Trabalho Prático de Sistemas Operativos

Grupo 33
André Santos A106854
Pedro Ferreira A107292
Daniel Parente A107363

17 de maio de 2025

Resumo

Este relatório descreve o desenvolvimento de um sistema cliente–servidor em C para indexação e pesquisa de documentos. A solução recorre a uma arquitectura multiprocesso, comunicação através de FIFOs nomeados, um protocolo binário baseado em TLV (Type-Length-Value) e uma cache LRU persistente para acelerar pesquisas repetidas. São apresentadas as principais decisões de engenharia, a organização modular do código, os ganhos de desempenho obtidos com paralelização e os desafios ultrapassados ao longo do projecto.

Conteúdo

1 Introdução

O presente trabalho tem como objetivo a construção de um sistema que permita o armazenamento, indexação e pesquisa de documentos com suporte a múltiplas funcionalidades. Utilizando C como linguagem de programação, foram desenvolvidos dois executáveis principais: `dserver`, que atua como servidor, e `dclient`, que interage com o utilizador.

2 Estrutura Geral do Projeto

2.1 Componentes Principais

O projeto está dividido nos seguintes módulos:

- `src/dserver.c` – processo pai; recepção de pedidos, dispatcher, cache e storage.
- `src/dclient.c` – cliente CLI que constrói pedidos TLV e apresenta respostas.
- `src/common/transport.c` – camada de transporte baseada em FIFOs nomeados.
- `src/common/protocol.c` – definição dos cabeçalhos e codificação/decodificação de TLVs.
- `src/server/dispatcher.c` – valida pedidos e invoca os handlers adequados.
- `src/server/handlers/` – lógica das operações (add, check, list, search, delete, flush).
- `src/server/storage.c` – ficheiro binário de registos fixos para persistência.
- `src/server/cache/cache_lru.c` – implementação da cache LRU sobre GLib `GHashTable`.
- `src/server/doc/docutil.c` – utilitários para paths e contagem de keywords via `mmap`.

2.2 Compilação

A compilação é feita via Makefile, e os binários são gerados na pasta `bin/`.

3 Funcionamento do Servidor

O servidor é executado com dois argumentos:

- `document_folder`: diretório onde se encontram os documentos.
- `cache_size`: número máximo de blocos a manter em cache.

3.1 FIFO e Conectividade

A comunicação entre o cliente e servidor é feita via FIFOs nomeados:

- O servidor cria um FIFO público `/tmp/dserver.fifo`, usado por todos os clientes para enviar pedidos.
- Cada cliente cria um FIFO privado `/tmp/client<PID>.fifo` no qual receberá a resposta.
- Processos internos do servidor (workers de primeiro e segundo nível) comunicam com o processo pai através de `pipe`s anónimos e regiões de memória partilhada `mmap` quando aplicável.

4 Implementação da Cache

A cache encontra-se encapsulada em `src/server/cache/cache_lru.c` e possui as seguintes características:

- Estrutura híbrida `GHashTable+lista duplamente ligada` para obter *O(1)* em leitura/escrita e manuntenção da ordem LRU.
- Persistência opcional em disco (`tmp/cache_lru.bin`) para aquecimento rápido em arranques futuros.
- Capacidade configurável na linha de comandos; compilação alternativa `CACHE_NONE` desactiva a cache.
- API mínima: `cache_init()`, `cache_get()`, `cache_put()` e `cache_cleanup()`.

5 Dificuldades Encontradas

- **Sincronização entre processos**: coordenar `fork()` em dois níveis e evitar *zombies* exigiu uso adequado de `waitpid` com `WNOHANG`.
- **Consistência da cache**: garantir que apenas o processo pai escreve na cache, evitando necessidades de bloqueios.
- **Gestão de I/O em TLV**: validação exaustiva dos tipos/valores para prevenir corrupção de protocolo.
- **Persistência e recuperação**: lidar com ficheiros parcialmente escritos em caso de falhas inesperadas.

6 Conclusão

O sistema desenvolvido alcançou os objetivos propostos: é modular, robusto e eficiente. A utilização de cache e comunicação por FIFO entre processos mostra-se adequada para o domínio do projeto, e as funcionalidades foram organizadas de forma clara e extensível.