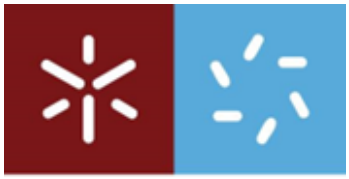

Universidade do Minho
Escola de Ciências

# Sistemas Operativos (2º ano de Ciências da Computação)

# **Trabalho Prático – Serviço de Indexação e Pesquisa de Documentos**

Grupo 16


**Feito por:**

- Alexandra Isabel de Sousa Calafate (A100060)
- André Filipe Barbosa de Sousa (A87999)
- João Nuno Rodrigues Fernandes (A87971)

# Table of Contents

# 1. Introdução

Este relatório descreve o desenvolvimento de um serviço de indexação e pesquisa de documentos, realizado no âmbito da unidade curricular de Sistemas Operativos. O sistema implementa uma arquitetura cliente-servidor que permite a gestão de meta-informação de documentos e a pesquisa sobre o seu conteúdo.

Os principais objetivos incluem a aplicação de conceitos de gestão de processos, comunicação entre processos (IPC) utilizando pipes nomeados, persistência de dados, otimização de desempenho através de caching e pesquisa concorrente.

A seguir, são expostas as decisões tomadas pelo grupo, bem como o porquê das mesmas e as suas vantagens e eventuais desvantagens. Também é explicada a implementação de forma breve, havendo um melhor aprofundamento apenas nas partes em que o grupo considera ser mais relevantes para a compreensão do funcionamento do software desenvolvido.

# 2. Arquitetura do Sistema

- Modelo Cliente-Servidor:
    - O sistema adota um modelo cliente-servidor, onde um processo servidor (dserver) centraliza a gestão dos dados e o processamento dos pedidos, e múltiplos processos cliente (dclient) interagem com o servidor para realizar operações.
- Comunicação Interprocessos (IPC):
    - A comunicação entre o cliente e o servidor é realizada exclusivamente através de pipes nomeados (FIFOs). Esta escolha justifica-se pela sua capacidade de permitir a comunicação entre processos não relacionados (não descendentes diretos), sendo um mecanismo standard em sistemas POSIX como o Linux.
    - Pipe Principal do Servidor: O servidor cria um FIFO conhecido, definido por SERVER_PIPE (e.g., /tmp/server_pipe_so), no qual aguarda por pedidos de todos os clientes.
    - Pipes de Resposta do Cliente: Cada cliente, antes de enviar um pedido, cria o seu próprio FIFO para receber a resposta do servidor. O nome deste FIFO é único e construído utilizando o PID (Process ID) do cliente, conforme o formato CLIENT_PIPE_FORMAT (e.g., /tmp/client_pipe_so_<PID>). O PID do cliente é incluído no pedido enviado ao servidor.

- Estruturas de Dados para Comunicação:
    - Os pedidos do cliente para o servidor são encapsulados na estrutura Request.

```c
typedef struct {
    int operation;                  // Código da operação a ser realizada (ver defines ADD_DOC, QUERY_DOC, etc.).
    Document doc;                   // Estrutura `Document` contendo os dados do documento.
                                    // Usada nas operações ADD_DOC (para enviar novos dados),
                                    // QUERY_DOC (para enviar o ID na `doc.id`),
                                    // DELETE_DOC (para enviar o ID na `doc.id`),
                                    // COUNT_LINES (para enviar o ID na `doc.id`).      You, 17 hours ago • Update [15/05/
    char keyword[MAX_KEYWORD_SIZE]; // Palavra-chave para operações de pesquisa de conteúdo.
                                    // Usada em COUNT_LINES e SEARCH_DOCS.
    int client_pid;                 // PID (Process ID) do processo cliente.
                                    // Essencial para o servidor saber para qual pipe de cliente deve enviar a resposta.
    int nr_processes;               // Número de processos a serem usados na pesquisa concorrente (SEARCH_DOCS).
                                    // Se não especificado ou ⩽ 1, a pesquisa é sequencial.
} Request;
```

    - As respostas do servidor para o cliente são encapsuladas na estrutura Response.

```c
typedef struct {
    int status;                     // Código de estado da operação:
                                    // 0 indica sucesso.
                                    // Um valor negativo indica um erro específico (ex: -1 para "não encontrado").
    Document doc;                   // Estrutura `Document` contendo os dados do documento retornado.
                                    // Preenchida na resposta a uma operação QUERY_DOC bem-sucedida.
                                    // Na resposta a ADD_DOC, `doc.id` contém o ID do novo documento.
    int count;                      // Resultado da contagem de linhas.
                                    // Preenchido na resposta a uma operação COUNT_LINES bem-sucedida.
    int ids[MAX_RESULT_IDS];        // Array de IDs dos documentos encontrados numa pesquisa.
                                    // Preenchido na resposta a uma operação SEARCH_DOCS.
    int num_ids;                    // Número de IDs válidos presentes no array `ids`.
                                    // Usado em conjunto com `ids` na resposta a SEARCH_DOCS.
} Response;
```

- Fluxo de Comunicação Típico:

1. O cliente cria o seu FIFO de resposta.
2. O cliente preenche uma estrutura Request e envia-a para o SERVER_PIPE.
3. O servidor lê o Request do SERVER_PIPE.
4. O servidor processa o pedido.
5. O servidor preenche uma estrutura Response e envia-a para o FIFO de resposta específico do cliente (identificado pelo client_pid no Request).
6. O cliente lê a Response do seu FIFO.
7. O cliente apresenta o resultado ao utilizador e remove o seu FIFO de resposta.

## 2.1 Servidor (dserver)

- Responsabilidades:
  - dserver é responsável por indexar meta-informação de documentos, processar consultas e pedidos de remoção, realizar pesquisas no conteúdo dos documentos, gerir a persistência dos dados e otimizar o acesso através de uma cache.

- Gestor de Pedidos:
  - servidor opera num ciclo principal, lendo sequencialmente os pedidos que chegam ao SERVER_PIPE.
    1. Processamento de Pedidos e Concorrência: Embora os pedidos sejam retirados do FIFO principal de forma sequencial, o servidor foi desenhado para minimizar o bloqueio de clientes. Para operações potencialmente longas, como a pesquisa por palavra-chave (-s), o servidor pode delegar o trabalho a múltiplos processos filhos, permitindo o processamento concorrente dessa pesquisa específica. No entanto, o ciclo principal do servidor que aceita novos pedidos permanece single-threaded, o que significa que um pedido muito demorado (mesmo que internamente paralelo) poderá atrasar o início do processamento do pedido seguinte.

- Arranque do Servidor:
  - Ao ser iniciado (./dserver document_folder cache_size ), o servidor:
    1. Valida os argumentos da linha de comandos (pasta de documentos e tamanho da cache ).
    2. Carrega a meta-informação persistida em disco.
    3. Inicializa a estrutura da cache com o tamanho especificado.
    4. Cria o SERVER_PIPE se este não existir.
    5. Regista handlers para sinais como SIGINT e SIGTERM para garantir um encerramento controlado (ver handle_signals no código).

- Encerramento do Servidor (dclient -f):
  - Quando o cliente envia o comando de encerramento (-f):
    1. O servidor recebe a operação SHUTDOWN.
    2. Realiza a persistência final dos dados da cache para o disco, se existirem modificações.
    3. Liberta os recursos alocados (e.g., memória da cache).
    4. Remove o SERVER_PIPE.
    5. Termina a sua execução.

## 2.2 Cliente (dclient)

- Funcionamento:
  - O dclient é uma aplicação de linha de comandos que executa uma única operação por invocação.

- Processamento de Argumentos:
  - Interpreta os argumentos fornecidos para construir a estrutura Request apropriada para a operação desejada (e.g., -a, -c, -d, -l, -s, -f).

- Comunicação:
  - Conforme descrito no fluxo de comunicação, cria o seu FIFO, envia o pedido, aguarda a resposta e processa-a.

- Saída e Limpeza:
  - Apresenta a informação relevante ao utilizador através do stdout e remove o seu FIFO privado após a conclusão da operação.

# 3. Implementação das Funcionalidades

## 3.1. Meta-informação de Documentos:

- A meta-informação de cada documento é armazenada numa estrutura Document.

```c
typedef struct {
    int id;                         // Identificador numérico único atribuído pelo servidor a cada documento.
    char title[MAX_TITLE_SIZE];     // Título do documento.
    char authors[MAX_AUTHORS_SIZE]; // Nome(s) do(s) autor(es) do documento.
    char year[MAX_YEAR_SIZE];       // Ano de publicação do documento (como string).
    char path[MAX_PATH_SIZE];       // Caminho relativo para o ficheiro físico do documento, a partir da pasta base configurada no servidor.
} Document;
```

## 3.2. Indexação de Documentos (dclient -a "título" "autores" "ano" "caminho")

- O cliente envia os metadados para o servidor.
- O servidor valida o tamanho total dos argumentos (não excede 512 bytes) e os tamanhos individuais (e.g., título e autores até 200 bytes cada, caminho até 64 bytes, ano até 4 bytes).

- O servidor verifica a acessibilidade do ficheiro no caminho especificado (relativo à document_folder). O ficheiro deve existir e ser legível.
- Geração do Identificador Único: É atribuído um identificador numérico único (id) a cada novo documento. Este id é gerido pelo servidor através de um contador global (next_id em dserver.c) que é incrementado a cada nova adição e persistido juntamente com os dados.
- A meta-informação é adicionada à cache (e posteriormente ao disco).
- O servidor retorna o id atribuído ao cliente, que o exibe.

## 3.3. Consulta de Meta-informação (dclient -c "ID")

- O cliente envia o ID do documento a consultar.
- O servidor procura o documento (primeiro na cache, depois no disco, se necessário).
- Se encontrado, a meta-informação (título, autores, ano, caminho) é devolvida ao cliente e exibida. Caso contrário, é comunicada uma mensagem de erro.

## 3.4. Remoção de Meta-informação (dclient -d "ID")

- O cliente envia o ID do documento a remover.
- O servidor remove a entrada de meta-informação da cache e do ficheiro de persistência.
- Importante: O ficheiro do documento em si não é apagado do sistema de ficheiros, apenas a sua indexação.
- O cliente informa o utilizador sobre o sucesso da operação.

## 3.5. Pesquisa de Conteúdo

### 3.5.1. Contagem de Linhas (dclient -l "ID" "palavra-chave")

- O servidor localiza o documento pelo ID.
- Para contar as linhas que contêm a palavra-chave, o servidor utiliza um mecanismo de pipeline entre os comandos grep e wc -l. Isto é implementado da seguinte forma (conforme a função count_lines_with_keyword em dserver.c):
    1. São criados dois pipes anónimos.
    2. O servidor cria um primeiro processo filho (fork).
        - O stdout deste filho é redirecionado (dup2) para a extremidade de escrita do primeiro pipe.
        - Este filho executa (execlp) grep palavra-chave /caminho/para/ficheiro_documento.
    3. O servidor cria um segundo processo filho.
        - O stdin deste segundo filho é redirecionado para a extremidade de leitura do primeiro pipe.
        - O stdout deste segundo filho é redirecionado para a extremidade de escrita do segundo pipe.
        - Este filho executa wc -l.
    4. O processo servidor (pai) lê o resultado (a contagem de linhas) da extremidade de leitura do segundo pipe.
    5. O servidor aguarda (waitpid) a terminação de ambos os filhos.

### 3.5.2. Pesquisa de Documentos por Palavra-Chave (Versão Sequencial - dclient -s "palavra-chave")

- O servidor itera por todos os documentos indexados (na cache e/ou disco).
- Para cada documento, verifica se contém a palavra-chave. Isto pode ser feito de forma similar à contagem de linhas, mas usando grep -q palavra-chave /caminho/para/ficheiro, que retorna um status de saída 0 se encontrar, sem produzir output. A função search_documents_with_keyword_serial em dserver.c implementa esta lógica.
- Os IDs dos documentos que contêm a palavra-chave são acumulados e enviados ao cliente numa lista.

# 4. Otimizações

## 4.1. Pesquisa Concorrente (dclient -s "palavra-chave" "nr_processos")

- Para acelerar a operação de pesquisa de documentos por palavra-chave, foi implementada uma versão concorrente que utiliza múltiplos processos.
- Implementação:
    1. O servidor obtém a lista de todos os documentos a serem pesquisados (combinando os da cache e os do disco, evitando duplicados).
    2. Se o número de processos (nr_processos) solicitado for maior que 1 e o número total de tarefas de pesquisa for significativo, a pesquisa paralela é ativada. Caso contrário (ou se nr_processos <= 1), recorre-se à pesquisa sequencial descrita em 3.5.2. (Ver lógica em search_documents_with_keyword_parallel em dserver.c).
    3. O conjunto total de documentos a pesquisar é dividido o mais equitativamente possível entre o nr_processos especificado.
    4. O servidor cria nr_processos processos filhos utilizando fork.
    5. Cada processo filho é responsável por pesquisar a palavra-chave no seu subconjunto de documentos. A pesquisa em cada ficheiro dentro do filho pode usar a mesma técnica de count_lines_with_keyword (para verificar se > 0) ou grep -q.
    6. Comunicação de Resultados Parciais: Cada processo filho escreve os IDs dos documentos que encontrou num ficheiro temporário único (e.g., /tmp/search_results_child_<PID_PAI>_<INDICE_FILHO>.tmp). Esta abordagem foi escolhida pela sua simplicidade na agregação e por evitar complexidades com múltiplos pipes para o pai.
    7. O processo pai (servidor) aguarda (waitpid) a terminação de todos os processos filhos.
    8. Após a terminação dos filhos, o pai lê os resultados de cada ficheiro temporário, agrega-os numa lista final de IDs (garantindo que não excede MAX_RESULT_IDS) e remove os ficheiros temporários.
    9. A lista final de IDs é enviada ao cliente.
- Justificação: A criação de processos distribuídos permite que a pesquisa em múltiplos ficheiros ocorra em paralelo, aproveitando sistemas multi-core e reduzindo o tempo total da operação para grandes volumes de dados. O uso de ficheiros temporários simplifica a agregação de resultados de múltiplos filhos assíncronos.

## 4.2. Persistência de Dados

- Para garantir a durabilidade da meta-informação dos documentos, mesmo após o encerramento do servidor, os dados são persistidos em disco.
- Implementação:
    - Ficheiro de Persistência: É utilizado um ficheiro binário, nomeadamente "database.bin" (conforme dserver.c).
    - Formato dos Dados:
        1. O next_id (próximo ID a ser atribuído) é guardado no início do ficheiro.
        2. Segue-se o número total de documentos (num_docs) atualmente guardados.
        3. Finalmente, as estruturas Document de cada documento são escritas sequencialmente.
    - Operações de Escrita (Save): Os dados são guardados no disco (save_documents em dserver.c):
        - Principalmente quando o servidor recebe o comando de encerramento (dclient -f) e a flag cache.modified indica que houve alterações.
        - Quando um documento é removido (remove_document), o ficheiro é reescrito para refletir a remoção.
    - Operações de Leitura (Load): Ao arrancar, o servidor (load_documents em dserver.c):
        1. Abre "database.bin".
        2. Lê o next_id e o total_docs_on_disk.
        3. Lê cada estrutura Document e carrega-a para a cache em memória, até ao limite da cache.

- Justificação: A persistência em ficheiro binário é eficiente em termos de espaço e velocidade de leitura/escrita comparativamente a formatos de texto para estruturas fixas.

## 4.3. Cache de Meta-informação

- Para acelerar o acesso à meta-informação frequentemente requisitada, o servidor mantém uma cache em memória.
- Implementação:
    - Tamanho da Cache: O tamanho da cache (N) é definido como um argumento na linha de comandos quando se inicia o dserver (./dserver document_folder cache_size). Este valor corresponde ao número máximo de entradas de meta-informação (Document) que a cache pode conter.
    - Estrutura da Cache: A cache é implementada como um array de ponteiros para estruturas Document (cache.docs em dserver.c).
    - Política de Cache:
        - Política de Adição: Novos documentos indexados são sempre adicionados à cache.
        - Política de Substituição: Quando um novo documento é adicionado e a cache está cheia (cache.num_docs >= cache.max_size), é aplicada uma política FCFS (First-Come, First-Served), também conhecida como FIFO (First-In, First-Out). O documento que está há mais tempo na cache (o que está no índice 0 do array, assumindo que novos elementos são adicionados no fim e os antigos "deslizam" ou são explicitamente geridos dessa forma) é removido para dar espaço ao novo. Esta política foi escolhida pela sua simplicidade de implementação. (Referenciar a função add_document em dserver.c que implementa esta lógica).

    - Política de Leitura: Ao consultar um documento (-c), o servidor verifica primeiro a cache. Se o documento estiver presente (cache hit), é retornado diretamente. Se não (cache miss), e o documento for encontrado no disco, ele é carregado para a cache se houver espaço, ou se a política de substituição o permitir (neste caso, a adição à cache após um miss do disco parece acontecer se houver espaço, conforme find_document).

- Justificação: A cache reduz a necessidade de acessos ao disco, que são significativamente mais lentos que o acesso à memória, melhorando o tempo de resposta para consultas a documentos "quentes". A política FCFS é simples de implementar e gerir.

# 5. Conclusão

O projeto permitiu consolidar bem os conhecimentos lecionados pela cadeira de Sistemas operativos, nomeadamente no uso de FIFOs, sinais e gestão de processos.

A arquitetura cliente/servidor implementada demonstrou ser eficaz para comunicação interprocesso, com capacidade para escalar a múltiplos clientes. Apesar da simplicidade da aplicação, foram feitas escolhas importantes quanto à robustez, tratamento de erros e gestão de recursos.

A nosso ver o projeto atingiu os objetivos propostos, permitindo uma gestão eficaz de pedidos entre processos através de técnicas sólidas de IPC.