Socket Programming Training

📝 Overview

In this training, you will solve **three detective-themed cases** designed to teach the fundamentals of **socket programming**, **TCP stream handling**, and **HTTP request analysis**.

You will step into the shoes of a **network detective**, uncovering hidden clues, scanning ports, and dissecting raw HTTP requests to solve mysteries.

🛠️ Problem Statements

🕵️ Task #1: The Missing Message Mystery

🧩 Scenario

Alice tries to send `"HELLO WORLD"` to Bob's server. However, Bob only receives part of the message. As the detective, your job is to **investigate why the message is incomplete** and fix the issue so Bob receives the full string.

📂 Code Provided

Server (Bob – Victim):

```
# server.py
import socket

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(('localhost', 9999))
server.listen(1)
print("🏠 Bob's server is listening...")

client_socket, address = server.accept()
message = client_socket.recv(8)
print(f"📨 Bob received: '{message.decode()}'")
client_socket.close()

```

Client (Alice – Sender):

```
# client.py
import socket

client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(('localhost', 9999))
client.send("HELLO WORLD".encode())
print("📤 Alice sent: 'HELLO WORLD'")
client.close()
```

```
9
```

🎯 **Expected Outcome**

After fixing the server to correctly handle TCP streams:

1. Run `server.py`:

```
$ python3 server.py
```

2. Run `client.py`:

```
$ python3 client.py
```

3. Correct output should be:

```
📨 Bob received: 'HELLO WORLD'

```

🛡️ **Task #2: The Port Scanner Probe**

🧩 **Scenario**

A detective always checks locked and unlocked doors. Similarly, you will create a **simple port scanner** that tests whether a given port on a host is **open** or **closed**.

💡 Hints:

- Use `socket.connect()`.
- Wrap the connection attempt inside `try/except` blocks.

📂 **Example Input & Output**

Case 1: Input → Host: `localhost`, Port: `22`

```
Port 22 on localhost is Open

```

Case 2: Input → Host: `google.com`, Port: `8000`

```
Port 8000 on google.com is Close
```

📦 **Task #3: The Curl Detective Mystery**

🧩 **Scenario**

Detectives often record every clue exactly as found. Similarly, you will build a **Python server** (`server.py`) that accepts **HTTP requests sent via curl** and prints the **raw request exactly as received**.

👉 Important notes:

- Use only socket programming
- You do not need to send a response back to curl.
- Curl may display an error such as:

```
curl: (52) Empty reply from server
```

  This is expected.

- In some cases (large body requests), curl may also show:

```
curl: (56) Recv failure: Connection reset by peer
```

  This happens if the server closes the connection without reading the full request stream.

  Ensure you read the complete message using repeated `recv(BUFFER_SIZE)` calls.

📂 **Test Cases**

```
# 1. Simple GET
curl http://localhost:9090/test

# 2. POST with small body
curl -X POST http://localhost:9090/test -d "Detective at work"

# 3. POST with large body (forces multiple recv calls)
curl -X POST http://localhost:9090/bigdata -d "$(python3 -c "print('X'*5000)")"

```

🎯 **Sample Expected Output**

GET Request:

```
GET /test HTTP/1.1
Host: localhost:9090
User-Agent: curl/8.4.0
Accept: */*

```

POST (small body):

```
POST /test HTTP/1.1
Host: localhost:9090
User-Agent: curl/8.4.0
Accept: */*
Content-Length: 17
Content-Type: application/x-www-form-urlencoded

Detective at work

```

POST (large body):

```
POST /bigdata HTTP/1.1
Host: localhost:9090
```

```
User-Agent: curl/8.4.0
Accept: */*
Content-Length: 5000
Content-Type: application/x-www-form-urlencoded

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX...(continues for 5000 characters)

```

🎯 Deliverables

By the end of this training, you should submit:

1. **Task #1** → Fixed server code + screenshot/log showing `"HELLO WORLD"` fully received.
2. **Task #2** → A Python script (`scanner.py`) that scans `localhost:22` and prints whether it's open or closed.
3. **Task #3** → A Python server (`server.py`) that handles and prints **raw curl requests** (headers + body).