### **1. Problem Statement:**

Goal of the project is to build a Proxy server that sits b/w client & internet

Handles multiple client requests concurrently, forward requests to server, get the responses from server and cache frequently accessed pages.

Caching enhances performance by reducing redundant network calls.

### **2. I used**

C language,

pthread library (for creating and managing threads), sockets(),

Semaphores- For controlling concurrent access,

mutex lock - For thread safety while working with shared resources

LRU cache - This has host, time it got accessed

### **3. To Give an Overall High-Level Flow:**

1. When we start the proxy server, this listens on given port using a socket, waits for incoming client connections.
2. When a client connects, a new thread is created using pthread\_create to handle the request (Semaphore is used to limit the concurrent clients to a safe maximum)
3. When a client sends an HTTP GET request - we will parse it through some custom HTTP PARSER and extract the fields like (method, path, version, port)
4. Before forwarding to remote server, proxy checks if the response is already present in the cache- if already present we will directly return that cached response.
5. If not forwards the request, receives the response, caches in the proxy server, and then returns the data to the user
6. Mutex lock is used to protect the cache contents when multiple clients access at same time

Multiple clients can be handled simultaneously without blocking.

### **5. Challenges Faced & How You Overcame Them:**

I faced challenges while

Handling multiple client connections without blocking or crashing the server.  
 **Resolved by:** Using pthread threads with proper semaphore control to manage concurrency

**Also while** Implementing a thread-safe shared cache that multiple threads can access and update.  
 **Solution:** Used a **mutex lock** to ensure only one thread can read/write to the cache at a time, and built a custom **LRU eviction policy** to maintain memory limits.

**"In the future, I plan to extend it by handling more HTTP methods like POST and PUT, and possibly logging analytics or Dashboards."**