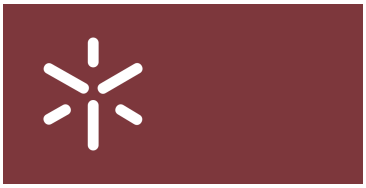

Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Unidade Curricular de Sistemas Operativos

Ano Letivo de 2024/2025
Grupo 86

Projeto SO

André Pinto A104267
Hélder Cruz a104174
Rafael Pereira a104095

Maio, 2025

SO

Data da Receção	
Responsável	
Avaliação	
Observações	

Projeto SO

André Pinto A104267
Hélder Cruz a104174
Rafael Pereira a104095

Maio, 2025

Índice

Lista de Figuras

1. Introdução

Este relatório tem como objetivo apresentar e discutir o processo de desenvolvimento da segunda fase do projeto realizado na unidade curricular de Sistemas Operativos.

1.1. Contextualização

O objetivo deste projeto é desenvolver um serviço que permita a indexação e pesquisa de documentos de texto armazenados localmente num computador. Este sistema é composto por dois programas principais: um servidor e um cliente.

O **servidor** é responsável por manter e gerir a meta-informação associada a cada documento, como por exemplo: identificador único, título, ano, autor e localização no sistema de ficheiros. Além disso, permite realizar interrogações tanto sobre a meta-informação como sobre o conteúdo textual dos documentos indexados.

Os utilizadores interagem com o serviço através de um programa **cliente**, que permite adicionar ou remover a indexação de documentos, bem como realizar pesquisas com base nos critérios suportados. Importa salientar que o cliente não é interativo – ou seja, cada execução corresponde à realização de uma única operação.

2. Resumo

2.1. Objetivos do Sistema

O principal objetivo do sistema é fornecer uma solução robusta para a organização e busca rápida de documentos em ambientes com grandes volumes de dados. Para isso, o sistema foi projetado para:

- Armazenar metadados de documentos (título, autor, ano) de forma eficiente.
- Permitir a rápida recuperação desses documentos com base em identificadores únicos.
- Minimizar o tempo de acesso aos documentos frequentemente consultados.
- Manter persistência dos dados para garantir continuidade após reinicializações.

2.2. Principais Funcionalidades

1. **Adição de Documentos (`-a`)**

- Permite adicionar um documento com os seguintes campos:
  - Título
  - Autores
  - Ano de publicação
  - Caminho para o ficheiro do documento
- Os metadados são extraídos automaticamente do conteúdo, guardados e associados a um identificador único.

2. **Consulta de Documentos (`-c`)**

- Consulta os metadados de um documento a partir do seu identificador (`id`).
- Os dados apresentados incluem título, autores, ano e caminho do ficheiro.
- Integra gestão de cache LRU, distinguindo entre `HIT` e `MISS`.

3. **Contagem de Linhas com Palavra-chave (`-l`)**

- Conta o número de linhas num documento que contêm uma palavra-chave.
- Implementado através de `fork` e `exec` com o comando `grep -c`.

4. **Pesquisa Concorrente (`-s`)**

- Pesquisa a palavra-chave em todos os documentos indexados usando múltiplos processos.
- Mostra o número de ocorrências por documento.
- Mede e apresenta o tempo de execução total da pesquisa.

5. **Remoção de Documento (`-d`)**

- Permite remover um documento do índice, atualizando os dados persistentes.

6. **Encerramento do Servidor (`-f`)**

- Encerra de forma segura o servidor, garantindo a escrita dos dados persistentes.
- Exporta estatísticas da cache e o estado atual da cache para ficheiro.

7. **Cache com Política LRU:** Para otimizar a velocidade das consultas, o sistema utiliza uma política de cache LRU (Least Recently Used), que armazena temporariamente os documentos mais recentemente acessados. Isso reduz significativamente o tempo de resposta para documentos populares.

2.3. Desempenho e Escalabilidade

O sistema foi projetado para ser escalável, suportando desde pequenos até grandes volumes de dados, com mecanismos eficientes de processamento paralelo e gerenciamento de cache.

3. Desenvolvimento

3.1. Estrutura do Sistema

O sistema desenvolvido segue uma arquitetura cliente-servidor local baseada em comunicação por FIFO.

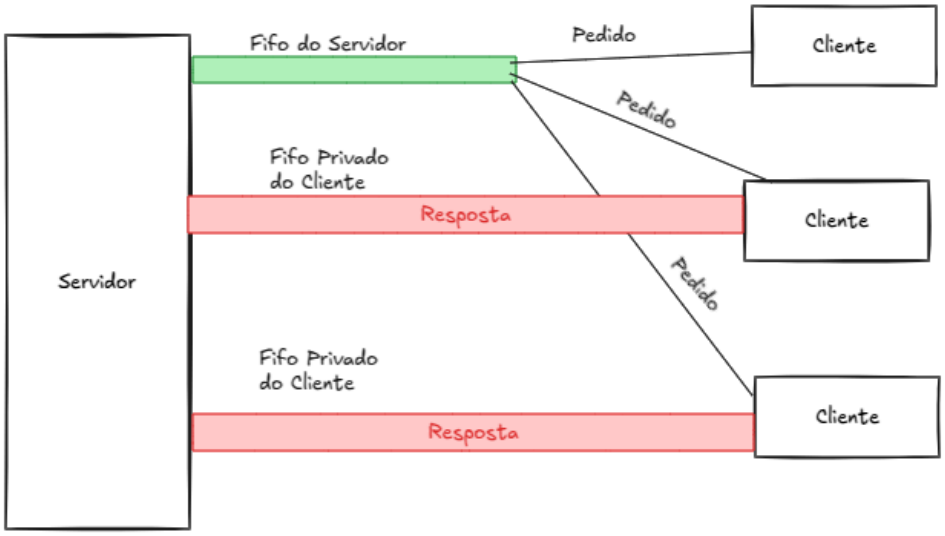

Figura 1: Comunicação Servidor Cliente

3.1.1. Componentes Principais

Cliente

Envia pedidos ao servidor através de um FIFO partilhado e recebe a resposta num FIFO exclusivo, criado dinamicamente com base no PID do processo.

Servidor

Recebe pedidos dos clientes e executa operações sobre a base de dados de documentos.

3.1.2. Fluxo de Comunicação

Pedido do Cliente:

O cliente constrói uma mensagem, contendo o comando, os argumentos e o seu PID.

Envia essa estrutura ao servidor através do FIFO associado ao servidor.

Resposta do Servidor:

O servidor processa o pedido com base no tipo de operação.

Cria uma resposta.

Envia a resposta para o FIFO privado do cliente, que este aguarda para ler.

Final da Comunicação:

O cliente remove o seu FIFO temporário após a leitura.

O servidor mantém-se ativo até receber o comando de encerramento (CMD_EXIT).

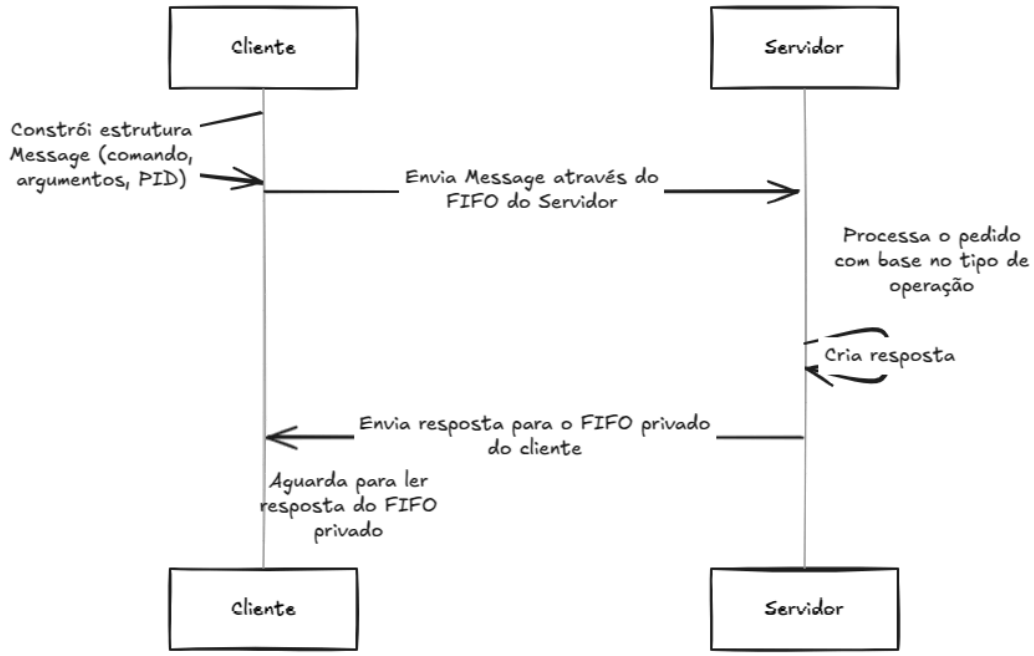

Figura 2: Funcionamento do sistema

3.1.3. Gestão de Dados

Os documentos não são guardados no sistema, apenas os seus metadados e localizações no disco.

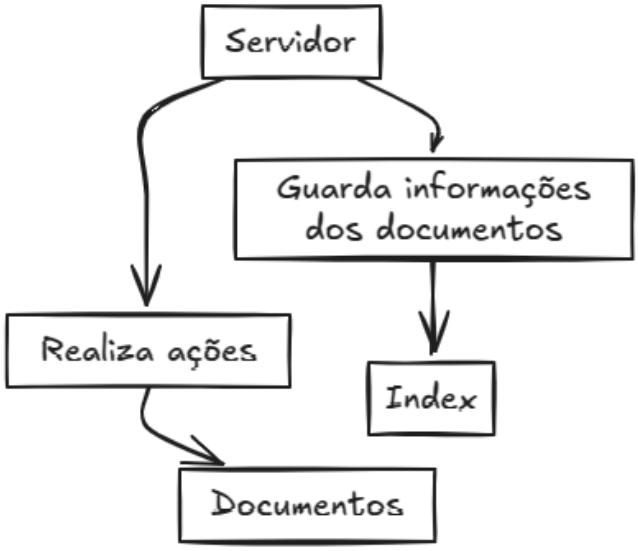

Figura 3: Gestão de Dados

A cache LRU guarda os documentos mais recentemente acedidos para acelerar futuras consultas.

A persistência é garantida com funções como index_save() e index_load(), que armazenam a base de dados num ficheiro (docdb.txt).

3.2. Servidor

O servidor é a peça central do sistema de indexação e pesquisa de documentos. Este processo é persistente e aguarda pedidos dos clientes, respondendo conforme o tipo de operação solicitada.

3.2.1. Responsabilidades do Servidor

1. Gestão da Base de Dados: O servidor é responsável por manter a base de dados de documentos indexados.
2. Execução de Operações: Realiza operações solicitadas pelos clientes, como adicionar, remover e consultar documentos.
3. Persistência de Dados: Garante a persistência das informações, carregando e salvando dados em ficheiros.
4. Gestão de Cache LRU: Utiliza uma cache LRU para otimizar o acesso a documentos frequentemente consultados.

3.2.2. Estrutura Geral de Funcionamento

- **Inicialização**

1. O servidor cria o FIFO público (geral) para a comunicação com os clientes.
2. A base de dados de documentos previamente salva é carregada.
3. O servidor entra num ciclo infinito aguardando pedidos dos clientes.

- **Ciclo de Atendimento a Pedidos**

1. O servidor lê a estrutura Message enviada por um cliente.
2. Analisa o tipo de comando e executa a operação correspondente.

3 .Constrói a resposta com o resultado da operação.

4. A resposta é então enviada de volta para o FIFO do cliente.

- ***Encerramento e Persistência***

Antes de encerrar:

1. O servidor salva a base de dados de documentos, garantindo a persistência dos dados.
2. O FIFO do servidor é removido.

3.2.3. Comandos SuportadosO servidor reconhece vários comandos,incluindo:

Adicionar (CMD_ADD): Indexa um novo documento com base no caminho fornecido.

Remover (CMD_REMOVE): Elimina um documento do índice, dado o seu ID.

Consultar (CMD_QUERY): Retorna os metadados de um documento.

Contagem de Palavras (CMD_WORDCOUNT): Conta o número de linhas onde uma palavra aparece num documento.

Pesquisa Global (CMD_SEARCH): Procura por uma palavra em todos os documentos indexados.

Exportar Cache (CMD_EXPORT): Gera um snapshot da cache para um ficheiro.

Encerrar Servidor (CMD_EXIT): Termina a execução do servidor de forma segura.

3.3. Cliente

O programa cliente,tem o papel de servir como intermediário entre o utilizador e o programa servidor. Este cliente não é interativo, ou seja, cada invocação do programa efetua apenas uma operação, definida pelos argumentos passados via linha de comandos.

3.3.1. Objetivo Principal

Permitir que o utilizador execute remotamente operações como adicionar, remover ou consultar documentos através da comunicação com o servidor.

3.3.2. Funcionamento Geral

O cliente analisa os argumentos passados na linha de comandos para determinar a operação a realizar. Tipicamente, o primeiro argumento define o comando (add, remove, search, etc.) e os seguintes contêm os dados necessários.

3.3.3. Formatação do Pedido:

O cliente constrói uma mensagem com o comando e os dados associados.
Esta mensagem é então enviada ao servidor, que a irá interpretar.

3.3.4. Receção da Resposta:

Após o envio do pedido, o cliente aguarda uma resposta do servidor (quando aplicável), por exemplo, o resultado de uma pesquisa ou o ID de um documento adicionado. Esta resposta é apresentada ao utilizador no stdout.

3.3.5. Operações Suportadas

- **-a**: Adicionar um novo documento.
- **-c**: Consultar um documento pelo ID.
- **-d**: Remover um documento pelo ID.
- **-l**: Contar o número de linhas contendo uma palavra-chave em um documento específico.
- **-s**: Procurar uma palavra-chave em todos os documentos.
- **-f**: Encerrar o servidor.

3.3.6. Robustez e Validação

O cliente inclui verificação básica dos argumentos, emitindo mensagens de erro quando o número ou tipo de argumentos não é o esperado. Isto ajuda a garantir que apenas pedidos válidos são enviados ao servidor.

3.4. Avaliação e Testes de Eficiência

3.4.1. Teste de paralelismo

Este teste foi desenvolvido para avaliar o desempenho do programa cliente, utilizando a opção ***-s***, sob diferentes níveis de paralelismo. O foco está na medição do tempo necessário para processar uma palavra-chave específica.

Funcionamento

- O script executa o comando dclient -s múltiplas vezes, variando a quantidade de processos paralelos entre os seguintes valores: 1, 2, 5, 10, 30, 50, 100, 200, 500, 1000, 1500 e 2000.

Medição de Tempo:

Para cada configuração de paralelismo, o script segue os seguintes passos:

Inicia um cronometro (em milissegundos).

Executa o comando dclient -s com o número atual de processos.

Encerra o cronometro e regista o tempo total de execução.

3.4.1.1. Resultados Obtidos

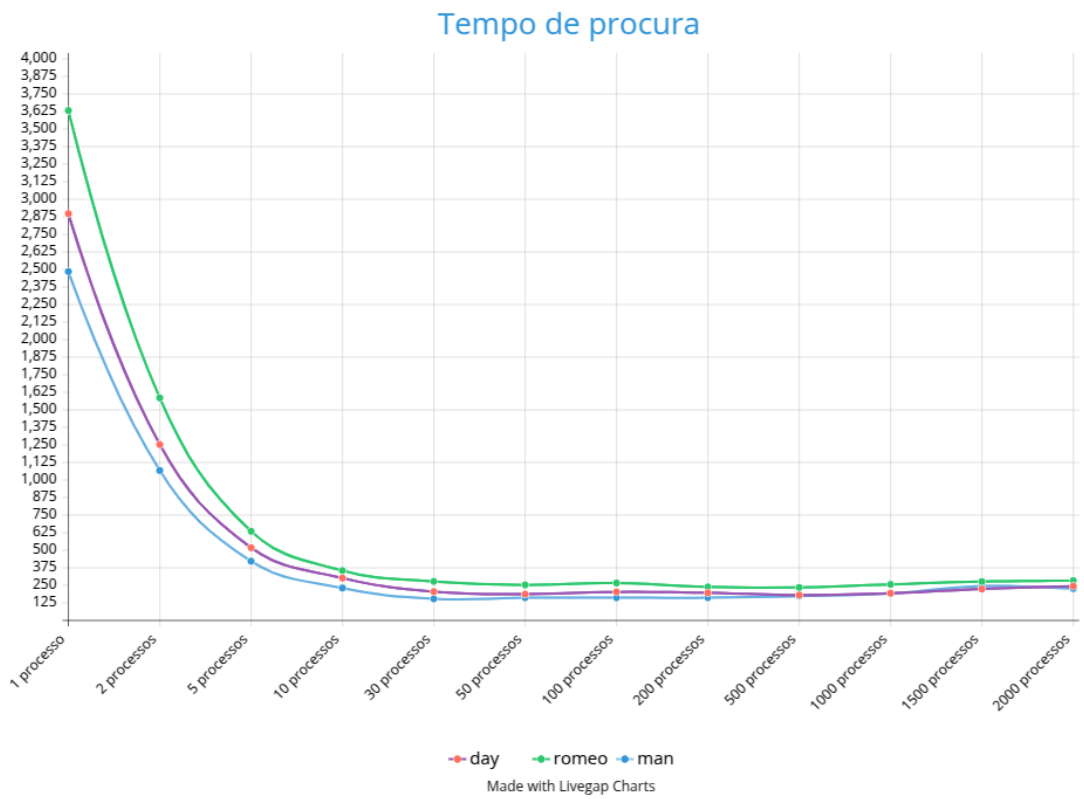

Figura 4: Funcionamento do sistema

Durante os testes com o comando dclient -s, foi possível observar que o tempo de execução diminui conforme o número de processos paralelos aumenta. Esse ganho de desempenho é evidente até cerca de 50 processos, onde o tempo total de busca é significativamente reduzido.

No entanto, a partir desse ponto, o tempo de execução não melhora significativamente. Mesmo com o uso de mais processos (100, 200, 500 ou mais), não há uma melhora percetível no desempenho. Vale ainda salientar que a partir de um certo ponto o tempo de execução começa a aumentar ligeiramente.

Este comportamento indica que o sistema atinge um limite ideal de paralelismo por volta dos 50 processos. Este comportamento pode ocorrer por:

- Limitações de Hardware
- Custo de gerir processos: (Criar, manter e alternar entre muitos processos tem um custo. O sistema operacional gasta mais tempo a coordenar os processos do que realmente a executar o trabalho)

3.4.2. Teste de tamanho da cache

Para avaliar o impacto do tamanho da cache no desempenho do sistema, foi criado um script de teste que executa o servidor dserver com diferentes configurações de cache e mede o tempo total necessário para processar uma sequência de pedidos feitos pelo cliente dclient. O objetivo deste teste é observar como a variação no número de entradas permitidas na cache afeta a eficiência do sistema, especialmente no contexto da política de substituição LRU (Least Recently Used).

O script percorre uma lista de tamanhos de cache e, para cada valor, inicia o servidor com a cache configurada para essa capacidade.

Em seguida, o cliente realiza uma série de requisições a documentos, simulando acessos normais ao sistema, com o intuito de forçar o funcionamento da política de substituição LRU e ativar a reutilização de itens armazenados na cache.

Ao final da sequência de requisições, o tempo total decorrido é calculado. O servidor é então encerrado, e o processo é repetido para o próximo tamanho de cache.

3.4.2.1. Resultados Obtidos

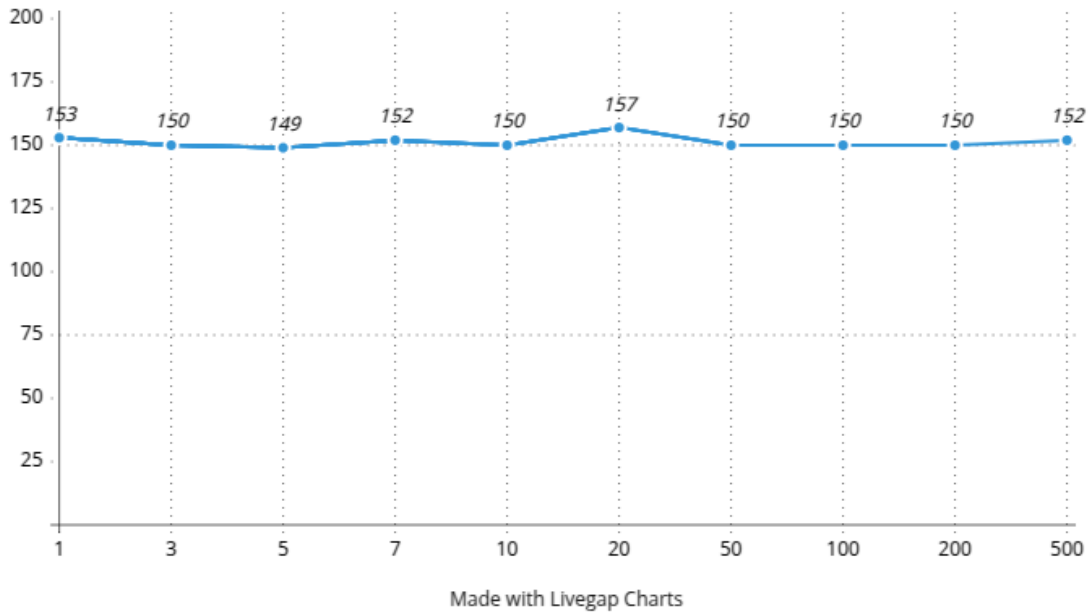

Figura 5: Resultado do teste de tamanho de cache

Como é possível observar, o aumento do tamanho da cache não resultou numa redução significativa do tempo médio de consultas. Este comportamento inesperado pode indicar a ocorrência de um dos seguintes problemas:

1. **Erro na implementação do programa**: A lógica de gestão da cache pode não estar a ser corretamente aplicada, o que impediria a utilização eficiente da memória adicional disponibilizada
2. **Erro na configuração ou conceção dos testes**: Os testes podem não ter sido desenhados de forma a mostrar o impacto que o aumento do tamanho da cache tem no desempenho do programa . Assim, é possível que a cache esteja corretamente implementada, mas não esteja a ser efetivamente utilizada durante os testes.

4. Dificuldades e Aspetos a Melhorar

Durante a realização do projeto, surgiram várias dificuldades,tais como:

- **Implementação correta da criação de processos com fork() e sincronização com wait()**: qualquer erro na lógica de criação ou espera de processos resultava facilmente em comportamentos incorretos, processos órfãos ou bloqueios.
- ***Comunicação entre processos com pipes***: garantir que a troca de dados entre processos ocorresse de forma correta e eficiente foi desafiador, especialmente ao lidar com buffers e o risco de bloqueio ou perda de dados.
- ***Dificuldades na depuração***: por envolver múltiplos processos e concorrência, o sistema era difícil de depurar.
- ***Uso adequado das chamadas de sistema***: houve dificuldade em compreender bem o funcionamento de chamadas de sistema.
- ***Compreensão clara da diferença entre modo utilizador e modo kernel***: entender quando o sistema estava a operar em cada modo foi algo complexo.
- ***Gestão de memória***: lidar com alocação e libertação de memória entre processos, buffers de comunicação e estruturas internas exigiu cuidado para evitar vazamentos ou acessos inválidos.
- ***Contagem precisa de hits e misses da cache e exportação de snapshots***: Contar corretamente os hits e misses da cache, e exportar snapshots úteis, exigiu gestão cuidadosa dos acessos e consistência nos logs.
- ***Execução paralela da pesquisa (-s) com múltiplos processos***: A execução de múltiplos processos para a pesquisa (-s) exigiu uma boa coordenação com fork, pipe, wait, e controlo de buffers. Erros mínimos resultavam em comportamentos incorretos ou bloqueios indesejados.
- ***Garantir a robustez do código***: foi desafiador manter o código estável e confiável. A adoção de boas práticas desde o início foi essencial para reduzir problemas e facilitar a manutenção.

5. Conclusão

Em resumo, o projeto foi uma experiência bastante enriquecedora. Permitiu-nos aprofundar o conhecimento sobre sistemas operativos. Apesar das dificuldades durante a execução do trabalho, acreditamos que o projeto final é bastante satisfatório e o conhecimento obtido poderá ser valioso para desafios académicos e profissionais futuros.