ECEN 602 Machine Problem 4
HTTP v1.0 Proxy Server & Client with Caching

Team 11:
1. Dhanraj Murali – UIN:734003894
2. Swetha Reddy Sangem – UIN: 433003591

Implementation Roles:
1. Dhanraj – Client, Documentation, Testing
2. Swetha – Server, Caching

Both were involved in the ChatGPT part of the submission. Same has been executed and tested with Hera servers.

Bonus feature (Last modified) has also been implemented.

Submission:
1. Submission 1 – Source Code/Make file written by the team.
2. Submission 2 – Source Code written by team and optimized by ChatGPT/Makefile.
3. Submission 3 – Source Code/Make file generated by ChatGPT.
4. README.pdf
5. Test case screenshots as PDF

Execution:
Run the following commands in order,

Open 2 terminals,
1. Terminal 1: make clean; make
2. Terminal 1: ./proxy <IP_address> <port_number >
3. Terminal 2: ./client <IP address> <port number> <URL>

Design of HTTP Server:

Proxy Server:
Proxy Server runs on a user defined port and acts as relay between the client and the web server. It listens to incoming HTTP requests from client. Based on the request, it first tries to identify from the cache if it is present and in case the requested page is cached already then it will be served to the client from the cache. Cache is implemented in the LRU strategy. In case, there is a cache miss the request is then relayed to the web server using port 80 and then it is read and cached by the proxy server and then served to the client.

Client:
The client takes the requested URL as a command line input. It then operates on the particular input to split the requested URL into host name and file name. Then the client code formulates a HTTP Get request based on the input and then relays it to the server on the specified IP and port. It then receives the request and stores it in the same directory from where it is executed. In case the file name is not mentioned it is saved as “default.html” file.

CHAT-GPT Optimization Comments:

- Enhanced LRU Cache Management: Implemented efficient LRU cache management to optimize cache hit rates and minimize cache replacement overhead.
- Improved Client Connection Handling: Streamlined client connection handling to reduce latency and improve responsiveness, allowing the server to handle multiple clients simultaneously.
- Streamlined Data Processing: Optimized data processing and response generation to provide faster content delivery and minimize resource consumption.
- Performance and Efficiency: Focused on performance enhancements to reduce response times and improve overall system efficiency.
- Code Organization: Structured the code for clarity and maintainability, making it easier to understand and extend while maintaining high performance.

References:
[1] Unix Network Programming, Volume 1, The Sockets Networking API, 3rd Edition
[2] Beej’s Guide to Network programming