THE UNIVERSITY OF TEXAS AT DALLAS

Erik Jonsson School of Engineering and Computer Science
CS/CE 4390.002: Computer Networks, Fall 2025

Objective

You will build a fully functional HTTP/1.0 web server from scratch using Python sockets. The server should be able to:

- Accept and parse HTTP requests.
- Handle the four basic HTTP methods:
  - GET – Retrieve a file.
  - HEAD – Same as GET, but without sending the body.
  - POST – Upload a file to the server.
  - PUT – Update/replace an existing file on the server.
- Return proper HTTP responses (200 OK, 404 Not Found, 403 Forbidden, etc.).
- Transfer files only if the file is world-readable.
- Default GET / should map to index.html.
- Support concurrent requests using multithreading.
- Maintain visitor tracking via cookies (visit count, last visit time).
- Prevent denial of service (DoS) attacks by banning IPs with >100 requests/minute.
- Include a custom HTTP client program for testing uploads, downloads, and DoS attacks.

Server Specification

1. **Setup & Execution**
   Run the server with:
   `python server.py <port>`
   The server will listen on `<port>`. All server files should be stored inside a directory named Upload.
2. **Request Handling**
   For each client connection, accept TCP connection. parse the HTTP request (ensure it's well-formed) and check if client IP is banned (deny if true). Processes based on method:
   - GET → Return requested file with headers.
   - HEAD → Return headers only.
   - POST → Save uploaded file into Upload/.
   - PUT → Replace an existing file in Upload/.

   Return proper error messages if:
   - File not found → 404 Not Found.
   - Permission denied → 403 Forbidden.

   Close the connection after serving the request.
3. **Multithreading**
   Use Python threading to serve multiple clients simultaneously. The main thread listens for connections. Each request is handled by a separate worker thread.
4. **Cookies & Visitor Tracking**
   Maintain a Python dictionary (or database) of visitors:
   Browser/IP → Visit count + Last visit time.

Use locks to synchronize updates. On server shutdown, save the visitor database to visitors.json (or CSV). On server startup, reload the database.

5. **DoS Protection**
   Track number of requests per IP per minute. If an IP exceeds 100 requests/minute, ban it temporarily (until restart).

Client Specification

1. **Setup & Execution**
   Run the client with:
   `python client.py <serverHost> <serverPort> <filename> <command> [options]`

   - <command> = GET | HEAD | POST | PUT
   - Optional DoS mode:
     `python client.py <host> <port> <file> GET -d 200`
   - Sends 200 rapid requests to test DoS protection.
2. **Behavior**
   - For GET → Save file into Download/ with the same name.
   - For HEAD → Print only headers.
   - For POST/PUT → Upload a file from Download/ to the server.
   - Print server response to console.

Submission

You will be asked to submit a zip/tar folder containing the web server implementation, the HTTP client implementation and instructions for running your server and client.

Rubric

- Basic Server (GET, HEAD)
- POST & PUT (File Upload/Update)
- Error Handling (404, 403, bad request)
- Multithreading
- Cookies & Visitor Tracking
- DoS Protection
- Custom Client

Breakdown will be published later.