



دانشگاه صنعتی امیرکبیر  
(پلی تکنیک تهران)

## فرم گزارش کار آزمایشگاه شبکه



دانشکده مهندسی کامپیوتر

نام و نام خانوادگی	حسین تاتار	شماره دانشجویی	40133014	نام و شماره آزمایش	Socket Programming با زبان پایتون
هدف آزمایش	هدف این آزمایش این است که با مفاهیم اولیه برنامه نویسی سوکت آشنا شوید. در این آزمایش، چگونگی استفاده از سوکتها و ایجاد یک ارتباط شبکه ای بین دو برنامه آموزش داده خواهد شد همچنین، با تفاوت های بین پروتکل های TCP و UDP نحوه ارسال و دریافت داده از طریق شبکه، و پیاده سازی یک سرور و کلاینت آشنا خواهید شد.				
ابزارهای مورد نیاز	نصب پایتون ویرایشگر متن دسترسی به ترمینال یا خط فرمان حداقل بر روی یک ماشین مجازی و سیستم عامل خودتان				
شرح آزمایش	<p><b>سوال 1:</b> یک سوکت TCP بسازید که به سروری با آدرس IP محلی 127.0.0.1 و پورت 1443 متصل شود و فرض کنید که سرور در حال حاضر در دسترس نیست، برای مدیریت خطاهای اتصال و جلوگیری از کرش کردن برنامه با استفاده از Try،Except کد مناسب بنویسید.</p> <p><b>جواب:</b></p> <p>کد مورد نظر به صورت زیر است:</p> <pre> TCP_socket_try_except.py X TCP_socket_try_except.py &gt; ... 1  import socket 2 3  # Socket configuration 4  HOST = '127.0.0.1' # Localhost IP address 5  PORT = 1443      # Target port 6 7  try: 8      # Create a TCP socket 9      client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM) 10 11     # Attempt to connect to the server 12     client_socket.connect((HOST, PORT)) 13     print("Successfully connected to the server!") 14 15 except socket.error as e: # Handle socket-related errors 16     print(f"Connection failed: {e}") 17 18 except Exception as e:    # Handle other unexpected errors 19     print(f"An error occurred: {e}") 20 21 finally: 22     # Close the socket to free resources 23     client_socket.close() 24     print("Socket closed.") </pre> <p>توضیحات کد:</p> <ol style="list-style-type: none"> <li>ایجاد سوکت TCP <ul style="list-style-type: none"> <li>socket.AF_INET نشان دهنده استفاده از IPv4 است.</li> <li>socket.SOCK_STREAM نشان دهنده استفاده از پروتکل TCP است.</li> </ul> </li> <li>مدیریت خطاها (try-except)</li> </ol>				

- اگر سرور در دسترس نباشد یا مشکلی در اتصال وجود داشته باشد socket.error یا Connection Refused رخ می‌دهد.
  - با استفاده از except، برنامه به جای کرش کردن، پیام خطا را نمایش می‌دهد.
  - 3. بستن سوکت (finally)
  - حتی اگر خطایی رخ دهد، سوکت در بلوک finally بسته می‌شود تا از مصرف منابع جلوگیری شود.
- جواب نهایی چایی توسط کد چون سرور در دسترس نیست:

```
PS C:\Users\hosse\OneDrive\Desktop\CNLab7> python .\TCP_socket_try_except.py
Connection failed: [WinError 10061] No connection could be made because the target machine actively refused it
Socket closed.
PS C:\Users\hosse\OneDrive\Desktop\CNLab7> █
```

**سوال 2:** یک سوکت سرور با ای پی و پورت دلخواه ساخته و سپس از طریق ابزارهایی مانند nc متصل شده و در سمت سرور اطلاعات کلاینت متصل شده را مشاهده کنید.

**جواب:**

کد سوال به صورت زیر است:

```
TCP_server.py X
TCP_server.py > ...
1 import socket
2
3 # Server configuration
4 HOST = '127.0.0.1' # Localhost IP
5 PORT = 12345 # Port to listen on
6
7 # Create a TCP socket
8 server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
9
10 try:
11     # Bind the socket to the address and port
12     server_socket.bind((HOST, PORT))
13
14     # Listen for incoming connections (max 5 queued connections)
15     server_socket.listen(5)
16     print(f"Server is listening on {HOST}:{PORT}...")
17
18     # Accept a client connection
19     client_socket, addr = server_socket.accept()
20     print(f"Connection established with {addr}")
21
22     # Send a welcome message to the client (encoded to bytes)
23     client_socket.send("Welcome to the server!".encode())
24
25     # Receive data from the client (max 1024 bytes)
26     data = client_socket.recv(1024)
27     print(f"Received from client: {data.decode()}")
28
29 except socket.error as e:
30     print(f"Server error: {e}")
31
32 finally:
33     # Close the client and server sockets
34     client_socket.close()
35     server_socket.close()
36     print("Server shut down.")
```

ابتدا کد را اجرا کرده و سپس با استفاده از دستور زیر در محیط ترمینال به سرور متصل می‌شویم:

```
C:\Users\hosse>ncat 127.0.0.1 12345
Welcome to the server! hello server
^C
C:\Users\hosse>█
```

با این اتصال سرور پیام خوشامدگویی را نمایش می‌دهد و با ارسال پیام پیام به سمت سرور منتقل می‌شود:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER
PS C:\Users\hosse\OneDrive\Desktop\CNLab7> python .\TCP_server.py
Server is listening on 127.0.0.1:12345...
Connection established with ('127.0.0.1', 22921)
Received from client: hello server

Server shut down.
PS C:\Users\hosse\OneDrive\Desktop\CNLab7> █
```

می‌بینیم که اطلاعات کلاینت متصل شده در سمت سرور نمایش داده شده است.

**سوال 3:** یک کلاینت با پروتکل UDP بسازید که پیامی شامل "درخواست زمان" را به یک سرور ارسال کند. سرور باید زمان فعلی سیستم را دریافت کرده و به کلاینت بازگرداند. در پاسخ خود توضیح دهید که چرا در پروتکل UDP نیازی به متد های `accept()` و `listen()`، `connect()` نیست.

**جواب:** کد کلاینت به صورت زیر است:

```
UDP_client.py X UDP_server.py
UDP_client.py > ...
1 import socket
2
3 # Client configuration
4 HOST = '127.0.0.1' # Server IP (localhost)
5 PORT = 12345      # Server port
6
7 # Create a UDP socket
8 client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
9
10 try:
11     # Send message to server (encoded to bytes)
12     message = "Request time"
13     client_socket.sendto(message.encode(), (HOST, PORT))
14     print("Sent to server:", message)
15
16     # Receive response from server (max 1024 bytes)
17     data, addr = client_socket.recvfrom(1024)
18     print("Received from server:", data.decode())
19
20 except socket.error as e:
21     print("UDP Error:", e)
22
23 finally:
24     client_socket.close()
```

و کد سرور ان نیز به صورت زیر میباشد:

```
UDP_client.py X UDP_server.py X
UDP_server.py > ...
1 import socket
2 from datetime import datetime
3
4 # Server configuration
5 HOST = '127.0.0.1'
6 PORT = 12345
7
8 # Create UDP socket
9 server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
10 server_socket.bind((HOST, PORT))
11
12 print("UDP Server is listening...")
13
14 while True:
15     # Receive data from client
16     data, addr = server_socket.recvfrom(1024)
17     print("Received from client:", data.decode())
18
19     # Send current time back to client
20     current_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
21     server_socket.sendto(current_time.encode(), addr)
```

چرا در UDP نیازی به `connect()`، `listen()` و `accept()` نیست؟

- اتصال گرا نبودن (Connectionless):  
UDP برخلاف TCP، حالت اتصال (connection) ندارد. هر بسته داده مستقل ارسال میشود و نیازی به handshake اولیه مانند TCP ندارد.
- عدم نیاز به مدیریت اتصال:  
UDP هیچ تضمینی برای تحویل داده یا ترتیب آن ندارد، بنابراین نیازی به نگهداری state (وضعیت اتصال) مانند TCP نیست.

نحوه اجرا و تست:

ابتدا سرور را اجرا میکنیم و سپس کلاینت را اجرا میکنیم خروجی سرور و کلاینت به صورت زیر میباشد که حاوی زمان دریافت شده از سرور است:

```
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER 2
PS C:\Users\hosse\OneDrive\Desktop\CNLab7> python .\UDP_server.py
UDP Server is listening...
Received from client: Request time
```

```
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS SPELL CHECKER 2
PS C:\Users\hosse\OneDrive\Desktop\CNLab7> python udp_client.py
Sent to server: Request time
Received from server: 2025-04-25 01:07:57
PS C:\Users\hosse\OneDrive\Desktop\CNLab7>
```

\*\*\* دقت داشته باشید که نمیتوان یک پیام کلاینت UDP را به یک سرور TCP ارسال نمود. \*\*\*

**سوال 4:** با استفاده از توابع تعریف شده کدی بنویسید که در آن سرور یک فایل با حجم 2048 به سمت کلاینت ارسال کند و پیغام ارسال و دریافت موفق نیز سمت کلاینت و سرور چاپ شود.

**جواب:** کد سمت ارسال فایل (سرور):  
فایل example.txt را به کلاینت ارسال میکند.

```
file_client.py  file_server.py X
file_server.py > send_file
1 import socket
2
3 def send_file(file_name, host, port):
4     # Create TCP socket
5     server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6     server_socket.bind((host, port))
7     server_socket.listen(1)
8     print(f"[SERVER] Waiting for client on {host}:{port}...")
9
10    # Accept client connection
11    client_socket, addr = server_socket.accept()
12    print(f"[SERVER] Connected to {addr}")
13
14    try:
15        # Open and send file in chunks of 1024 bytes
16        with open(file_name, 'rb') as file:
17            while True:
18                file_data = file.read(1024) # Read 1KB at a time
19                if not file_data:
20                    break
21                client_socket.send(file_data)
22            print("[SERVER] File sent successfully!")
23
24    except Exception as e:
25        print(f"[SERVER] Error: {e}")
26
27    finally:
28        client_socket.close()
29        server_socket.close()
30
31    # Example usage
32    send_file("example.txt", "127.0.0.1", 12345)
```

کد سمت دریافت فایل (کلاینت):

فایل دریافتی با اسم جدید received\_example.txt ذخیره میگردد.

```
file_client.py X  file_server.py
file_client.py > ...
1 import socket
2
3 def receive_file(file_name, server_ip, server_port):
4     # Create TCP socket
5     client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
6     client_socket.connect((server_ip, server_port))
7     print(f"[CLIENT] Connected to server at {server_ip}:{server_port}")
8
9
10    try:
11        # Receive file in chunks of 1024 bytes
12        with open(file_name, 'wb') as file:
13            while True:
14                file_data = client_socket.recv(1024) # Receive 1KB at a time
15                if not file_data:
16                    break
17                file.write(file_data)
18            print("[CLIENT] File received successfully!")
19
20    except Exception as e:
21        print(f"[CLIENT] Error: {e}")
22
23    finally:
24        client_socket.close()
25
26    # Example usage
27    receive_file("received_example.txt", "127.0.0.1", 12345)
```

## توضیحات کد

### 1. سرور (send\_file):

- یک سوکت TCP ایجاد می‌کند و در حالت listen قرار می‌گیرد.
- فایل را به صورت باینری ('rb') خوانده و در قطعات 1024 بایتی ارسال می‌کند.
- پس از اتمام ارسال، سوکت را می‌بندد.

### 2. کلاینت (receive\_file):

- به سرور متصل شده و داده‌ها را در قطعات 1024 بایتی دریافت می‌کند.
- داده‌های دریافتی را در فایل جدید ذخیره می‌کند. ('wb')
- پیام موفقیت را چاپ می‌کند.

حال با اجرای ابتدا سرور و سپس کلاینت پیام‌های خروجی زیر مشاهده می‌شود:

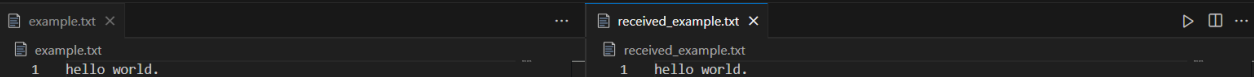
خروجی سمت سرور:

```
PS C:\Users\hosse\OneDrive\Desktop\CNLab7> python .\file_server.py
[SERVER] Waiting for client on 127.0.0.1:12345...
[SERVER] Connected to ('127.0.0.1', 63169)
[SERVER] File sent successfully!
```

خروجی سمت کلاینت:

```
PS C:\Users\hosse\OneDrive\Desktop\CNLab7> python .\file_client.py
[CLIENT] Connected to server at 127.0.0.1:12345
[CLIENT] File received successfully!
```

با اجرای این کدها فایل ایجاد می‌گردد با محتوای ارسالی که در عکس زیر قابل مشاهده است:



```
example.txt  ×  ...  received_example.txt  ×  ▶  □  ...
example.txt
1 hello world.

received_example.txt
1 hello world.
```