Documento de Design
Servidor Web Multi-Thread

Membros:

Alexandre Regalado 124572

Cristiane Moreno 128076

Introdução

O sistema consiste num servidor HTTP implementado na linguagem C para distribuir ficheiros.

Este projeto foi desenvolvido no âmbito da cadeira de Sistemas Operativos, aplicando conceitos como processos, threads, sincronização e comunicação entre processos.

O projeto tem como objetivo a distribuicao de ficheiros (html, css, imagens) de forma concurrente e controlada.

O sistema implementa:

- Suporte ao protocolo HTTP com método GET
- Arquitetura prefork com múltiplos processos worker
- Pool de threads por worker para a paralelizada de procesamiento
- Shared queue para distribuição de conexões
- Cache LRU de ficheiros estáticos
- Estatísticas globais em memoria partilhada
- Endpoints de monotorização

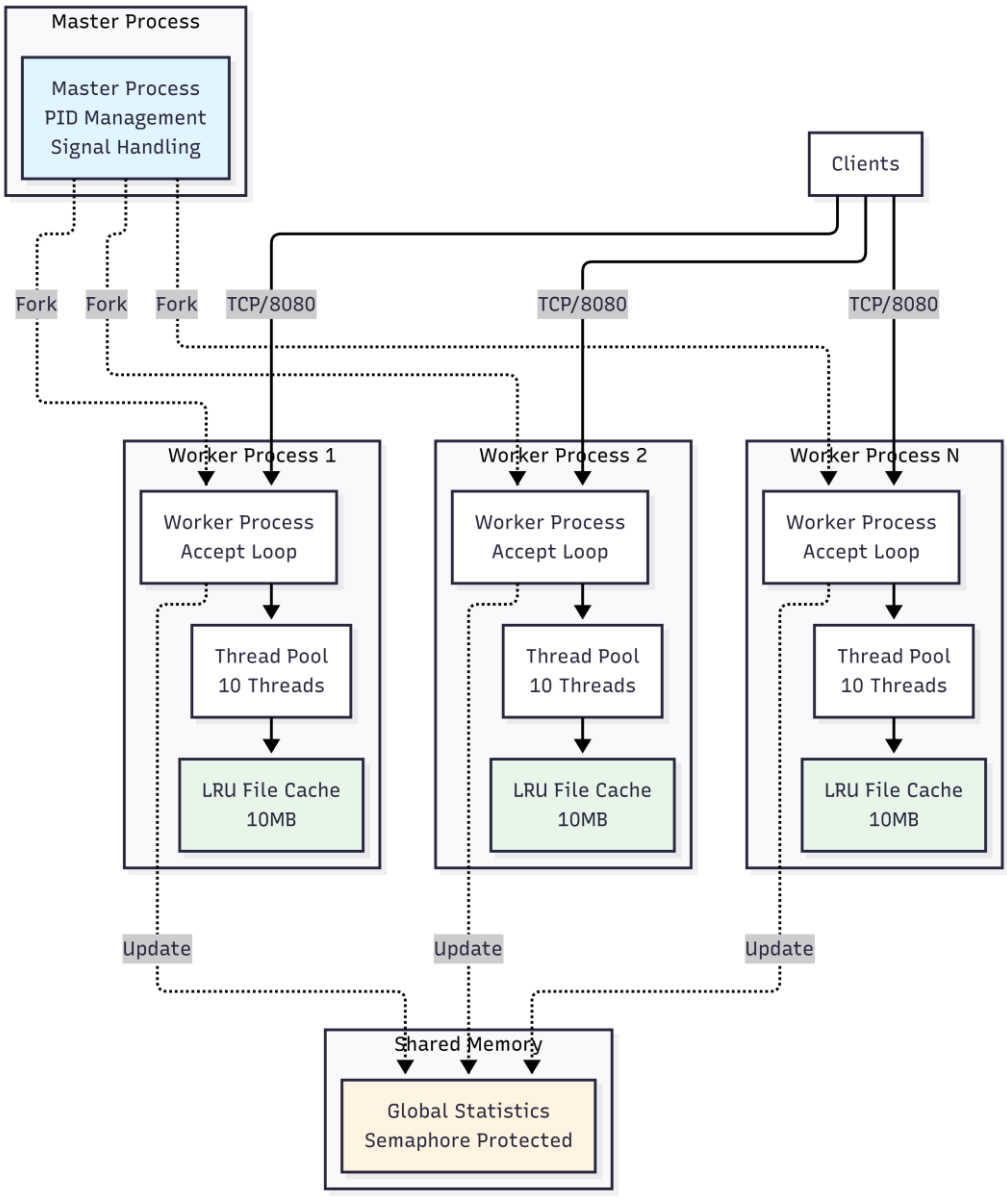
O sistema e composto por um processo master responsável pela inicialização e coordenação global de um conjunto fixo de workers. Cada worker aceita conexões TCP, distribui-as por um pool de threads através de uma fila producer-consumer e atualiza estatísticas globais mantidas em memoria partilhada, protegida por semáforos POSIX.

Arquitetura geral do sistema

A arquitetura segue o modelo master-worker com pools de threads locais. O processo master inicializa os recursos globais e cria os workers, que são responsáveis pelo tratamento das requisições

A utilização da opção SO_REUSEPORT permite que múltiplos workers executem *accept()* em paralelo.

Cada worker matem recursos locais (cache, fila e threads) enquanto as estatísticas são globais e partilhadas.



Arquitetura de Processos

Processo Master

O processo master tem como principais responsabilidades:

- Carregamento da configuração do servidor
- Inicialização de memora partilhada para estatísticas globais
- Criação e configuração do scoker TCP
- Criação dos Processos worker através do *fork()*
- Monotorização do ciclo de vida dos workers
- Gestão do “gracefull shutdown”

O master não processa requisições HTTP diretamente.

Processo Worker

Cada processo worker e responsável por:

- Aceitar conexões TCP
- Inicialização de uma cache LRU local
- Criação e gestão de um pool fixo de threads
- Por na fila as conexões
- Processar diretamente os endpoints prioritários
- Responder com código HTTP 503 quando fila se encontra saturada

Modelo de Concorrência

O modelo de concorrência combina múltiplos processos com múltiplas threads por processo. Esta abordagem isola falhas entre os vários workers e reduz o “racing” de recursos através de recursos locais.

A comunicação entre o loop de aceitação e as threads e realizada através de uma fila producer-consumer.

Design de Sincronização

Estatísticas em Memoria Partilhada

As estatísticas globais são armazenadas numa região de memoria partilhada alocada com o *mmap()*. O acesso e protegido por semáforo POSIX unico evitando *race-conditions*.

Fila de Conexões

A fila de conexões e implementada com um buffer circular limitado e protegida por três semáforos:

- Semáforo de slots vazios
- Semáforo de slots ocupados
- Semáforo mutex para exclusão mútua

Esta estrutura prevê o crescimento ilimitado do consumo de memorio sob carga extrema.

Cache LRU

O cache de ficheiros estáticos e mantido localmente em cada worker e protegido por um “reader-writer lock”. Assim e permitido múltiplas leituras concorrentes sendo a escrita necessária apenas em situações de *cache miss* ou *eviction*.

Funcionalidades Adicionais

Os enpoints de monitorização foram integrados diretamente no modelo de concorrência:

- Endpoints prioritários não utilizam a fila de conexões
- O acesso as estatísticas e feito diretamente da memora partilhada

Esta implementação garante que os mecanismos de monitorização permanecem funcionais mesmo em carga elevada.

Decisões de Design

As principais decisões sobre a arquitetura são:

- Arquitetura master-worker
- Cache LRU por worker em vez de cache global
- Utilização de memoria partilhada para estatísticas em vez de filas de mensagens
- Fila limitada para controlo explicito de carga
- Processamento prioritário de endpoints críticos

Conclusão

As decisões tomadas foram feitas priorizando a simplicidade, eficiência e segurança, fazendo com que o servidor HTTP seja capas de escalar sobre carga e falhar de forma controlada.

Nota

Todo o código fonte, scripts de teste e ficheiros de configuração encontram-se disponíveis no repositório entregue.

Os diagramas detalhados encontram-se disponíveis em https://github.com/Alxit0/so-websocket-ipc/tree/main