload <filename>: فایل را از کلاینت دریافت و ذخیره می‌کند.
   * download <filename>: فایل را به کلاینت ارسال می‌کند.

#### **کلاینت:**

1. **ارسال دستورات:**
   * کلاینت دستورات مختلفی را به سرور ارسال می‌کند.
   * برای آپلود فایل، فایل را به سرور ارسال می‌کند.
   * برای دانلود فایل، فایل را از سرور دریافت می‌کند.
2. **مدیریت فایل‌ها:**
   * فایل‌های دانلودشده در همان پوشه‌ای که کلاینت اجرا می‌شود ذخیره می‌شوند.

### **اجرای پروژه:**

#### **1. اجرای سرور:**

ابتدا سرور را اجرا کنید:

bash

python server.py

سرور آماده است و به درخواست‌های کلاینت گوش می‌دهد.

#### **2. اجرای کلاینت:**

سپس کلاینت را اجرا کنید:

bash

python client.py

کلاینت به سرور متصل می‌شود و می‌توانید دستورات مختلف را ارسال کنید.

#### **3. تست دستورات:**

* **مشاهده فایل‌ها:**

در کلاینت:  
 bash

Enter command (list, upload <filename>, download <filename>, exit): list

پاسخ سرور:  
 bash

Response from server: No files available.

* **آپلود فایل:**

در کلاینت:  
 bash

Enter command (list, upload <filename>, download <filename>, exit): upload test.txt

پاسخ سرور:  
 bash

Response from server: File 'test.txt' uploaded successfully.

* **دانلود فایل:**

در کلاینت:  
 bash

Enter command (list, upload <filename>, download <filename>, exit): download test.txt

* + فایل test.txt در پوشه کلاینت ذخیره می‌شود.

### **نمونه اجرا:**

#### **سرور:**

bash

File Storage Server is running on 127.0.0.1:12345...

Command from ('127.0.0.1', 54321): upload test.txt

Command from ('127.0.0.1', 54321): list

Command from ('127.0.0.1', 54321): download test.txt

#### **کلاینت:**

bash

Connected to File Storage Server at 127.0.0.1:12345

Enter command (list, upload <filename>, download <filename>, exit): upload test.txt

Response from server: File 'test.txt' uploaded successfully.

Enter command (list, upload <filename>, download <filename>, exit): list

Response from server: test.txt

Enter command (list, upload <filename>, download <filename>, exit): download test.txt

File 'test.txt' downloaded successfully.

### **نکات:**

1. **امنیت:**
   * در حال حاضر، فایل‌ها بدون رمزنگاری ارسال می‌شوند. برای امنیت بیشتر، می‌توانید از TLS/SSL استفاده کنید.
2. **مدیریت فایل‌های بزرگ:**
   * برای فایل‌های بزرگ، ارسال و دریافت در بلوک‌های کوچک انجام می‌شود تا از مصرف زیاد حافظه جلوگیری شود.
3. **ذخیره‌سازی پایدار:**
   * فایل‌های آپلودشده به صورت پایدار روی سرور ذخیره می‌شوند.

### **نتیجه:**

این پروژه یک سیستم ساده برای ذخیره‌سازی و بازیابی فایل‌ها است. می‌توانید قابلیت‌های بیشتری مانند احراز هویت کاربران، رمزنگاری فایل‌ها، یا مدیریت دسترسی را به آن اضافه کنید.

### **پروژه 10: ماشین حساب توزیع‌شده (Distributed Calculator)**

### **توضیح پروژه:**

در این پروژه، یک سیستم Client-Server طراحی می‌کنیم که کلاینت‌ها بتوانند عملیات ریاضی (مانند جمع، تفریق، ضرب و تقسیم) را به سرور ارسال کنند و سرور نتیجه را محاسبه کرده و به کلاینت بازگرداند. این پروژه می‌تواند به عنوان یک ماشین حساب توزیع‌شده عمل کند.

### **پیاده‌سازی:**

#### **1. کد سرور**

سرور درخواست‌های کلاینت‌ها را دریافت کرده، عملیات ریاضی را پردازش می‌کند و نتیجه را به کلاینت بازمی‌گرداند.

python

import socket

import threading

def calculate(expression):

"""

پردازش عبارت ریاضی و بازگشت نتیجه

"""

try:

# ارزیابی عبارت ریاضی

result = eval(expression)

return f"Result: {result}"

except ZeroDivisionError:

return "Error: Division by zero is not allowed."

except Exception as e:

return f"Error: Invalid expression. {e}"

def handle\_client(conn, addr):

"""

مدیریت درخواست‌های کلاینت

"""

print(f"New connection from {addr}")

while True:

try:

# دریافت عبارت ریاضی از کلاینت

expression = conn.recv(1024).decode()

if not expression or expression.lower() == "exit":

print(f"Connection closed by {addr}")

break

print(f"Received from {addr}: {expression}")

# محاسبه نتیجه

result = calculate(expression)

# ارسال نتیجه به کلاینت

conn.send(result.encode())

except Exception as e:

print(f"Error with client {addr}: {e}")

break

conn.close()

def run\_server():

# تنظیمات سرور

HOST = '127.0.0.1'

PORT = 12345

# ایجاد سوکت سرور

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_socket.bind((HOST, PORT))

server\_socket.listen(5)

print(f"Distributed Calculator Server is running on {HOST}:{PORT}...")

while True:

# پذیرش اتصال کلاینت

conn, addr = server\_socket.accept()

client\_thread = threading.Thread(target=handle\_client, args=(conn, addr))

client\_thread.start()

if \_\_name\_\_ == "\_\_main\_\_":

run\_server()

#### **2. کد کلاینت**

کلاینت عبارت‌های ریاضی را به سرور ارسال می‌کند و نتیجه را دریافت و نمایش می‌دهد.

python

import socket

def run\_client():

# تنظیمات سرور

HOST = '127.0.0.1'

PORT = 12345

# ایجاد سوکت کلاینت

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

client\_socket.connect((HOST, PORT))

print(f"Connected to Distributed Calculator Server at {HOST}:{PORT}")

while True:

# دریافت عبارت ریاضی از کاربر

expression = input("Enter a mathematical expression (or 'exit' to quit): ")

if expression.lower() == "exit":

client\_socket.send(expression.encode())

print("Exiting...")

break

# ارسال عبارت به سرور

client\_socket.send(expression.encode())

# دریافت نتیجه از سرور

result = client\_socket.recv(1024).decode()

print(f"Response from server: {result}")

client\_socket.close()

if \_\_name\_\_ == "\_\_main\_\_":

run\_client()

### **توضیحات کد:**

#### **سرور:**

1. **محاسبه عبارت ریاضی:**
   * سرور از تابع eval برای ارزیابی عبارت ریاضی استفاده می‌کند.
   * خطاهای احتمالی مانند تقسیم بر صفر یا عبارات نامعتبر مدیریت می‌شوند.
2. **مدیریت چند کلاینت:**
   * سرور با استفاده از threading می‌تواند چندین کلاینت را به صورت همزمان مدیریت کند.
3. **امنیت:**
   * استفاده از eval ممکن است خطرناک باشد زیرا امکان اجرای کدهای مخرب را فراهم می‌کند. برای افزایش امنیت، می‌توانید از یک کتابخانه ریاضی مانند sympy برای پردازش عبارات استفاده کنید.

#### **کلاینت:**

1. **ارسال عبارت ریاضی:**
   * کلاینت عبارت ریاضی را از کاربر دریافت کرده و به سرور ارسال می‌کند.
2. **دریافت نتیجه:**
   * کلاینت نتیجه محاسبه را از سرور دریافت کرده و نمایش می‌دهد.

### **اجرای پروژه:**

#### **1. اجرای سرور:**

ابتدا سرور را اجرا کنید:

bash

python server.py

سرور آماده است و به درخواست‌های کلاینت گوش می‌دهد.

#### **2. اجرای کلاینت:**

سپس کلاینت را اجرا کنید:

bash

python client.py

کلاینت به سرور متصل می‌شود و می‌توانید عبارات ریاضی را ارسال کنید.

#### **3. تست عبارات:**

* در کلاینت، عبارات مختلفی را وارد کنید، مانند:
  + 2 + 3 \* 4
  + (10 / 2) + 5
  + 10 / 0 (برای تست مدیریت خطا)

### **نمونه اجرا:**

#### **سرور:**

bash

Distributed Calculator Server is running on 127.0.0.1:12345...

New connection from ('127.0.0.1', 54321)

Received from ('127.0.0.1', 54321): 2 + 3 \* 4

Received from ('127.0.0.1', 54321): 10 / 0

Connection closed by ('127.0.0.1', 54321)

#### **کلاینت:**

bash

Connected to Distributed Calculator Server at 127.0.0.1:12345

Enter a mathematical expression (or 'exit' to quit): 2 + 3 \* 4

Response from server: Result: 14

Enter a mathematical expression (or 'exit' to quit): 10 / 0

Response from server: Error: Division by zero is not allowed.

Enter a mathematical expression (or 'exit' to quit): exit

Exiting...

### **نکات:**

1. **امنیت:**
   * استفاده از eval ممکن است خطرناک باشد زیرا کاربران می‌توانند کدهای مخرب ارسال کنند. برای جلوگیری از این مشکل:
     + از یک کتابخانه ریاضی مانند sympy برای پردازش عبارات استفاده کنید.
     + ورودی‌ها را به دقت بررسی کنید و فقط عبارات ریاضی معتبر را بپذیرید.
2. **پشتیبانی از چند کلاینت:**
   * سرور می‌تواند چندین کلاینت را به صورت همزمان مدیریت کند.
3. **گسترش قابلیت‌ها:**
   * می‌توانید عملیات بیشتری مانند توان، لگاریتم، یا توابع مثلثاتی را به سیستم اضافه کنید.

### **نتیجه:**

این پروژه یک ماشین حساب توزیع‌شده ساده است که می‌تواند عبارات ریاضی را از چندین کلاینت دریافت و پردازش کند. می‌توانید این پروژه را با افزودن قابلیت‌های بیشتر مانند احراز هویت کاربران، پشتیبانی از عبارات پیچیده‌تر، یا رابط کاربری گرافیکی (GUI) گسترش دهید.