Sistemas Operativos

Trabalho Prático

Serviço de indexação e Pesquisa de Documentos

Desenvolvido por:
Francisco Dias a108561

Pedro Campos a108482

Rodrigo Rocha a108649

LCC

Este relatório descreve a nossa implementação de um serviço cliente-servidor desenvolvido em C, cujo objetivo é a gestão de documentos de texto. Este projeto foi desenvolvido para a UC de Sistemas Operativos. Neste relatório justificámos alguma das nossas opções a desenvolver este projeto.

Arquitetura Geral do Sistema

O sistema é composto por dois processos principais:

- dserver: Processo do servidor que recebe os pedidos dos clientes e processa a informação

- dclient: Processo do cliente que envia os pedidos para o servidor

Estes processos comunicam entre si através de *pipes com nome*, que vamos referir neste relatório como FIFO's.

Cada cliente cria um FIFO exclusivo, baseado no seu PID, que garante que as respostas do servidor para vários clientes não se misturem. O servidor tem um FIFO fixo (*/tmp/dserver_fifo*) por onde recebe os pedidos dos clientes.

O cliente é apenas responsável por processar o comando que lhe é dado, construir e enviar o pedido e esperar pela resposta do servidor. O servidor é capaz de estar ligado num loop infinito enquanto aguarda por pedidos. Para facilitar o processo da construção do pedido e da resposta, decidimos criar duas structs, a struct Request e a struct Resposta. Alem disso, criámos um enum Operacao, que contém todas as operações que o servidor pode efetuar. Assim, dependendo do comando dado pelo utilizador, o cliente indica ao servidor que tipo de operação é e como deve agir.

Funcionalidades do Programa

- **-a:** Adiciona um novo documento, com título, autores, ano e o caminho do documento.
- **-c:** Consulta a meta-informação de um documento dando o seu id.
- **-d:** Remove a meta-informação de um documento.
- **-l:** Conta as linhas de um documento que contém uma determinada *keyword.*
- **-s:** Indica os documentos que contém uma determinada *keyword*. Esta opção pode ser executada concorrentemente por vários processos, indicando o número de processos que devem ser executados.
- **-f:** Encerra o servidor.

Comunicação entre cliente e servidor

O cliente começa por preparar a struct Request com os dados do seu pedido. Esta struct é preenchida de maneira diferente dependendo do tipo de pedido que o cliente efetuou. Na operação de adicionar um documento, a struct guarda o título, autor, ano e o caminho do documento. Nas operações de consultar ou apagar um documento, esta guarda o id do documento. Na operação de contar linhas guarda o id do documento e a keyword a procurar. Na operação de encontrar os documentos com uma keyword, guarda a keyword e o número de

processos que deve efetuar. Esta struct também guarda o tipo de operação que está a ser pedida, para comunicar ao servidor.

Após o Request estar pronto, o cliente abre o FIFO do servidor em modo escrita, envia o seu pedido e cria o seu FIFO privado em modo leitura para receber o pedido.

O servidor recebe o Request do cliente e cria um processo filho que trata do pedido. O uso de fork() permite que o servidor continue receber pedidos de novos clientes, enquanto que os processos filhos tratam os pedido de forma independente. Assim, garantimos que o servidor não bloqueia à espera que um pedido anterior seja processado.

A resposta está construída numa struct que contém a mensagem do servidor e um inteiro que indica se a operação foi sucedida ou não (1 indica operação sucedida, 0 indica falha).

Escolhemos processar os pedidos e as respostas através de structs, porque estas facilitam a organização da informação de maneira compacta. Alem disso, estas tornam o processo dos FIFO's mais eficientes com o write e read. Também temos a oportunidade de facilmente expandir as structs para adicionar novas operações ou campos.

O servidor mantém- se ativo até encontrar algum erro de leitura ou receber ordem do cliente para encerrar. Quando isto acontece fecha-se o FIFO de leitura do servidor e executa um "while wait(NULL) > 0)" para que o servidor espere que os seus filhos acabem todos para não haverem processos zombies.

Para melhorar o desempenho e evitar acessos redundantes ao disco, implementámos um sistema de cache no servidor. Esta cache guarda até N entradas de meta-informação dos documentos mais recentemente usados. Quando a cache fica cheia, esta remove a entrada menos usada, segundo a política LRU (Least Recently Used). Quando o cliente faz o pedido de consulta de um documento, o servidor primeiro procura na cache pelo documento antes de aceder ao ficheiro com a meta-informação no disco.

Para implementarmos a pesquisa concorrente, o servidor cria N filhos para dividir o trabalho de pesquisa de documentos. Cada filho executa "grep -q" num subconjunto dos ficheiros e comunica ao pai através de pipes. O processo pai junta os resultados e responde ao cliente com os documentos encontrados.

Também assegurámos a persistência da meta-informação dos documentos em disco, que permite reiniciar o programa sem perder os dados. Estes dados são guardados num ficheiro "documentos.dat".

Para concluir, achámos que este projeto põe em prática todos os conteúdos que fomos aprendendo ao longo do semestre e testou as nossas capacidades em Sistemas Operativos.