C-Proxy: A Multithreaded Caching Proxy Server Low-Level Design (LLD)

This LLD breaks down the project into its logical modules, defining their responsibilities and interactions.

1. Modules and Components

The system is composed of several key modules:

- `Server Core (proxy_server.c)`: The main entry point and orchestrator.
- `Thread Pool (proxy_server.c)`: Manages concurrency.
- `LRU Cache (proxy_server.c)`: Handles in-memory caching.
- `Request Handler (proxy_server.c)`: Contains the business logic for processing requests.
- `Configuration Manager (proxy_server.c)`: Loads and manages settings.
- `Logger (proxy_server.c)`: Provides thread-safe logging.
- `HTTP Parser (proxy_parse.c)`: Decodes raw HTTP requests.

2. Class/Module Diagram

This diagram shows the relationships between the different modules.

3. Module Responsibilities & Interfaces

`Server Core`

- Responsibilities:
  - Initializes all other modules (Logger, Config, Cache, Thread Pool).
  - Sets up the main listening TCP socket.
  - Listens for and `accepts` incoming client connections.
  - Enqueues new connections into the `TaskQueue`.
  - Handles graceful shutdown on `SIGINT/SIGTERM`.
- Key Functions: `main()`, `signal_handler()`

`Thread Pool`

- Responsibilities:
  - Creates and manages a fixed number of worker threads.
  - Implements a thread-safe task queue (`TaskQueue`) using a mutex and condition variables to manage access.
- Data Structures:
  - `TaskQueue`: A circular buffer to hold client socket descriptors.
  - `pthread_t[]`: An array to hold thread identifiers.

- **Interfaces:**
    - `init_task_queue()`: **Initializes the queue.**
    - `enqueue_task(int client_socket)`: **Adds a new client to the queue (Producer).**
    - `dequeue_task()`: **Removes a client from the queue (Consumer).**
    - `worker_thread(void *arg)`: **The main function for each worker thread.**

`LRU Cache`

- **Responsibilities:**
    - **Provides fast, thread-safe storage for frequently accessed web content.**
    - **Implements the LRU eviction policy to manage a fixed-size memory space.**
- **Data Structures:**
    - `CacheNode`: **A struct containing the key, data, size, and pointers for the hash table and doubly-linked list.**
    - `LRUCache`: **The main struct holding the hash table, list head/tail, size, capacity, and a mutex.**
- **Interfaces:**
    - `create_cache()`: **Initializes the cache.**
    - `get_from_cache(const char *key)`: **Retrieves an item, moving it to the front of the list.**
    - `put_in_cache(const char *key, const char *data, size_t size)`: **Adds an item, evicting the LRU item if necessary.**

`Request Handler`

- **Responsibilities:**
    - **Acts as the main brain for a worker thread.**
    - **Reads the raw request and calls the `HTTP Parser`.**
    - **Routes the request based on its type (HTTP vs. HTTPS) and blacklist status.**
    - **Orchestrates the cache check and forwarding logic for HTTP requests.**
- **Interfaces:**
    - `handle_request(int client_socket)`: **The main entry point for a worker.**
    - `handle_http_request(...)`: **Logic for cacheable `GET` requests.**
    - `handle_connect_request(...)`: **Logic for non-cacheable `CONNECT` (HTTPS) tunnels.**

`HTTP Parser (proxy_parse.c)`

- **Responsibilities:**
    - **Takes a raw character buffer and parses it into a structured `ParsedRequest` object.**
    - **Correctly extracts the method, host, port, path, and version for both proxy-style and server-style requests.**
- **Data Structures:**
    - `ParsedRequest`: **A struct to hold the tokenized components of an HTTP request.**

- **Interfaces:**

- **Interfaces:**
    - `ParsedRequest_create()`: **Allocates a new request object.**
    - `ParsedRequest_parse(struct ParsedRequest *, const char *buf, int buflen)`: **The core parsing function.**
    - `ParsedRequest_destroy()`: **Frees all memory associated with a parsed request.**

Configuration Manager & Logger

- **Responsibilities:**
    - **Config: Reads key-value pairs from `proxy.conf` to set global variables like port and thread count.**
    - **Logger: Provides a single, thread-safe function to write formatted log messages to `proxy.log`.**
- **Interfaces:**
    - **`load_configuration(const char *filename)`**
    - **`load_blacklist(const char *filename)`**
    - **`log_message(const char* level, const char* format, ...)`**