Universidade do Minho

Sistemas Operativos

Trabalho Prático

Grupo 36

Ano letivo 2021/2022

a96106, Miguel Silva Pinto

a97755, Orlando José da Cunha Palmeira

a97613, Pedro Miguel Castilho Martins

Introdução

Este relatório descreve a nossa implementação de um serviço que permite aos utilizadores armazenar os seus ficheiros de forma segura e eficiente, poupando espaço de disco e disponibilizando funcionalidades de compressão e cifragem de ficheiros.

Este serviço permite a submissão de pedidos de clientes para processar e armazenar ficheiros, consoante diferentes ordens dadas pelo cliente, bem como recuperar ficheiros guardados previamente.

Estrutura

Este serviço foi implementado com uma arquitetura cliente servidor, em que a comunicação entre eles é feita com pipes com nome. Um dos pipes é criado no início da execução do servidor com o nome de "**main_fifo**" e é responsável por receber os pedidos dos clientes para serem depois processados pelo servidor. Quando um cliente envia um pedido para o servidor, o cliente irá se encarregar de criar um outro pipe, com o nome de "**p(pid)**" de modo a que os nomes de cada pipes sejam únicos a cada cliente, que terá a função de ser o meio de comunicação da informação vinda do servidor (status, estado do request…) para ser posteriormente apresentada ao cliente.
O servidor quando recebe um pedido guarda-o numa fila de espera que guarda todos os pedidos feitos pelos clientes. Esses pedidos são atendidos assim que o servidor tenha a capacidade de os atender tendo em conta os limites impostos pelo o ficheiro de configuração do servidor.

Cliente

A implementação do cliente "**sdstore**" começa com a validação dos argumentos recebidos pelo programa. Caso o programa não receba argumentos ou os argumentos estejam errados, é escrito no **STDOUT** a lista de comandos válidos do programa de maneira a dar um pequeno guia de funcionamento ao cliente.

O cliente tem 2 comandos principais:
Status: O comando "**status**" tem a função de revelar ao cliente o estado do servidor, mostrando o limite de cada uma das transformações que o servidor pode executar e quantas estão a ser executadas nesse momento.
Quando é feito "**./bin/sdstore status**" é escrito no pipe "**s;**" que indica ao servidor que esta instrução é uma instrução de status e de seguida é escrito o nome do pipe criado para que o servidor retorne a informação a devolver ao cliente, sendo enviado uma mensagem do género **"s;tmp/p(pid do cliente)"**. O servidor irá interpretar esta mensagem mandando para o pipe criado pelo cliente, a informação do estado do servidor.

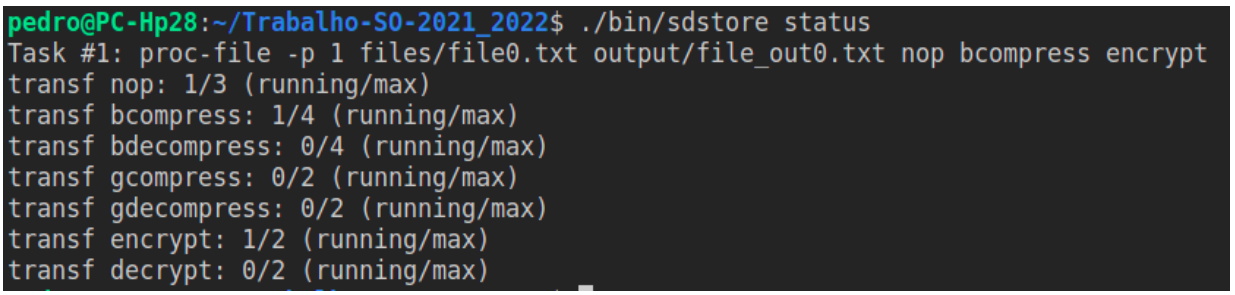

Fig. 1 - Instrução status.

Proc-file: O comando "**proc-file**" é o comando que envia pedidos ao servidor.
Esse comando tem esta sintaxe "(executável) proc-file -p <priority> input-filename output-filename transformation-1 transformation-2 ...".
Uma instrução proc-file pode ser enviada ao servidor com a flag de prioridade "-p", ou sem a flag e quando é enviada sem a flag de prioridade, é assumida uma prioridade padrão de 0.
O "**input-filename**" é a path do ficheiro ao qual vão ser aplicadas as transformações e "**output-filename**" é a path do ficheiro resultado dessas transformações.
De seguida podem ser definidas as transformações que queremos aplicar ao ficheiro e em que ordem através do seu nome (bcompress, bdecompres, encrypt, …).
Para dizer ao servidor que deseja fazer um determinado conjunto de transformações a um ficheiro, o cliente vai mandar pelo **main_fifo** (pipe que é usado para mandar requests ao servidor) uma mensagem com o seguinte formato:
"1;files/file0.txt;output/file_out0.txt;nop;bcompress;encrypt;end;tmp/p10623"
No primeiro campo da string, é indicada a prioridade do request com um número entre 0 e 5. No segundo e terceiro campo estão indicados respetivamente as paths dos ficheiros de input e output. Os restantes campos vão indicar as transformações a serem aplicadas ao ficheiro de input, com um campo "end" a determinar o final das transformações. O último campo irá conter o nome do pipe que o cliente criou para receber informação vinda do servidor sobre o estado do pedido.

Quando o servidor está a processar o pedido, escreve no pipe criado pelo cliente "processing" e o cliente escreve no **STDOUT** essa mensagem. No caso do pedido não poder ser atendido de imediato a mensagem "pending" é escrita. Quando o pedido termina de ser processado a mensagem "concluded (bytes-input: (nº bytes), bytes-output: (nº bytes))" é enviada ao cliente a indicar que o seu pedido terminou e o número de bytes do ficheiro inicial e final.

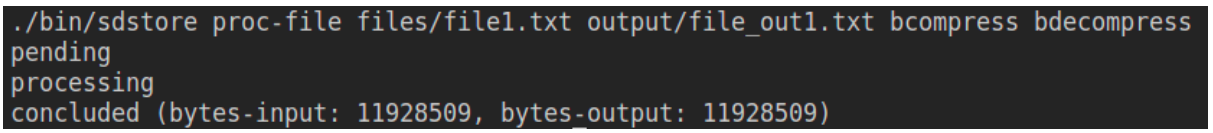

Fig. 2 - Instrução proc-file.

Servidor

Configuração:
O servidor no seu arranque irá retirar a informação do ficheiro de configuração para uma struct *transfs*, na qual vai indicar o número de transformações simultâneas máximas para cada transformação, bem como o caminho para os executáveis de cada transformação indicado no segundo argumento da chamada do executável do servidor.

Criação do main_fifo:
O pipe principal que servirá de meio de transporte dos pedidos é criado também no arranque do servidor e será posteriormente aberto.

Gestão de file descriptors:
De seguida irá criar um processo filho para abrir um file descriptor de escrita no **main_fifo** (pipe que transporta os pedidos dos clientes para o servidor) que irá fazer com que o servidor bloqueie no read, sempre que tente ler deste pipe e não houver ninguém para escrever (não houver nenhum file descriptor de escrita aberto), evitando estar num loop constante a verificar se tem pedidos para ler. Este processo-filho também se encarregará de mandar uma mensagem sem conteúdo de 5 em 5 segundos para o **main_fifo** de modo a que o servidor não bloqueie na libertação de pedidos e possa verificar se existe algum pedido que já possa começar a processar permitindo ao mesmo tempo que venham pedidos para o **main_fifo**.

Gestão de novos pedidos:
Quando uma mensagem chega através do **main_fifo**, vamos dividir a mensagem em vários requests, ou seja, cada request enviado por um cliente, tem um delimitador '\n' a indicar o fim do seu request, e quando o servidor ler o **main_fifo** existe a possibilidade de ler mais do que um request, daí a necessidade do parsing da mensagem em requests. Em seguida os requests serão processados quanto ao seu tipo de instrução.
Se for uma instrução status, será utilizada a função return_status que verifica a queue de requests (mais à frente explicada) e a struct *transfs,* devolvendo a string a enviar ao cliente com a informação relativa ao estado do servidor.
Se for uma instrução proc-file, vamos verificar se o request é exequível tendo em conta o número máximo de transformações definidas e o número de transformações do pedido. Por exemplo, se o limite máximo de bcompress fosse 2 e o request exigisse que fossem aplicadas 3 vezes a transformação bcompress, esse request nunca seria possível de ser processado, sendo retornada uma mensagem ao cliente sobre essa impossibilidade.
Caso o pedido seja válido, este será adicionado ordenadamente por ordem decrescente relativamente à sua prioridade a uma fila de espera representado por uma struct *requests* que é uma lista ligada em que cada nodo tem a informação relativa ao pedido.

```c
// Estrutura de dados que serve como Queue de pedidos enviados por clientes.
typedef struct requests {

    int task;           // identificador do número da task.
    int prio;           // prioridade do pedido.
    char * source_path;
    char * output_path;
    char ** transformations;    // array com as strings relativas à transformação
    int n_transformations;       // numero de elementos do array 'transformations'
    int mem;            // indica a memoria alocada no array 'transformations'
    char * ret_fifo;    // string que indica o pipe que devem ser enviadas mensagens de reply ao cliente
    pid_t pid;          // 0 (default value), -1 se já foi verificado para processar, >0 se está em processamento
    struct requests * next;

} *REQUEST;
```

Fig. 3 - Struct requests.

Atendimento e execução de pedidos:

Após a adição dos pedidos, estes vão passar pelo processo de atendimento, em que é verificado se é possível começar a processar o pedido, tendo em conta as transformações a serem utilizadas no momento. Se for possível, são alterados os valores indicativos das transformações a serem utilizados no momento, na struct *transfs,* e começa a execução do pedido através da criação de um processo-filho que irá se encarregar da execução do request através da função exec_request sendo atribuído ao campo “pid” desse pedido, o valor do pid do processo que está a tratar do seu request, sendo isto utilizado na gestão de remoção de pedidos, explicado mais à frente. Se não for possível, vai ser enviada uma mensagem ao cliente a informar que o pedido está pendente e será atribuído ao campo “pid” o valor -1, indicando que o processo já foi verificado para processar, mas ainda não está em execução.

Gestão de remoção de pedidos:

Após os pedidos serem atendidos, passamos a explicar a parte da remoção de pedidos da fila de espera, em que a fila é percorrida de maneira a verificar se podem ser removidos pedidos já atendidos.
O indicador do término da execução de um request será o campo “pid” que auxiliará na remoção do pedido da fila de espera quando já estiver concluída a sua execução, ou seja, se houver um pedido com o valor do campo “pid” maior que 0, significa que já entrou em processamento e irá ser usada a função waitpid com a flag WNOHANG, para verificar se o processo de execução deste pedido já terminou, não havendo bloqueio na espera do processo-filho devido à flag utilizada. Se o processo já deu exit, então significa que já terminou a execução do seu pedido, sendo esse pedido, removido da fila e são alterados os valores indicativos das transformações a serem utilizados no momento na struct *transfs*, caso contrário o pedido permanece.

Conclusão

O serviço criado possui todas as funcionalidades básicas e avançadas pedidas no enunciado, e com isso acreditamos que o nosso trabalho está bem conseguido.
Uma mudança que poderia ser feita na nossa implementação é o uso de uma lista ligada para a gestão dos pedidos que poderá não ser a estrutura mais eficiente, porém achamos que para este caso foi uma opção aceitável.
Como este trabalho foi feito ao longo do lecionamento das aulas, não conseguimos ter o serviço planeado a 100%, tendo de implementar certas funcionalidades sem saber todos os conceitos aprendidos em SO. Isto pode ter causado algumas inconsistências no desenvolvimento do programa.
Para terminar gostaríamos de dizer que este trabalho ajudou-nos a consolidar os conceitos obtidos nas aulas de SO e a aplicar esses conceitos.