Design Document

Multi-Threaded Web Server with IPC

[Student Name 1] (ID: [XXXXX]) Student Name 2>Student Name 2
(ID: [YYYYY])
Operating Systems - University of Aveiro

December 5, 2025

Contents

1 Architecture Overview

The system follows a multi-process, multi-threaded architecture designed to handle high concurrency.

1.1 Process Model

The application consists of a single **Master Process** and N **Worker Processes**:

- **Master:** Responsible for socket binding, accepting connections, and distributing them via a shared queue.
- **Workers:** Responsible for consuming requests from the queue and processing them using a thread pool.

Figure 1: High-Level Architecture Diagram

2 Shared Data Structures

Inter-Process Communication (IPC) is achieved through POSIX Shared Memory.

2.1 Memory Layout

The main shared structure `shared_data_t` contains:

```
typedef struct {
    request_queue_t queue;   // Circular buffer for socket FDs
    server_stats_t stats;    // Global statistics counters
    // ...
} shared_data_t;
```

2.2 Circular Buffer (Producer-Consumer)

The request queue is implemented as a bounded circular buffer (FIFO) located in shared memory. It uses `in` and `out` indices to manage insertions and removals.

3 Synchronization Mechanisms

3.1 Process Synchronization (Semaphores)

We use named POSIX semaphores to coordinate the Master and Workers:

- `SEM_MUTEX`: Ensures mutual exclusion when accessing the shared queue indices.
- `SEM_ITEMS`: Counts the number of pending requests (Signals Workers).
- `SEM_SPACES`: Counts the available slots in the queue (Signals Master).

3.2 Thread Synchronization (Mutex & Cond Vars)

Inside each Worker process, a Thread Pool handles requests. Synchronization is achieved via:

- `pthread_mutex_t`: Protects the internal worker queue/state.
- `pthread_cond_t`: Allows threads to sleep until new work arrives.

4 Component Design

4.1 LRU Cache

Each worker maintains a local Least Recently Used (LRU) cache for static files.

- **Structure:** Hash map for O(1) lookups + Doubly Linked List for eviction.
- **Concurrency:** Protected by `pthread_rwlock` to allow multiple simultaneous readers.

4.2 Thread-Safe Logging

A single log file is shared. Writes are serialized using a specific semaphore/mutex to prevent interleaved lines.

5 System Lifecycle

5.1 Initialization

1. Load configuration. 2. Create/Open Shared Memory and Semaphores. 3. Bind and Listen on TCP Port. 4. Fork Worker processes.

5.2 Shutdown (Graceful Exit)

Upon receiving `SIGINT`: 1. Master stops accepting connections. 2. Signals all workers to terminate. 3. Workers join all threads and exit. 4. Master unlinks shared memory and semaphores before exiting.

6 Flowcharts

Figure 2: Request Processing Flowchart