# PROJECT AND TEAM INFORMATION

## Project Title

(Try to choose a catchy title. Max 20 words).

|  |
| --- |
| Multi-threaded proxy web server |

## Student / Team Information

|  |  |
| --- | --- |
| Team Name:  Team # |  |
| **Team member 1 (Team Lead)**  (Last Name, name: student ID: email, picture): | Shubham Negi – 22021844  snegi4260@gmail.com |
| **Team member 2**  (Last Name, name: student ID: email, picture): | Neha Dobriyal - 220221247  nehadobriyal70@gmail.com |
| **Team member 3**  (Last Name, name: student ID: email, picture): | Gokul Singh – 220211572  Jaie6252@gmail.com |
| **Team member 3**  (Last Name, name: student ID: email, picture): | Rohit Kumar – 22021569  rohitraj.oficial03@gmail.com |

# PROPOSAL DESCRIPTION (10 pts)

## Motivation (1 pt)

(Describe the problem you want to solve and why it is important. Max 300 words).

|  |
| --- |
| The growing demand for fast and efficient web communication has made performance and reliability critical in modern network systems. Repeated requests to the same server can lead to increased latency, excessive bandwidth consumption, and stress on the main server— especially during peak loads or targeted disruptions like Denial-of-Service (DoS) or Man-in-the-Middle (MITM) attacks.  With billions of users accessing regionally restricted content or suffering from high latency, a proxy server also plays a key role in overcoming geoblocking and network bottlenecks—thus reinforcing both accessibility and performance.  This project addresses those challenges by implementing a multithreaded proxy web server with an LRU (Least Recently Used) cache. By intercepting client requests and serving cached responses when available, the system significantly reduces response time and minimizes unnecessary load on the main server. When a response isn’t cached, the proxy fetches it, stores it locally, and returns it to the client, improving future response efficiency.  Acting as an intelligent intermediary, the proxy server enhances responsiveness, supports scalability, and provides a robust mechanism to handle high request volumes. It offers practical benefits for real-world deployments where low latency, bandwidth efficiency, and system stability are critical. |

## State of the Art / Current solution (1 pt)

(Describe how the problem is solved today (if it is). Max 200 words).

|  |
| --- |
| Today, web browsers implement local caching mechanisms to improve performance. However, this client-side caching is limited in scope, often bypassed in private sessions or restricted networks, and doesn't scale across users.  Enterprise-grade proxy servers partially address these concerns but are typically optimized for large-scale systems and offer limited customization. They function as black-box tools, not ideal for learning or low-level debugging. Additionally, they lack fine-grained control over cache eviction policies or concurrency control.  This project bridges the gap by building a lightweight, educational multithreaded proxy server with a transparent and tunable LRU caching layer. It provides a solid foundation for understanding web proxy behavior, efficient request handling, and system-level programming while delivering real-world performance benefits. |

## Project Goals and Milestones (2 pts)

(Describe the project general goals. Include initial milestones as well any other milestones. Max 300 words).

|  |
| --- |
| To build a multithreaded proxy web server in C that efficiently handles HTTP requests and leverages an LRU cache to optimize performance for repeated requests. The system aims to reduce latency, optimize bandwidth, and act as an intermediary between clients and target servers.  Initial Milestones:   * Implement a basic proxy server using socket programming to handle requests/responses * From multiple clients . * Introduce multithreading to support concurrent client handling in our server . * Design and implement an LRU cache using a combination of doubly linked list and hash map for fast operations * Implement thread-safe concurrency control using mutexes or semaphores to protect shared cache data from race conditions * Support basic HTTP methods like GET and correct forwarding and response delivery   Further Milestones:   * Extend cache support for larger content and dynamic pages * Add access logging and performance monitoring features * Test under simulated high-traffic conditions to evaluate load handling and stability * Allow configurable settings like cache size and maximum threads for flexibility |

## Project Approach (3 pts)

(Describe how you plan to articulate and design a solution. Including platforms and technologies that you will use. Max 300 words).

|  |
| --- |
| We adopt a modular system-level approach to build a robust and efficient proxy server in C. The design emphasizes low-level networking, memory management, concurrency, and caching.  Technologies & Concepts:   * C Standard Programming : For memory management and data handling standard C methods like Malloc , free ( memory management) and string handling ( for data caching) are to be used. * Socket Programming : C based highly optimised libraries like sys.socket.h is to be used to open and maintain a TCP connection , this library also allows use to use multiple socket related functionalities that we need. * Pthreads : C based multi threading libraries like pthread.h can be used for implementing multithreading to serve multiple client at one using worker threads , this also provides methods to synchronize these threads via classic synchronization method like Mutex and Read-Write-Lock. * Plateform : Linux (e.g., Ubuntu) for full control over our code , compiled with GCC for its robust POSIX support.   Implementation Strategy:   1. Socket Programming The proxy server listens for client HTTP requests via a TCP sockets called the Main Listening Socket. Upon making the TCP connection via handshake it used worker threads to make a Connection Socket per client request , these Connection Sockets remember what unique client   That they are serving , this leaves the Main Listening Thread alone to get ready to listen to another client.   1. Multithreaded Architecture Each client request is handled by a separate thread to ensuring non-blocking behaviour and efficient processing of concurrent users. 2. LRU Cache Each thread uses a cache of responses to first look if the data requested is already present or not .Responses are cached using the Least Recently Used (LRU) eviction policy. When the cache exceeds its size limit, the least recently accessed item is evicted. 3. Thread Safety Shared access to cache is protected using mutex locks or a Read-Write-lock, preventing data corruption due to race conditions. 4. Each thread after getting the data wither form cache or from server is given back to the client. 5. End-to-End Flow    * Client sends request    * Proxy checks cache    * If cache hit → serve cached response    * If miss → fetch from origin, update cache, return to client   This platform-independent proxy design highlights key systems programming skills including synchronization, efficient memory access, and thread lifecycle management. |
|  |

## System Architecture (High Level Diagram)(2 pts)

(Provide an overview of the system, identifying its main components and interfaces in the form of a diagram using a tool of your choice).

|  |
| --- |
| Summary Flow:   1. Clients (c1, c2) initiate GET requests (e.g., for google.com, facebook.com). 2. The proxy server receives these requests via socket connections . 3. The proxy then spawns threads within a multithreaded web server to handle each request concurrently. 4. Each thread:    * Checks the LRU (Least Recently Used) cache to see if the requested content is already stored.    * If found → the cached response is returned immediately.    * If not found → the proxy connects to the actual server (e.g., Google), fetches the content, caches it (evicting least recently used if needed), then returns the response. 5. Semaphores (sem\_wait(), sem\_signal()) are used to synchronize access to shared resources (like the cache), ensuring thread safety in concurrent environments. |

## Project Outcome / Deliverables (1 pts)

(Describe what are the outcomes / deliverables of the project. Max 200 words).

|  |
| --- |
| Project Outcome / Deliverables (1 pt)  By the end of this project, we deliver a fully functional and optimized multithreaded proxy server with intelligent caching capabilities.  Deliverables Include:   * A proxy server that supports concurrent client connections using POSIX threads * A thread-safe LRU cache implemented from scratch using doubly linked list + hash map * Complete GET method support and proper HTTP forwarding * A clean and modular codebase, suitable for educational use and extensibility * Functional on Linux systems, tested with tools like curl, wget, and browser-based requests * Logs to monitor cache hits/misses and request handling performance * Optionally configurable cache size and thread limits allowing the proxy to efficiently relay responses between clients and web servers while optimizing performance through intelligent caching. |

# Assumptions

( Describe the assumptions ( if any ) you are making to solve the problem. Max 100 words )

|  |
| --- |
|  The project runs on a Unix-like OS with support for POSIX threads and sockets.   All client requests are standard HTTP GET requests.   Only HTTP (not HTTPS) is supported in the current version.   Cache stores only static content (e.g., HTML, images, CSS).   Maximum thread count and cache size are fixed at compile time.   DNS resolution and network access are available.   No advanced features like HTTP pipelining or compression are required.   Input errors and malformed requests are minimal.   Users interact via browser or command-line tools like curl.   The system is used for local testing or educational purposes, not production. |

## References

(Provide a list of resources or references you utilised for the completion of this deliverable. You may provide links).

|  |
| --- |
| [GitHub - AlphaDecodeX/MultiThreadedProxyServerClient](https://github.com/AlphaDecodeX/MultiThreadedProxyServerClient)  [Free web proxy - browse fast & anonymously](https://proxyium.com/) |