Universidade do Minho

Licenciatura em Ciências da Computação

Unidade Curricular de Sistemas Operativos

Ano Letivo de 2024/2025

Serviço de Indexação e Pesquisa de Documentos

Ivo Sousa a102935

Ricardo Cerqueira a102878

Índice

Índice de Figuras

1. Arquitetura do Sistema

O sistema desenvolvido é baseado numa arquitetura de dois processos independentes: o **cliente** (dclient) e o **servidor** (dserver). A comunicação entre ambos é realizada através de **FIFOs nomeados** (named pipes), que permitem a troca de mensagens entre processos distintos de forma assíncrona e segura.

O servidor, ao iniciar, cria um FIFO global com o nome fixo /tmp/dserver_fifo, onde permanece em escuta contínua de pedidos enviados pelos clientes. Cada cliente, ao efetuar uma operação, cria um FIFO temporário com o nome /tmp/client_fifo_<PID>, onde <PID> corresponde ao identificador do processo cliente. Este FIFO é utilizado exclusivamente para receber a resposta do servidor.

Este modelo permite o suporte a múltiplos clientes simultâneos, sem necessidade de ligações persistentes, mantendo a arquitetura simples e eficiente. A utilização de FIFOs

garante isolamento entre clientes, evitando colisões de mensagens e promovendo uma comunicação direta e unívoca com o processo servidor.

2. Funcionalidades Implementadas

O sistema implementa um conjunto de funcionalidades acessíveis através da aplicação cliente (dclient), que comunica com o servidor (dserver) utilizando mensagens formatadas via FIFO. As funcionalidades disponíveis são as seguintes:

- **-a** — *Adicionar documento*: Indexa um novo documento no sistema, associando-lhe um título, autores, ano e caminho relativo ao diretório de documentos. Esta operação atribui automaticamente um ID incremental ao novo documento e armazena os seus metadados.
- **-c** — *Consultar metadados*: Permite consultar os metadados (título, autores, ano e caminho) de um documento previamente indexado, através do seu ID.
- **-l** — *Contar linhas com palavra-chave*: Conta e devolve o número de linhas de um documento que contêm uma determinada palavra-chave, usando uma combinação de comandos grep e wc -l.
- **-s** — *Pesquisar documentos (modo sequencial)*: Procura todos os documentos que contêm uma determinada palavra-chave no seu conteúdo. A pesquisa é realizada sequencialmente, documento a documento.
- **-s <keyword> <n>** — *Pesquisar documentos (modo concorrente)*: Variante otimizada da operação anterior, permite a execução concorrente da pesquisa utilizando até n processos simultâneos, acelerando consideravelmente a operação em grandes volumes de dados.
- **-d** — *Remover documento*: Remove um documento da base de dados, eliminando também a sua entrada da cache se estiver presente. Esta operação liberta o ID do documento e impede futuras consultas.
- **-f** — *Encerrar servidor*: Solicita a terminação controlada do servidor, libertando recursos como FIFOs e encerrando a escuta de pedidos.

3. Gestão da Cache

Foi implementado um sistema de cache com política **LFU (Least Frequently Used)**, que prioriza a manutenção em memória dos documentos mais frequentemente acedidos ao

longo do tempo. Cada entrada da cache mantém um contador de acessos associado, o qual é incrementado sempre que o documento correspondente é consultado pelas operações -c, -l ou -s.

Para garantir que a frequência de acesso de um documento não se perde caso este seja removido da cache (por exemplo, devido a falta de espaço), foi criada uma estrutura auxiliar denominada *access_history[]*. Esta estrutura mantém um registo persistente da contagem de acessos de cada documento com base no seu ID. Assim, se um documento for removido e mais tarde readicionado à cache, a sua contagem de acessos é restaurada e incrementada corretamente, mantendo a integridade da política LFU.

A substituição na cache ocorre sempre que esta atinge o seu limite de capacidade. Nesse caso, o documento com menor número de acessos é removido, libertando espaço para a nova entrada. O sistema também evita duplicação de entradas na cache, garantindo consistência e eficiência.

Esta abordagem garante que a cache se adapta dinamicamente ao padrão de utilização real dos documentos, o que favorece o desempenho em cenários com acessos repetidos aos mesmos dados.

4. Concorrência

Na operação de pesquisa concorrente (-s <keyword> <n>), o servidor paraleliza a verificação do conteúdo dos documentos criando múltiplos processos filhos. Cada processo executa a busca por uma palavra-chave num subconjunto dos documentos indexados. O número máximo de processos em execução simultânea é limitado pelo valor n, fornecido como argumento pelo cliente, de forma a controlar o grau de concorrência e evitar sobrecarga do sistema.

Para implementar esta funcionalidade, foi utilizado o sistema de criação de processos com fork(). A comunicação entre os processos filhos e o processo pai é realizada através de pipe(), permitindo que cada filho envie o resultado da sua pesquisa (os IDs dos documentos que contêm a palavra-chave) de volta ao processo principal. Este, por sua vez, agrega os resultados parciais de todos os filhos, compila uma resposta final formatada e envia-a ao cliente através do FIFO correspondente.

Esta abordagem permite uma utilização mais eficiente dos recursos disponíveis, especialmente em sistemas com múltiplos núcleos, reduzindo significativamente o tempo de execução da operação de pesquisa em grandes volumes de dados.

5. Comunicação Cliente-Servidor

A comunicação entre o cliente (dclient) e o servidor (dserver) é realizada exclusivamente através de FIFOs (named pipes), isto garante uma troca de mensagens síncrona e orientada por comandos. O servidor cria um FIFO global (/tmp/dserver_fifo) no qual permanece a escutar pedidos provenientes dos clientes.

Cada mensagem enviada pelo cliente segue um formato padronizado: **COMANDO|arg1|arg2|...:PID**, onde COMANDO representa a operação a realizar (ex: ADD, CONSULT, DELETE, SEARCH, SEARCH_PAR, LINES e SHUTDOWN), os argumentos são específicos à operação, e PID é o identificador do processo cliente, usado para criar um FIFO de resposta único.

Com base no PID incluído na mensagem, o servidor responde através de um FIFO específico, com o nome /tmp/client_fifo_<PID>. Este mecanismo permite múltiplos clientes em simultâneo, garantindo que cada um recebe apenas a resposta à sua própria requisição. O cliente cria este FIFO antes de enviar o pedido e remove-o assim que a resposta é recebida, assegurando uma gestão limpa dos recursos.

6. Avaliação Experimental

Para a avaliação experimental, foi utilizado um script automático (testPerformance.sh) para testar o desempenho com diferentes tamanhos de cache e diferentes níveis de concorrência. Os tempos de execução foram medidos para os comandos -s, -l, -c e -d. Estes comandos foram aplicados a todos os documentos indexados.

Realizaram-se três testes onde se obteve os seguintes resultados. No primeiro teste:

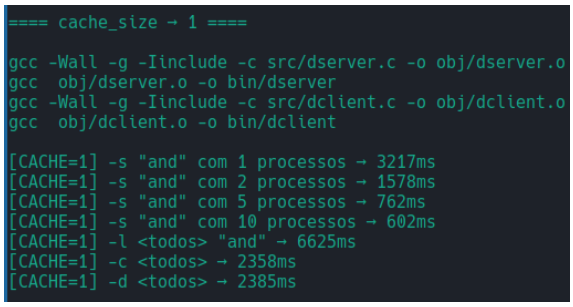

Figura 1 - cache_size 1 do teste 1

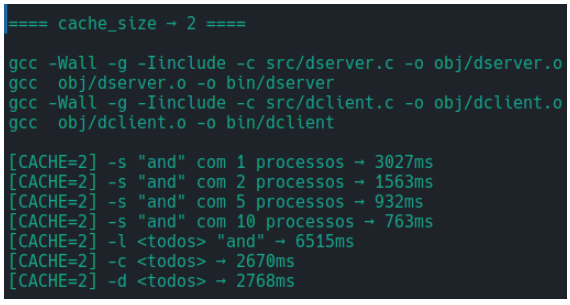

Figura 2 - cache_size 2 do teste 1

```
==== cache_size → 10 ====

gcc -Wall -g -Iinclude -c src/dserver.c -o obj/dserver.o
gcc  obj/dserver.o -o bin/dserver
gcc -Wall -g -Iinclude -c src/dclient.c -o obj/dclient.o
gcc  obj/dclient.o -o bin/dclient

[CACHE=10] -s "and" com 1 processos → 3019ms
[CACHE=10] -s "and" com 2 processos → 1558ms
[CACHE=10] -s "and" com 5 processos → 732ms
[CACHE=10] -s "and" com 10 processos → 595ms
[CACHE=10] -l <todos> "and" → 6438ms
[CACHE=10] -c <todos> → 2388ms
[CACHE=10] -d <todos> → 2505ms
```

Figura 4 - cache_size 10 do teste 1

```
==== cache_size → 50 ====

gcc -Wall -g -Iinclude -c src/dserver.c -o obj/dserver.o
gcc  obj/dserver.o -o bin/dserver
gcc -Wall -g -Iinclude -c src/dclient.c -o obj/dclient.o
gcc  obj/dclient.o -o bin/dclient

[CACHE=50] -s "and" com 1 processos → 2995ms
[CACHE=50] -s "and" com 2 processos → 1565ms
[CACHE=50] -s "and" com 5 processos → 733ms
[CACHE=50] -s "and" com 10 processos → 615ms
[CACHE=50] -l <todos> "and" → 6417ms
[CACHE=50] -c <todos> → 2703ms
[CACHE=50] -d <todos> → 2554ms
```

Figura 3 - cache_size 50 do teste 1

No segundo teste:

```
==== cache_size → 1 ====

gcc -Wall -g -Iinclude -c src/dserver.c -o obj/dserver.o
gcc  obj/dserver.o -o bin/dserver
gcc -Wall -g -Iinclude -c src/dclient.c -o obj/dclient.o
gcc  obj/dclient.o -o bin/dclient

[CACHE=1] -s "and" com 1 processos → 2860ms
[CACHE=1] -s "and" com 2 processos → 1557ms
[CACHE=1] -s "and" com 5 processos → 721ms
[CACHE=1] -s "and" com 10 processos → 555ms
[CACHE=1] -l <todos> "and" → 6301ms
[CACHE=1] -c <todos> → 2518ms
[CACHE=1] -d <todos> → 2577ms
```

Figura 7 - cache_size 1 do teste 2

```
==== cache_size → 2 ====

gcc -Wall -g -Iinclude -c src/dserver.c -o obj/dserver.o
gcc  obj/dserver.o -o bin/dserver
gcc -Wall -g -Iinclude -c src/dclient.c -o obj/dclient.o
gcc  obj/dclient.o -o bin/dclient

[CACHE=2] -s "and" com 1 processos → 2853ms
[CACHE=2] -s "and" com 2 processos → 1573ms
[CACHE=2] -s "and" com 5 processos → 732ms
[CACHE=2] -s "and" com 10 processos → 612ms
[CACHE=2] -l <todos> "and" → 6525ms
[CACHE=2] -c <todos> → 2409ms
[CACHE=2] -d <todos> → 2450ms
```

Figura 6 - cache_size 2 do teste 2

```
==== cache_size → 10 ====

gcc -Wall -g -Iinclude -c src/dserver.c -o obj/dserver.o
gcc  obj/dserver.o -o bin/dserver
gcc -Wall -g -Iinclude -c src/dclient.c -o obj/dclient.o
gcc  obj/dclient.o -o bin/dclient

[CACHE=10] -s "and" com 1 processos → 3022ms
[CACHE=10] -s "and" com 2 processos → 1564ms
[CACHE=10] -s "and" com 5 processos → 739ms
[CACHE=10] -s "and" com 10 processos → 579ms
[CACHE=10] -l <todos> "and" → 6536ms
[CACHE=10] -c <todos> → 2541ms
[CACHE=10] -d <todos> → 2581ms
```

Figura 8 - cache_size 10 do teste 2

```
==== cache_size → 50 ====

gcc -Wall -g -Iinclude -c src/dserver.c -o obj/dserver.o
gcc  obj/dserver.o -o bin/dserver
gcc -Wall -g -Iinclude -c src/dclient.c -o obj/dclient.o
gcc  obj/dclient.o -o bin/dclient

[CACHE=50] -s "and" com 1 processos → 2863ms
[CACHE=50] -s "and" com 2 processos → 1556ms
[CACHE=50] -s "and" com 5 processos → 727ms
[CACHE=50] -s "and" com 10 processos → 594ms
[CACHE=50] -l <todos> "and" → 6430ms
[CACHE=50] -c <todos> → 2442ms
[CACHE=50] -d <todos> → 2654ms
```

Figura 5 - cache_size 50 do teste 2

Por fim, no último teste:

```
==== cache_size → 1 ====

gcc -Wall -g -Iinclude -c src/dserver.c -o obj/dserver.o
gcc  obj/dserver.o -o bin/dserver
gcc -Wall -g -Iinclude -c src/dclient.c -o obj/dclient.o
gcc  obj/dclient.o -o bin/dclient

[CACHE=1] -s "and" com 1 processos → 2885ms
[CACHE=1] -s "and" com 2 processos → 1568ms
[CACHE=1] -s "and" com 5 processos → 733ms
[CACHE=1] -s "and" com 10 processos → 595ms
[CACHE=1] -l <todos> "and" → 6387ms
[CACHE=1] -c <todos> → 2613ms
[CACHE=1] -d <todos> → 2443ms
```

Figura 10 - cache_size 1 do teste 3

```
==== cache_size → 2 ====

gcc -Wall -g -Iinclude -c src/dserver.c -o obj/dserver.o
gcc  obj/dserver.o -o bin/dserver
gcc -Wall -g -Iinclude -c src/dclient.c -o obj/dclient.o
gcc  obj/dclient.o -o bin/dclient

[CACHE=2] -s "and" com 1 processos → 2840ms
[CACHE=2] -s "and" com 2 processos → 1546ms
[CACHE=2] -s "and" com 5 processos → 727ms
[CACHE=2] -s "and" com 10 processos → 632ms
[CACHE=2] -l <todos> "and" → 6453ms
[CACHE=2] -c <todos> → 2655ms
[CACHE=2] -d <todos> → 2446ms
```

Figura 9 - cache_size 2 do teste 3

```
==== cache_size → 10 ====

gcc -Wall -g -Iinclude -c src/dserver.c -o obj/dserver.o
gcc  obj/dserver.o -o bin/dserver
gcc -Wall -g -Iinclude -c src/dclient.c -o obj/dclient.o
gcc  obj/dclient.o -o bin/dclient

[CACHE=10] -s "and" com 1 processos → 2842ms
[CACHE=10] -s "and" com 2 processos → 1565ms
[CACHE=10] -s "and" com 5 processos → 724ms
[CACHE=10] -s "and" com 10 processos → 607ms
[CACHE=10] -l <todos> "and" → 6426ms
[CACHE=10] -c <todos> → 2819ms
[CACHE=10] -d <todos> → 2507ms
```

Figura 12 - cache_size 10 do teste 3

```
==== cache_size → 50 ====

gcc -Wall -g -Iinclude -c src/dserver.c -o obj/dserver.o
gcc  obj/dserver.o -o bin/dserver
gcc -Wall -g -Iinclude -c src/dclient.c -o obj/dclient.o
gcc  obj/dclient.o -o bin/dclient

[CACHE=50] -s "and" com 1 processos → 2866ms
[CACHE=50] -s "and" com 2 processos → 1563ms
[CACHE=50] -s "and" com 5 processos → 741ms
[CACHE=50] -s "and" com 10 processos → 605ms
[CACHE=50] -l <todos> "and" → 6494ms
[CACHE=50] -c <todos> → 2549ms
[CACHE=50] -d <todos> → 2444ms
```

Figura 11 - cache_size 50 do teste 3

Após observer os resultados obtidos é possivel concluir que:

A operação -s demonstrou uma melhoria clara com o aumento do número de processos, passando de cerca de 3000 ms com 1 processo para cerca de 600 ms com 10 processos. Isto mostra que a implementação de concorrência com fork() e pipe() foi eficaz.

A operação -l, que executa grep | wc -l para cada documento, mostrou pouca variação com o tamanho da cache, pois depende principalmente do conteúdo dos ficheiros, não beneficiando diretamente da cache de metadados.

A operação -c mostrou melhorias modestas com tamanhos de cache maiores, o que era esperado dado que os acessos eram uniformes e únicos por documento. Uma política LFU (Least Frequently Used) é mais eficaz quando há padrões de reutilização, o que não foi o caso nestes testes.

A operação -d, de remoção, manteve um tempo de execução estável em todos os testes, dado que a sua execução é simples e rápida, com ou sem cache. Estes resultados validam a correta implementação do sistema e confirmam que a cache e a concorrência estão a funcionar conforme esperado.