Universidade do Minho
Escola de Engenharia

Sistemas Operativos

Trabalho Prático

grupo17

2ºano - 2ºsemestre

Joana Branco (A96584)
Joana Pereira (A97588)
Marta Sá (A97158)

Braga, 29 de maio de 2022

Introdução

O problema consiste na implementação de um servidor em que seja possível fazer cópias de ficheiros inseridos pelo utilizador e, por consequente, guardar a mesma. Há funcionalidades específicas próprias para que isto se possa desenvolver tais como, a compressão e cifragem dos ficheiros em causa.

Deverá ser dada a oportunidade ao utilizador para armazenar ficheiros novos e diferentes dos já existentes mas também, conseguir reaver informação antiga. Para além disto, o programa deve ser capaz de produzir as tarefas que estão a ser executadas assim como, as estatísticas das mesmas.

Começou-se por separar o problema em dois pipes diferentes: o cliente (programa sdstore) e o servidor (programa sdstored). Neste caso, os pipes são designados como pipes com nome. O cliente permite que o utilizador possa inserir o que necessita pelo terminal. É através da linha de comandos que o utilizador informa a tarefa que o servidor deve executar. Por outro lado, o servidor retém o resultado do que é pedido pelo utilizador.

As diversas transformações que os ficheiros podem sofrer baseiam-se em:

- **bcompress/bdecompress** (comprime/descomprime dados com o formato bzip);
- **gcompress/gdecompress** (comprime/descomprime dados com o formato gzip);
- **encrypt/decrypt** (cifra/decifra dados);
- **nop** (copia dados sem realizar qualquer transformação).

Ou seja, dependendo do comando inserido na linha de comandos, o cliente irá agir. O cliente é o que vai ser responsável por enviar os pedidos ao servidor através dos seus parâmetros. Já o servidor irá “responder” ao cliente através da informação dada como output.

Para a execução do programa também foi criado um Makefile que permite que apenas com o comando make seja tudo executado em simultâneo.

Desenvolvimento

O ponto de partida destina-se a inicializar um servidor que recebe dois argumentos com informação a ser guardada. Desta forma, a estratégia adotada tem em conta que o primeiro parâmetro inserido corresponde ao caminho para um ficheiro de texto que indica o limite de cada uma das transformações. Exemplo disto é:

- ➔ no terminal do servidor: `./servidor txt ../transf/`
- ➔ no terminal do cliente: `./cliente proc-file txt2 txt3 nop nop`
- ➔ outra alternativa para o cliente: `./cliente status`

Para lidar com as transformações, decidimos criar um array de capacidade 7 onde cada posição corresponde a uma transformação. O array tem como objetivo guardar o número de instâncias de uma certa transformação para executar concorrentemente num determinado momento sendo por isso atualizado à medida que recebe e executa pedidos.

Já o segundo argumento faz referência à diretoria onde estão guardados os executáveis após as transformações em cima referidas. Esta informação foi guardada numa string.

Posteriormente, o pipe com nome responsável por receber a informação do cliente é inicializado no servidor para leitura a aguardar a receção de pedidos. De seguida, o servidor também terá que ser inicializado para poder fazer a escrita.

O comando **status** tem a função de informar sobre as tarefas que estão a ser executadas pelo servidor. Este permite que o utilizador possa solicitar a qualquer momento quais os pedidos de processamento que estão pendentes mas também, o estado de utilização das transformações. Para enviar ao servidor o pedido, este é criado a partir de uma string com informação relativa ao identificador do pipe concatenada com outra que contém o tipo de pedido (status, no caso) e um '\n' para identificar o final do mesmo. De seguida, o servidor processa o pedido e envia para o cliente as informações mencionadas anteriormente através de um pipe com nome inicializado pelo cliente.

Por sua vez, quando o pedido ao servidor é o comando **proc-file**, o programa cria uma string cujos argumentos, separados por um espaço, incluem novamente o identificador do pipe criado, seguido pelo tipo de pedido (proc-file, no caso), pelo ficheiro a ser processado, pelo destino do novo ficheiro, pelas várias transformações a serem executadas e, por fim, por um '\n' com o mesmo propósito daquele utilizado no pedido explicado anteriormente.

Se inserido o comando **proc-file**, vai ser devolvido o número de bytes que foram lidos do ficheiro e também os que foram escritos no ficheiro já transformado. Para este efeito de devolver o número de bytes dos ficheiros referidos, recorreu-se à função lseek. O servidor verifica se o pedido está de acordo com os limites impostos pelas transformações. Se existirem o número de instâncias necessárias para o pedido, este é executado. Caso contrário, o pedido fica pendente e vai ser adicionado a um array especificamente criado para estas situações. Para além disso, foi adicionada uma flag para indicar se o pedido pode ser executado (flag = 1) ou se fica pendente (flag = 0). No momento em que se recebe um pedido, o array com a informação dos limites das transformações vai ser alterado, diminuindo assim o número correspondente. Caso este último valor seja igual a 0 e ainda existirem argumentos relativos a essa mesma transformação, o pedido ficará pendente. Se for possível, o pedido vai ser executado e será enviada uma mensagem do tipo “finished” para o cliente no seu término. Criou-se de seguida um novo pedido (aviso) que manda ao servidor um aviso a informar que a execução do pedido terminou. Posteriormente, os limites de cada uma das transformações é incrementado, ou seja, volta a existir disponibilidade para uso. A partir disto, analisa-se o array dos pedidos pendentes e verifica-se se é possível executar algum deles. Será de novo verificado o valor da flag, 1 se poder ser executada e 0, caso contrário, e assim, o processo será semelhante ao indicado anteriormente. Na eventualidade de um deles ser executado, o mesmo será removido do array dos pedidos pendentes.

Conclusão

Este projeto permitiu desenvolver diferentes capacidades relacionadas com todo o conteúdo desenvolvido nas aulas da UC de Sistemas Operativos.

Tendo como retrospetiva os resultados obtidos pela execução do programa, podemos afirmar que o projeto podia ter corrido melhor. Achamos podíamos ter melhorado o projeto e temos noção de que não nos foi possível cumprir todos os objetivos propostos no enunciado. Os aspetos que ficaram aquém das nossas expectativas foram as prioridades do comando proc-file e o comando de término do servidor, o sinal SIGTERM.

Apesar de tudo, sentimos que o projeto contribuiu para melhorar a nossa interpretação em certos aspetos e aprofundar esses mesmos conhecimentos.