Programming Project 2 Concept Memo

Authors: Anna Running Rabbit, Joseph Mills, Jordan Senko
Date: November 4, 2025

Project Overview

A ligh-weight, multi-threaded HTTP web server implemented with C and designed to support modern real-time communication workloads. The server will be capable of handling multiple concurrent client connections, demonstrating core OS concepts such as multi-threading, synchronization, socket programming, and low-level I/O within a networked environment.

Core Functionality

The server will:

- Accept and manage incoming HTTP requests on a specified port
- Use a thread pool to handle many simultaneous connection efficiently
- Parse HTTP GET requests and serve static resources such as HTML, CSS, JS, images, etc.
- Provide correct HTTP status responses:
    - 200 OK
    - 404 Not Found
    - 500 Internal Server Error
- Maintain thread-safe request processing via synchronization

A dedicated listener thread will accept client connections and assign them to worker threads via a producer-consumer request queue protected by mutexes and condition variables.

OS Concepts Demonstrated

Concept	**Description / API Usage**
Multi-threading	Use pthreads for concurrent request handling: *pthread_create()*, *pthread_join()*
Thread synchronization	Mutexes and condition variables to protect shared resources: *pthread_mutex_lock()*, *pthread_mutex_unlock(),* *pthread_cond_wait(), sem_wait(), sem_post()*
Socket programming	TCP/IP networking for HTTP

	communication: *socket(), bind(), listen(), accept(), read(), write()*
File system I/O	Reading static files to respond to HTTP requests: *open(), read(), close()*
Thread pooling	Worker pool for resource efficiency and improved throughput
IPC	Client-server communication through sockets

Implementation Details

- **Language**: C
- **Environment**: Linux VM
- **Technology**: POSIX threads, low-level socket API
- **Structure**: Modular design separating networking, HTTP parsing, file I/O and logging
- **Testing**: Stress testing concurrent requests, verifying HTTP response accuracy

Timeline

Week	**Task**
1 - Planning & Design	Requirements gathering, architecture design, define thread-pool model and request queue, set up development environment
2 - Core Networking & Threading	Implement socket setup, listener thread, thread pool creation, producer-consumer request queue with mutexes/condition variables
3 - HTTP Request Handling & File Serving	Implement GET request parsing, serve static files, return proper HTTP responses + status codes, begin logging module
4 - Synchronization, Error Handling & Testing	Add thread-safe I/O handling, improve error response system, concurrency stress testing, memory leak checks, performance evaluation
5 - Optimization & Extensions	Code refinement and optimization, documentation, optional extension development (e.g., chat feature groundwork, POST support, live stats page)

Optional Extensions

- **Basic Chat Functionality:**
    - WebSocket-like behaviour using persistent TCP sockets or long-polling to demonstrate real-time communication support
- HTTP POST support for simple user input forms
- Live statistics dashboard for connections and requests
- Support for persistent HTTP connections
- Robust logging & enhanced error handling for malformed requests and high-load conditions