TASK 1: Question 1:

Briefly describe your code. Do not include code snippets. Rather, we are interested in your design and major functionality.

Response:

My code implements a simple HTTP server listens on a specified port and handles HTTP GET and HEAD requests. The server uses the socket library to establish a connection, allowing it to accept requests from clients, like web browsers as well as Postman, on the loopback address 127.0.0.1. The specified port I chose for my server is 8001. When a request is received, the server parses it to determine the HTTP method (GET or HEAD) and the requested file path. If the requested path is the root, "/", it returns a default HTML page I created that just prompts users to specify a file. For other paths, the server checks if the file specified by the path such as HelloWorld.html, exists and returns/displays it if found. The response includes key headers such as Date, Server, Last-Modified, Content-Length, and Content-Type to comply with HTTP standards. The code supports persistent connections by keeping the server socket open indefinitely, while the client socket closes after each request, simulating a persistent HTTP server without implementing timeouts. If a requested file is missing, the server responds with a "404 Not Found" message. For unsupported HTTP methods, it sends a "405 Method Not Allowed" response. This design enables the server to respond appropriately based on the type of request and the availability of the requested file.

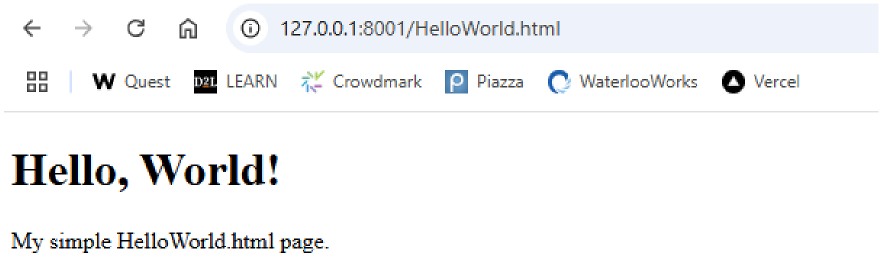
TASK 1: Question 2:

Show screenshots of your client browser demonstrating:

1. That you have received the contents of the HTML file from your server
2. That you have received a 404 not found error for a file that does not exist
3. That you have successfully used Postman for both GET and HEAD requests

Response:

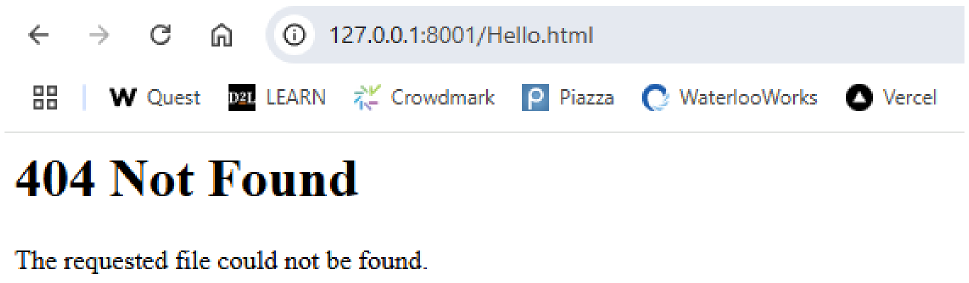
1.



Hello, World!

My simple HelloWorld.html page.

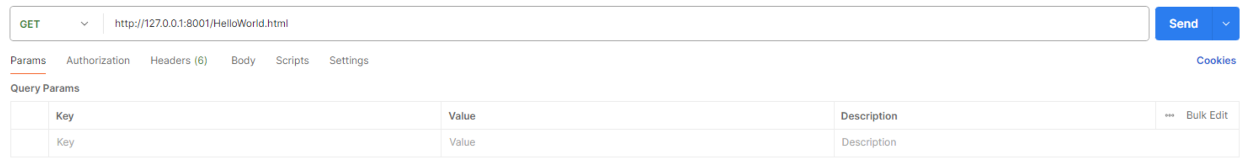
2.



404 Not Found

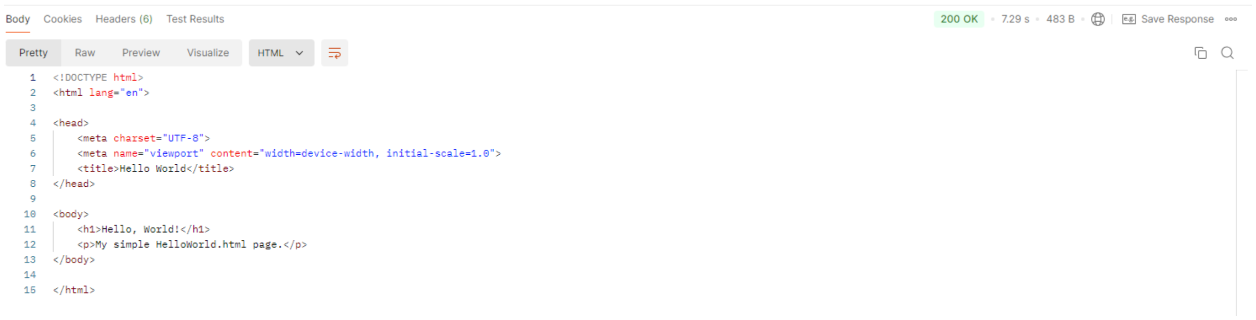
The requested file could not be found.

3.



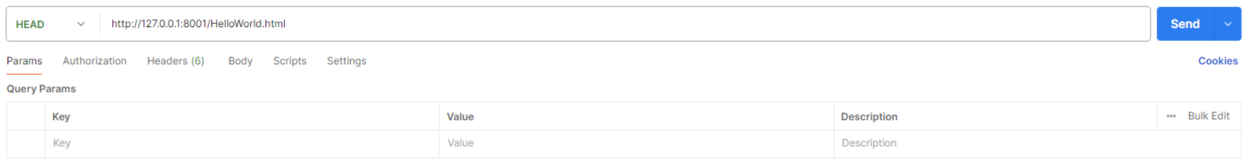
GET http://127.0.0.1:8001/HelloWorld.html

Key	Value	Description
Key	Value	Description



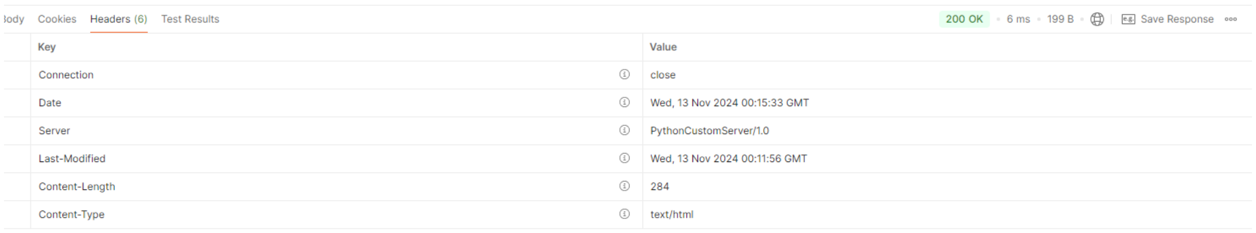
200 OK · 7.29 s · 483 B

```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Hello World</title>
</head>

<body>
    <h1>Hello, World!</h1>
    <p>My simple HelloWorld.html page.</p>
</body>

</html>
```



HEAD http://127.0.0.1:8001/HelloWorld.html

Key	Value	Description
Key	Value	Description



200 OK · 6 ms · 199 B

Key	Value
Connection	close
Date	Wed, 13 Nov 2024 00:15:33 GMT
Server	PythonCustomServer/1.0
Last-Modified	Wed, 13 Nov 2024 00:11:56 GMT
Content-Length	284
Content-Type	text/html

TASK 2: Question 1:

Briefly describe your code. Do not include code snippets. Rather, we are interested in your design and major functionality.

Response:

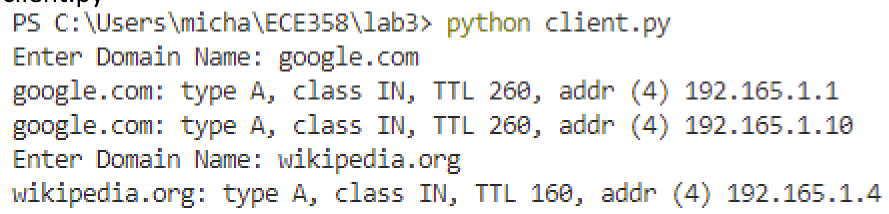
The DNS server I designed is structured to handle DNS requests by listening for incoming queries over UDP, interpreting the request data, and crafting appropriate responses based on a set of preconfigured domain records. When a request arrives, my server.py file first parses out essential components from the DNS query, such as the transaction ID (AKA query ID), domain name, record type, and record class. In my code, I break down both the DNS header and question sections before converting the received questions into strings that can be interpreted by my server. After parsing the request, the server then references my ‘dnsRecords’ variable that contains each domains respective information, such as its IP addresses, TTL, type, and class. If a match for the requested domain is found, the server assembles a DNS response packet, embedding the original transaction ID for consistency, along with flags and data. If the domain is not found, the server generates a response indicating a failure to resolve the domain name, using the appropriate error codes as per DNS standards. Finally, the server responds to the client by sending the crafted packet back through the same UDP socket, ensuring that the client receives the answer in the expected format. This explanation was inspired by explanations given by ChatGPT.

TASK 2: Question 2:

Show screenshots of both client and server terminals. The client screenshot must include two inputs: google.com and wikipedia.org. The screenshots must show the respective output.

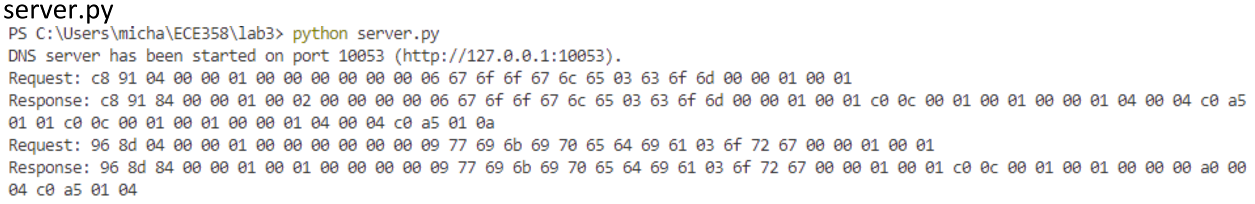
Response:

client.py



```
PS C:\Users\micha\ECE358\lab3> python client.py
Enter Domain Name: google.com
google.com: type A, class IN, TTL 260, addr (4) 192.165.1.1
google.com: type A, class IN, TTL 260, addr (4) 192.165.1.10
Enter Domain Name: wikipedia.org
wikipedia.org: type A, class IN, TTL 160, addr (4) 192.165.1.4
```

server.py



```
PS C:\Users\micha\ECE358\lab3> python server.py
DNS server has been started on port 10053 (http://127.0.0.1:10053).
Request: c8 91 04 00 00 01 00 00 00 00 00 00 06 67 6f 6f 67 6c 65 03 63 6f 6d 00 00 01 00 01
Response: c8 91 84 00 00 01 00 02 00 00 00 00 06 67 6f 6f 67 6c 65 03 63 6f 6d 00 00 01 00 01 c0 0c 00 01 00 01 00 00 01 04 00 04 c0 a5
01 01 c0 0c 00 01 00 01 00 00 01 04 00 04 c0 a5 01 0a
Request: 96 8d 04 00 00 01 00 00 00 00 00 00 09 77 69 6b 69 70 65 64 69 61 03 6f 72 67 00 00 01 00 01
Response: 96 8d 84 00 00 01 00 01 00 00 00 00 09 77 69 6b 69 70 65 64 69 61 03 6f 72 67 00 00 01 00 01 c0 0c 00 01 00 01 00 00 00 a0 00
04 c0 a5 01 04
```

TASK2: Question 3:

For a query to youtube.com, produce a table for the query HEX messages and another table for the response HEX messages. See Table 2 in the lab manual for format. The table must be a proper MS Word table in the response section below:

Response

Request: 6f 6d 04 00 00 01 00 00 00 00 00 00 07 79 6f 75 74 75 62 65 03 63 6f 6d 00 00 01 00 01

Response: 6f 6d 84 00 00 01 00 01 00 00 00 00 07 79 6f 75 74 75 62 65 03 63 6f 6d 00 00 01 00 01 c0 0c 00 01 00 01 00 00 00 a0 00 04 c0 a5 01 02

Query

Type	Key	Value
DNS HEADER	ID	13 7c
	FLAGS	04 00
	QDCOUNT	00 01
	ANCOUNT	00 00
	NSCOUNT	00 00
	ARCOUNT	00 00
QUERY	QNAME	07 79 6f 75 74 75 62 65 03 63 6d 00
	QTYPE	00 01
	QCLASS	00 01

Response

Type	Key	Value
DNS HEADER	ID	13 7c
	FLAGS	84 00
	QDCOUNT	00 01
	ANCOUNT	00 00
	NSCOUNT	00 00
	ARCOUNT	00 00
QUERY	QNAME	07 79 6f 75 74 75 62 65 03 63 6d 00
	QTYPE	00 01
	QCLASS	00 01
RESOURCE RECORD (ANSWER SECTION)	NAME	c0 0c
	TYPE	00 01
	CLASS	00 01
	TTL	00 00 00 a0
	RDLENGHT	00 04
	RDATA	c0 a5 01 02

TASK 3: Question 1:

In your own words, with proper referencing, discuss:

a. What is a socket, and how does it work?
b. Why are sockets required in modern TCP/IP networking? Are there alternatives?
c. What is the DNS and how does it work?

Response:

a.

A socket acts as an endpoint for sending or receiving data across a network. They are integral to networking because they mask the complications of an underlying network protocol, making it easier for programs to exchange information. Sockets operate by establishing a connection between two devices using an IP address and a port number. In the case of task 1, the IP address is 127.0.0.1 and the port number is 8001. For task 2, the IP address is 127.0.0.1 and the port number is 10053. When a socket is created, it can either be in a listening state (waiting for a connection such as the case for server.py) or in an active state (initiating a connection as seen in client.py). For example, a server sets up a socket to listen on a specific port, and when a client connects to that port, the server's socket accepts the connection and opens a pathway for communication. Sockets support different communication protocols like TCP (for reliable connections) and UDP (for faster, connectionless data transmission).

b.

Sockets are fundamental to TCP/IP networking because they allow the communication between devices on a network in a standardized way. By providing a consistent interface for sending and receiving data, sockets allow applications to connect over networks without requiring detailed knowledge of lower-level networking protocols used by the server. They make it possible for programs to reliably transmit data by handling important functions like packet segmentation, IP addressing, and even retransmissions (in the case of TCP). Sockets allow developers to build networked applications such as web browsers, messaging apps, and file transfer services that function effectively over the internet. While sockets are a primary tool for networked communication, there are alternatives such as Remote Procedure Calls (RPCs) or message-oriented middleware that enable interaction between distributed systems with less direct socket management. These alternatives handle some of the network operations internally, which can make application development easier in specific contexts since it may result in a partially standardized procedure. Despite the alternatives, sockets remain the most popular and commonly used method of TCP/IP communication due to their flexibility and performance.

c.

The Domain Name System (DNS) is a hierarchical naming system that translates human-friendly domain names (like google.com) into IP addresses that computers use to identify each other on a network. Without DNS, users would have to remember IP addresses for every website and human-friendly domain names would be useless, making the internet far less accessible for computers, also known as clients. The DNS operates as a distributed database where different servers hold parts of the information. When a user enters a domain name, the computer queries a DNS server. If that server doesn't have the domain's IP address, it forwards the request to other DNS servers up the hierarchy until it reaches an authoritative server for that domain. This server returns the corresponding IP address, which is then used by the client to establish a connection with the website's server.

References

1. Socket - low-level networking interface. Python Documentation. (n.d.). https://docs.python.org/3/library/socket.html
2. McMillan, G. (n.d.). Socket programming howto. Python Documentation. https://docs.python.org/3/howto/sockets.html#socket-howto
3. Ostapiuk, R. (n.d.). Introduction to TCP/IP (part 4) - sockets and Ports. Introduction to TCP/IP (Part 4) - Sockets and Ports - Developer Help. https://developerhelp.microchip.com/xwiki/bin/view/applications/tcp-ip/sockets-ports/#:~:text=A%20socket%20is%20a%20software,to%20the%20socket's%20receive%20buffer
4. Kohler, M. (2024, January 15). Websocket alternatives for Realtime Communication. PubNub. https://www.pubnub.com/blog/websockets-alternatives-for-realtime-communication/
5. Gillis, A. S., Rosencrance, L., & Matturro, B. (2024, May 13). What is remote procedure call (RPC)?: Definition from TechTarget. Search App Architecture. https://www.techtarget.com/searchapparchitecture/definition/Remote-Procedure-Call-RPC
6. What is DNS? | how DNS works | cloudflare. Cloudfare. (n.d.). https://www.cloudflare.com/learning/dns/what-is-dns/
7. What is domain name system (DNS)?. Fortinet. (n.d.). https://www.fortinet.com/resources/cyberglossary/what-is-dns

TASK 3: Question 2:

Draw the flow diagram of TCP Socket Programming that you used in this lab.

Response:

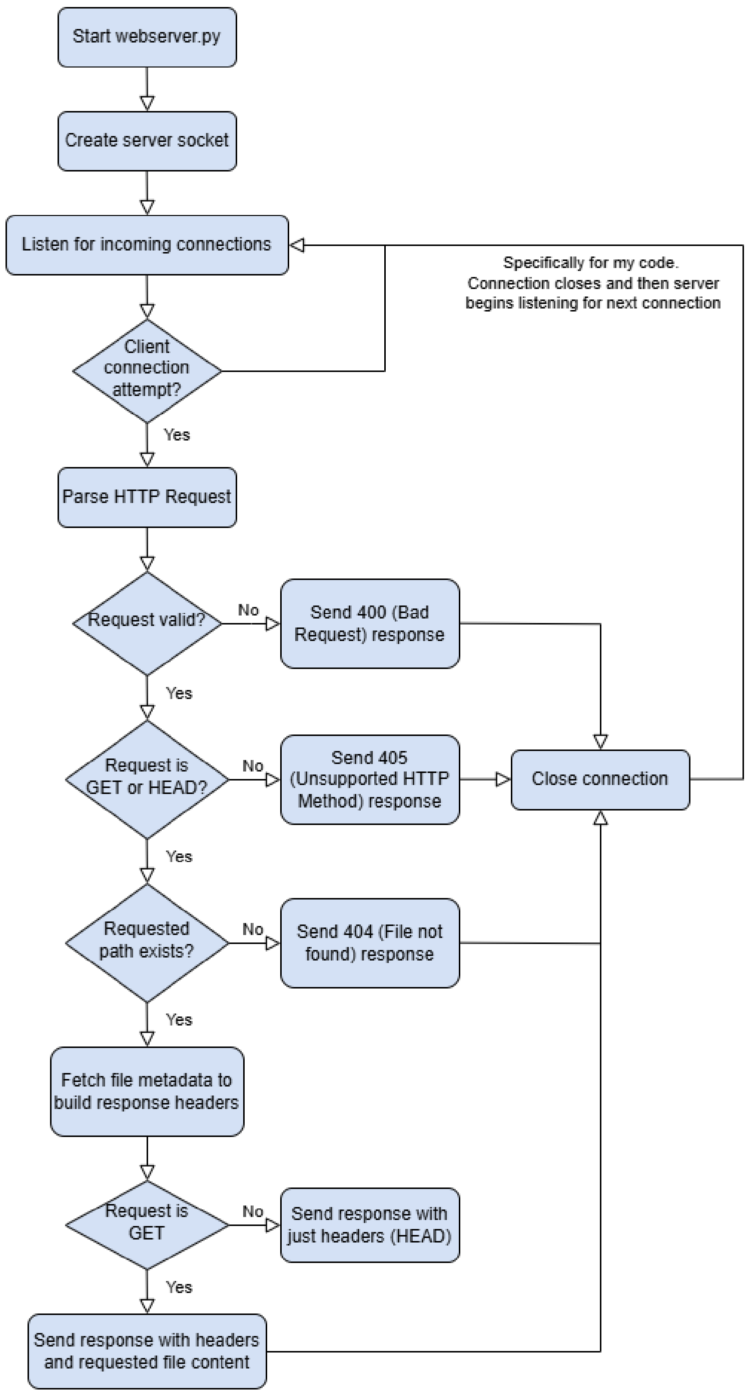

Start webserver.py
Create server socket
Listen for incoming connections
Specifically for my code. Connection closes and then server begins listening for next connection
Client connection attempt?
Yes
Parse HTTP Request
Request valid?
No
Send 400 (Bad Request) response
Yes
Request is GET or HEAD?
No
Send 405 (Unsupported HTTP Method) response
Close connection
Yes
Requested path exists?
No
Send 404 (File not found) response
Yes
Fetch file metadata to build response headers
Request is GET
No
Send response with just headers (HEAD)
Yes
Send response with headers and requested file content

Diagram created with draw.io and based on my TCP code from Task 1