**Introdução**

O intuito deste trabalho é implementar um serviço que permita a indexação e pesquisa sobre documentos de texto guardados localmente num computador. O programa servidor é responsável por registar meta-informação sobre cada documento (identificador único, título, ano, autor, localização), permitindo também um conjunto de interrogações relativamente a esta meta-informação e ao conteúdo dos documentos.

Os utilizadores utilizam um programa cliente para interagir com o serviço. Esta interação permite que os utilizadores adicionem ou removam a indexação de um documento no serviço, e que efetuem pesquisas (interrogações) sobre os documentos indexados. O programa cliente executa uma operação por invocação e não é interativo.

Estas funções serão explicadas detalhadamente no decorrer do relatório.

**Estrutura e Decisões tomadas**

**Cliente (dclient.c)**

No cliente começamos por identificar o tipo do pedido através dos argumentos da linha de comando. O cliente cria um pipe com nome para comunicar com o servidor, utilizando o seu PID para tornar esse pipe único. O pipe do cliente tem o nome "/tmp/client\_pipe\_[PID]" e serve para receber respostas do servidor após enviar requisições.

O cliente envia a requisição para o servidor através do pipe do servidor ("/tmp/server\_pipe") e aguarda a resposta no seu próprio pipe. Após receber a resposta, o pipe do cliente é fechado e removido, garantindo que não há resíduos de comunicações anteriores no sistema.

**Servidor (dserver.c)**

O servidor começa por inicializar sua estrutura interna, carregando documentos previamente indexados do disco. Em seguida, cria um pipe com nome ("/tmp/server\_pipe") que servirá como ponto de entrada para todas as requisições dos clientes.

O servidor mantém em memória um array de documentos indexados e uma cache para otimizar o acesso frequente. Quando recebe um pedido de um cliente, extrai o caminho do pipe do cliente da requisição para poder enviar a resposta de volta.

Dependendo do tipo de requisição, o servidor invoca a função correspondente para processá-la (adicionar documento, consultar, remover, contar linhas, pesquisar palavra-chave ou encerrar). Após processar a requisição, envia a resposta diretamente para o pipe do cliente.

Este processo é repetido continuamente até que seja recebido um comando de encerramento. A seguir serão descritas as funções principais do servidor:

**Estruturas de dados**

Utilizamos duas structs principais para gerenciar os dados do sistema:

1. Document - Armazena informações sobre cada documento indexado:
   * key: Identificador único do documento
   * title: Título do documento
   * authors: Autor(es) do documento
   * year: Ano de publicação
   * path: Caminho relativo ao documento
2. Cache - Implementa um sistema de cache para otimizar consultas frequentes:
   * entries: Array de documentos na cache
   * usage\_count: Contador de uso para implementar política LFU
   * size: Tamanho atual da cache
   * capacity: Capacidade máxima da cache

Para gerenciar a cache, implementamos as seguintes funções:

* init\_cache(int size): Inicializa a cache com o tamanho especificado
* get\_from\_cache(int key): Recupera um documento da cache
* add\_to\_cache(Document \*doc): Adiciona um documento à cache
* remove\_from\_cache(int key): Remove um documento da cache

**Comandos**

**Adicionar Documento**

O comando para adicionar um novo documento ao índice é:

$ ./dclient -a "título" "autores" "ano" "caminho"

Esta operação indexa a meta-informação do documento no servidor, atribuindo-lhe um identificador único. O documento físico deve já existir na localização especificada. O servidor armazena os metadados em memória e os persiste em disco para recuperação futura.

**Consultar Documento**

O comando para consultar informações sobre um documento indexado é:

$ ./dclient -c "key"

Ao receber este comando, o servidor busca primeiro na cache e, caso não encontre, busca no array principal de documentos. A resposta é então enviada ao cliente, que exibe os detalhes do documento.

**Remover Documento**

O comando para remover um documento do índice é:

$ ./dclient -d "key"

Esta operação remove apenas a indexação do documento, sem afetar o arquivo físico. O documento é removido tanto da memória quanto da cache, e as alterações são persistidas em disco.

**Contar Linhas com Palavra-chave**

O comando para contar linhas que contêm uma determinada palavra-chave em um documento é:

$ ./dclient -l "key" "keyword"

O servidor utiliza uma abordagem eficiente com pipes e o programa grep para realizar esta contagem, criando um processo filho que executa o comando e captura sua saída.

**Pesquisar Documentos com Palavra-chave**

O comando para encontrar documentos que contenham uma palavra-chave é:

$ ./dclient -s "keyword"

Esta operação pesquisa em todos os documentos indexados e retorna uma lista de identificadores dos documentos que contêm a palavra-chave especificada.

**Pesquisa Paralela**

Uma versão otimizada da pesquisa permite especificar o número de processos para realizar a busca em paralelo:

$ ./dclient -s "keyword" "nr\_processes"

O servidor divide os documentos entre vários processos filhos, cada um responsável por uma parte da coleção. Os resultados são então combinados pelo processo pai antes de serem enviados ao cliente.

**Desligar o Servidor**

O comando para encerrar o servidor é:

$ ./dclient -f

Quando recebe este comando, o servidor salva o estado atual do índice em disco antes de terminar, garantindo a persistência dos dados.

**Funcionalidades adicionais**

**Cache com política LFU (Least Frequently Used)**

Implementamos um sistema de cache utilizando a política LFU (Least Frequently Used). Quando a cache está cheia e um novo documento precisa ser adicionado, o documento com menor contador de uso é substituído.

Esta abordagem melhora significativamente o desempenho para consultas repetidas aos mesmos documentos, já que evita a necessidade de buscar os dados no armazenamento principal.

A dimensão da cache é configurável no momento de inicialização do servidor, permitindo ajustar o equilíbrio entre uso de memória e desempenho.

**Pesquisa Paralela Eficiente**

Nossa implementação de pesquisa paralela distribui eficientemente o trabalho entre múltiplos processos, usando pipes para comunicação entre processos pai e filhos.

Cada processo filho é responsável por um subconjunto dos documentos indexados, realizando a pesquisa de forma independente. Os resultados são então enviados de volta ao processo pai através dos pipes, que os combina antes de enviar ao cliente.

Esta abordagem permite um ganho significativo de desempenho em servidores com múltiplos núcleos, especialmente para grandes coleções de documentos.

**Conclusão**

Este projeto permitiu-nos aplicar os conhecimentos adquiridos na disciplina de Sistemas Operativos, especialmente no que diz respeito à comunicação entre processos usando pipes com nome, gerenciamento de processos, e acesso a arquivos.

Conseguimos implementar com sucesso todas as funcionalidades básicas e avançadas solicitadas, criando um sistema de indexação e pesquisa de documentos eficiente e robusto.

A implementação de otimizações como cache e pesquisa paralela demonstra a aplicação prática dos conceitos de eficiência e concorrência em sistemas operativos.

Embora o sistema atual atenda aos requisitos especificados, identificamos algumas áreas para melhoria futura, como uma interface gráfica para o cliente, suporte a formatos de arquivo adicionais, e implementação de algoritmos de busca mais avançados.

Agradecemos aos professores da disciplina pelos conhecimentos transmitidos e pelo apoio durante o desenvolvimento deste projeto.