Programming Assignment
CSE 342
Choose one

Assignment: Build a Basic Web Server in Python

Objective:

In this assignment, you will create a simple web server using Python. The purpose of this project is to deepen your understanding of web server fundamentals, HTTP protocols, and handling client requests in a server environment.

Requirements:

- You must create a basic HTTP web server that listens for incoming connections from a web browser or HTTP client.
- The server should be capable of responding to at least GET requests with appropriate status codes (e.g., 200 OK, 404 Not Found).
- The web server should be able to serve basic HTML files from a directory.
- Include handling for at least two different types of content (e.g., text/html and text/plain).
- The server should log each request, showing the request method, URL, and response status.
- The server should also handle timeouts

Instructions:

1. **Set up the Web Server:**
   - Use Python's built-in socket library to create the server.
   - The server should listen on a specified port (e.g., 8080) and handle incoming connections.
2. **Handle HTTP GET Requests:**
   - When a client requests a specific URL, return the appropriate HTML file or text response.
   - The server should return a 200 status code for valid requests and a 404 status code for missing pages.

3. **Serving Static Files:**
    - Create a directory called static/ in your project folder.
    - The server should look for HTML or text files in this directory and serve them based on the client's request.
4. **Logging Requests:**
    - For every request made to the server, log the following information:
        - Request method (GET, POST, etc.)
        - Requested URL
        - Status code of the response
5. **Handling Errors:**
    - If a requested file is not found, return an appropriate 404 error page.

Bonus Features (Optional):

- Implement support for additional HTTP methods (e.g., POST, PUT).
- Add support for serving other types of files (e.g., images or CSS).
- Create a simple HTML page that interacts with your server using JavaScript (e.g., by making AJAX requests).

Submission:

- Submit your Python code along with a brief README file that explains how to run the server and test it.
- Include any sample HTML files or other static content that should be served by the web server.
- Please include CLI output of the code starting, as well as example client requests

Grading Criteria:

- Functionality: Does the server handle GET requests and serve files correctly?
- Code Quality: Is the code clean, well-organized, and appropriately commented?
- Logging: Does the server log the necessary details for each request?
- Error Handling: Does the server correctly handle errors and provide meaningful responses?

Assignment: Implement a Basic DNS Server in Python

Objective:

In this assignment, you will design and implement a basic Domain Name System (DNS) server using Python. This project will give you hands-on experience with DNS protocols, packet structures, and network programming.

Task:

Your goal is to build a DNS server that can resolve domain names to IP addresses. You will use Python to handle DNS queries, construct DNS responses, and follow the DNS protocol. The server will act as a recursive resolver, querying external DNS servers if it does not have the answer cached.

Requirements:

1. **DNS Query Handling:**
   - The server must listen for incoming DNS requests on UDP port 53.
   - Upon receiving a query, the server should parse the DNS packet to extract the domain name being queried.
2. **Recursive DNS Resolution:**
   - If the server cannot resolve the domain locally (i.e., it doesn't have a cached entry), it should forward the request to an upstream DNS server (like Google's DNS at 8.8.8.8).
   - Receive the response from the upstream server and forward the answer back to the original client.
3. **Packet Structure:**
   - Your server must correctly handle the DNS packet structure, including the header, question, and answer sections.
   - Ensure the server responds with the appropriate DNS records (e.g., A records for IPv4 addresses).
4. **Caching:**
   - Implement basic caching so that once a domain name is resolved, the result is stored in memory for future requests.

- Entries should expire after a certain Time-To-Live (TTL) value.

5. **Logging:**
   - Log each DNS request, including the queried domain name, timestamp, and whether the result was served from cache or resolved via an external server.

Bonus (Optional):

- Support additional DNS record types such as, CNAME, or MX records.
- Implement rate limiting to prevent abuse from too many requests from a single client.

Submission:

- Your Python code for the DNS server.
- A README file that explains:
  - How to run the server.
  - How to test the server (e.g., using tools like nslookup or dig).
  - Any design decisions or features you've implemented.
- Include CLI output of the code starting, as well as example client requests

Grading Criteria:

- **Correctness:** Does the server correctly handle DNS requests and respond appropriately?
- **Code Quality:** Is the code well-organized, commented, and efficient?
- **Caching Implementation:** Does the caching mechanism work as expected and respect TTL values?
- **Error Handling:** Does the server handle errors (e.g., malformed requests or unreachable external DNS servers) gracefully?

Tools:

- Python's socket library for handling UDP and network communication.
- External tools like dig or nslookup to test your DNS server.