1. Introdução

Este documento de design descreve a arquitetura, módulos, modelo de concorrência e mecanismos de sincronização de um servidor web HTTP multithread.
O servidor é capaz de lidar com múltiplos clientes simultaneamente usando uma arquitetura multiprocessos e multithread.

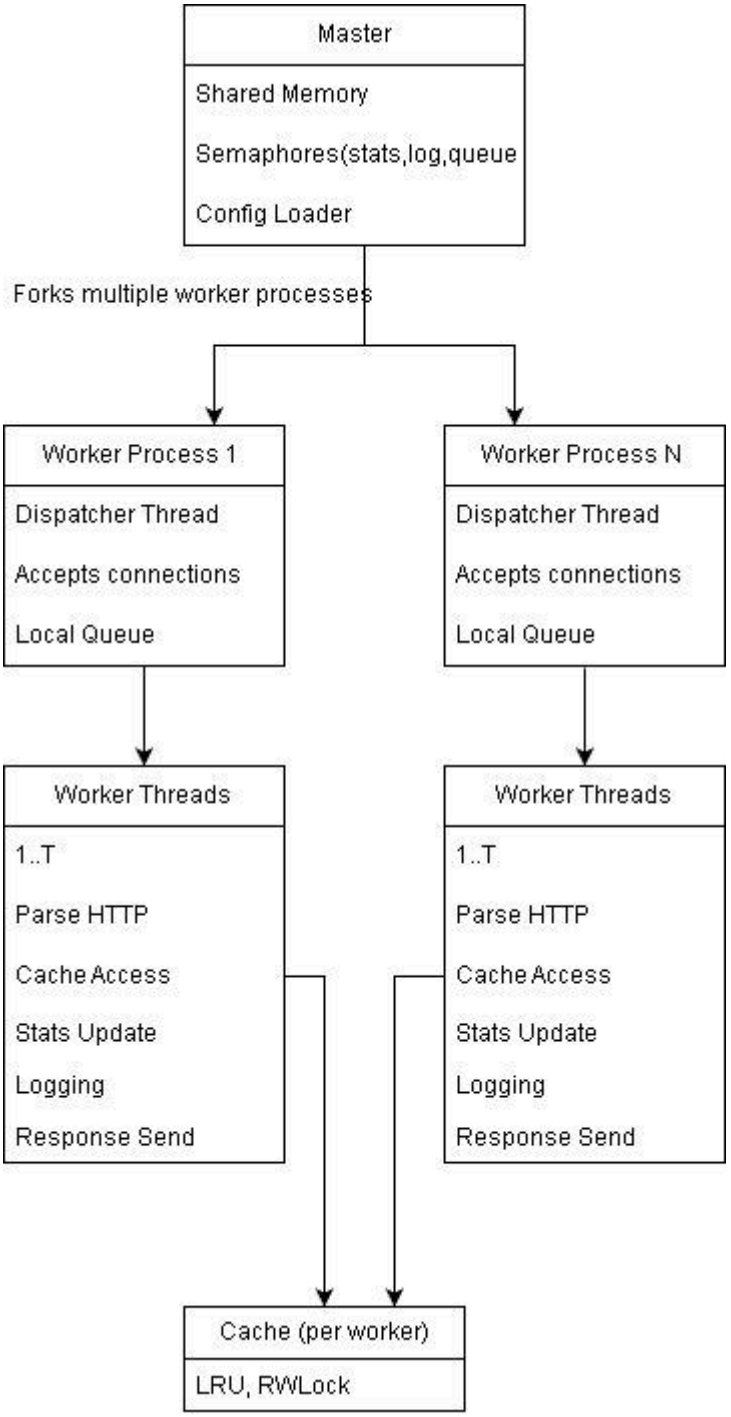
Principais funcionalidades incluem:

- Múltiplos processos *worker* com *thread pools*
- Filas de requisições limitadas usando o padrão produtor–consumidor
- Cache LRU em memória por *worker*
- Estatísticas e logging compartilhados com semáforos a nível de processo
- Conexões *keep-alive*, requisições HTTP *range* e endpoints especiais (/stats, /dashboard)

2. Arquitetura

A arquitetura do servidor consiste no processo *Master* e múltiplos processos *Worker*, cada um com um *thread pool* local.

Diagrama de arquitetura



3. Descrição dos Módulos

3.1 Processo Master

Responsabilidades:

- Carregar configuração do arquivo *server.conf*
- Criar memória compartilhada e semáforos para comunicação interprocessos
- Criar (fork) múltiplos processos worker
- Monitorizar workers (via SIGCHLD) e realizar desligamento gracioso (SIGINT/SIGTERM)

Recursos compartilhados:

- Memória compartilhada (*shared_data_t*) para estatísticas e fila de requisições
- Semáforos para logging, estatísticas e gerenciamento da fila de requisições

Funções principais:
master_main(), shutdown_handler(), cleanup_resources().

3.2 Processo Worker

Responsabilidades:

- Reabrir semáforos herdados do master
- Inicializar logging e cache
- Iniciar o thread pool (dispatcher + threads de trabalho)
- Lidar com desligamento gracioso (SIGTERM)

Funções principais:
worker_loop(), worker_shutdown_handler().

3.3 Thread Pool

Componentes:

- **Dispatcher thread**: aceita conexões e insere-as numa fila local limitada
- **Worker threads**: removem conexões da fila e tratam requisições HTTP

Fila local:

- Implementada com *pthread_mutex* + *pthread_cond*
- Suporta o padrão produtor–consumidor

Tratamento de requisições:

- Parse das requisições HTTP
- Servir arquivos estáticos usando *cache_t*
- Tratar endpoints especiais (/stats, /dashboard, /causeXXX)
- Atualizar estatísticas e registrar logs usando semáforos

Funções principais:
thread_pool_start(), dispatcher_thread(), worker_thread(), handle_client_request().

3.4 Módulo de Cache

Responsabilidades:

- Armazenar arquivos recentemente servidos para acesso rápido
- Remover entradas LRU quando o limite de memória é atingido

Concorrência:

- *pthread_rwlock* permite múltiplas leituras simultâneas
- Escritas (inserção/remoção) exigem *write lock*

Funções principais:
cache_get(), cache_put(), cache_init(), cache_destroy().

3.5 Módulo de Log (Logger)

Responsabilidades:

- Registrar entradas de log no estilo Apache
- Rotacionar o arquivo de log quando exceder 10 MB

Concorrência:

- Logging protegido por semáforo compartilhado entre processos (*sems->log*)

Funções principais:
create_logger(), destroy_logger(), log_request(), check_and_rotate_log().

3.6 Módulo HTTP

- Faz o parse das requisições e gera respostas
- Suporta:
    - Keep-alive (até 50 requisições por conexão)
    - Range requests (conteúdo parcial)
    - Respostas de erro (400, 403, 404, 500, 501, 503)

4. Fluxo da Requisição

Cliente
→ Dispatcher (processo worker)
→ Insere o *fd* na fila local
→ Worker thread remove o *fd*
→ Faz parse da requisição HTTP
→ Valida o caminho (/.., /causeXXX, /stats, /dashboard)
→ Busca no cache (*cache_get*)

→ Serve arquivo / gera resposta
→ Atualiza estatísticas (protegido por semáforo)
→ Registra log (protegido por semáforo)
→ Envia resposta

5. Concorrência & Sincronização

5.1 Recursos Compartilhados e Proteções

Recurso	Mecanismo de Proteção	Alvo
Fila de requisições	pthread_mutex + pthread_cond	Threads do worker
Estatísticas (shared_data_t)	sem_wait(sems->stats) / sem_post	Todos os workers + processo de stats
Logging	sem_wait(sems->log) / sem_post	Todos os workers
Cache	pthread_rwlock_rdlock/wrlock	Threads do worker

5.2 Padrões de Concorrência

Produtor–consumidor:

- Dispatcher = produtor
- Worker threads = consumidores

Leitores–escritores:

- Cache suporta múltiplos leitores simultâneos

Semáforos:

- Protegem estatísticas e logging entre processos

Desligamento gracioso:

- Dispatcher e threads verificam *worker_shutdown*
- Variáveis de condição sinalizam parada

6. Procedimento de Desligamento

Master recebe SIGINT/SIGTERM:

1. Define **shutdown_requested = 1**
2. Envia SIGTERM para todos os workers
3. Workers:
    - Ativam **worker_shutdown = 1**
    - Param dispatcher e threads
    - Fecham conexões e destroem logger/cache
4. Master limpa memória compartilhada e semáforos

7. Considerações de Desempenho

- Fila de requisições limitada previne sobrecarga
- Keep-alive reduz overhead de conexões TCP
- Cache minimiza I/O em disco
- Thread pool maximiza uso da CPU
- Uso equilibrado de semáforos e RW-lock garante eficiência e correção

8. Funcionalidades Especiais

- **/stats** → retorna estatísticas em JSON
- **/dashboard** → retorna dashboard HTML
- **/causeXXX** → endpoints de teste que geram erros HTTP específicos
- Suporte a Range Requests
- Rotação de log ao atingir 10 MB