Design Document

High-Level Architecture

The system follows a **Client-Proxy-Server** model. The proxy operates at the Application Layer (handling HTTP semantics) and the Transport Layer (managing TCP sockets).

Component Description

- **Listener (Main Thread):** Responsible solely for bind()ing to the port and accept()ing new connections. It hands off the resulting file descriptor to a worker thread immediately.
- **Worker Thread:** An ephemeral thread that manages the lifecycle of a single client request (Parse Filter Connect Relay Close).
- **Filter Engine:** A blocking logic component that acts as a gatekeeper before any upstream connection is attempted.
- **Relay Engine:** The data pump that moves bytes between the client socket and the upstream server socket.

Concurrency Model

Selected Model: Thread-per-Connection.

Rationale

- **Isolation:** In a proxy environment, upstream servers have varying response times. If we used a single-threaded event loop, a slow DNS resolution or a hanging server could block all other clients. Threads isolate these delays to the specific user experiencing them.
- **Simplicity:** The logic for a proxy is linear: Read Request Process Send Response. This maps perfectly to a procedural function running in a thread, making the code easier to maintain than a state-machine based approach (like epoll).
- **Standard Compliance:** The pthread library is the standard POSIX solution for concurrency, ensuring portability across Unix-like systems.

Data Flow

❖ **Incoming Request Handling**

1. **Ingress:** Client initiates a TCP Handshake.
2. **Buffering:** The Worker Thread reads the request into a 4KB buffer.
3. **Parsing:**
    - The request line is split to find the **METHOD** (GET, POST, CONNECT).

- The **URL** is parsed to extract the **HOSTNAME** and **PORT**.

4. **Filtering:** The Hostname is compared against the loaded blacklist.
    - *Match Found:* Abort flow, send 403 Forbidden.
    - *No Match:* Proceed to forwarding.

❖ **Outbound Forwarding (HTTP vs. HTTPS)**

- **HTTP Mode (Cleartext):**
    1. Resolve Hostname to IP (gethostbyname).
    2. Connect to Upstream Server.
    3. Send the Client's original request buffer.
    4. Read Upstream Response and write to Client in a while(bytes > 0) loop.
- **HTTPS Mode (Tunneling):**
    1. Detect CONNECT method.
    2. Connect to Upstream Server (usually Port 443).
    3. Send HTTP/1.1 200 Connection Established to Client.
    4. Enter **Blind Relay Mode**: Use select() to monitor both sockets. If data arrives on one, immediately write it to the other without inspection.

Error Handling & Limitations

Error Handling Strategy

- **Socket Failures:** If socket() or accept() fails, the error is logged to stderr, and the server continues running (fatal errors are isolated).
- **Upstream Unreachable:** If connect() to the target server fails (e.g., DNS error or timeout), the proxy constructs a standard 502 Bad Gateway HTTP response and sends it to the client.
- **Malformed Requests:** If parsing fails, the thread simply closes the connection to prevent undefined behavior.

Limitations

- **No Caching:** Every request fetches fresh data from the origin server; no local storage is implemented.
- **HTTP/1.0 vs 1.1:** The proxy does not actively manage Keep-Alive connections; it tends to close sockets after a transaction to free resources (closer to HTTP/1.0 behavior).
- **Security:** The proxy does not inspect the contents of HTTPS traffic (SSL Termination is out of scope), meaning it cannot filter specific URLs inside a secure session, only the domain name during the handshake.