Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Unidade Curricular de Sistemas Operativos

Ano Letivo de 2021/2022

SDStore: Armazenamento Eficiente e Seguro de Ficheiros

Hugo Moreira A43148

29 de maio de 2022

Resumo

Pretende-se implementar um serviço que permita aos utilizadores armazenar uma cópia dos seus ficheiros de forma segura e eficiente, poupando espaço de disco. Para tal o serviço disponibilizará funcionalidades de compressão e cifragem dos ficheiros a serem armazenados.

O serviço deverá permitir a submissão de pedidos para processar e armazenar novos ficheiros bem como para recuperar o conteúdo original de ficheiros guardados previamente. Ainda, deverá ser possível consultar as tarefas de processamento de ficheiros a serem efetuadas num dado momento e estatistícas sobre as mesmas.

Índice

Lista de Figuras

1 Introdução

Pretende-se implementar um sistema capaz de armazenar ficheiros e efetuar transformações com os mesmos. Deverá ser desenvolvido um cliente (programa sdstore) que ofereça uma interface com o utilizador via linha de comando. O utilizador poderá agir sobre o servidor através dos argumentos especificados na linha de comando deste cliente. Deverá ser também desenvolvido um servidor (programa sdstored), mantendo em memória a informação relevante para suportar as funcionalidades descritas neste enunciado.

O standard output deverá ser usado pelo cliente para apresentar o estado do serviço ou o estado de processamento do pedido.

Existem diferentes tipos de transformações que podem ser aplicadas:

- O registo deverá ser efetuado pelo utilizador de forma a este conseguir aceder à aplicação.
- 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.

As funcionalidades, a serem implementadas, suportadas pelo servidor são:

- Processamento das diversas tranformações mencionadas anteriormente.
- Estado das tarefas em processamento e dos recursos utilizados e disponíveis das transformações.
- Informar o cliente sobre a utilização do sistema.

Ao longo deste relatório iremos abordar a aquitetura definida de forma a suportar os pedidos dos diversos clientes de forma concorrente por parte do servidor. Medidas e decisões tomadas durante a implementação do projeto, comunicação entre processos assim como possíveis melhorias do sistema.

2 Arquitetura e Estruturas de Dados

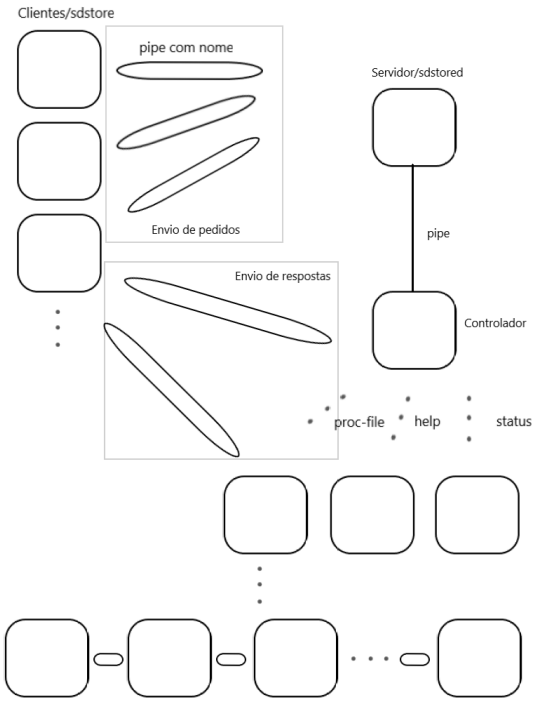

Figura 2.1: Arquitetura do sistema desenvolvido.

Os programas clientes poderão enviar os seus pedidos ao servidor através de um pipe com nome "fifo".

O programa servidor deverá suportar o processamento dos pedidos concorrentemente, para ser possível este comportamento, este irá receber os pedidos a processar e enviará o pedido para um processo controlador. Esta comunicação entre processos é estabelecida através de um pipe anónimo. Este controlador recebe o pedido e criará um outro processo que irá processar o pedido e irá enviar a resposta ao cliente.

A comunicação do servidor com os clientes é estabelecida através de um pipe com nome, de forma a garantir que os pipes são criados com diferentes nomes é enviado o identificador do processo (pid) nos pedidos, este pipe é criado pelo cliente assim que envia o pedido e fica a aguardar a resposta do servidor.

```c
typedef struct configs
{
    int nop;
    int bcompress;
    int bdecompress;
    int gcompress;
    int gdecompress;
    int encrypt;
    int decrypt;
} Configs;
```

Figura 2.2: Estrutura de dados utilizada para guardar recursos em utilização e disponíveis

```c
typedef struct queue
{
    int task_num;
    int args_num;
    int priority;
    char *clt_pid;
    char *args[MAX_TASKS];
    struct queue *next;

} Queue;
```

Figura 2.3: Estrutura de dados utilizada para guardar tarefas em execução e tarefas pendentes.

3 Implementação

Funcionalidades

Obter o estado de funcionamento do servidor, conforme o exemplo abaixo:

$ ./sdstore status

task #3: proc-file 0 /home/user/samples/file-c file-c-output nop bcompress

task #5: proc-file 1 samples/file-a file-a-output bcompress nop gcompress encrypt nop

task #8: proc-file 1 file-b-output path/to/dir/new-file-b decrypt gdecompress

transf nop: 3/3 (running/max)

transf bcompress: 2/4 (running/max)

transf bdecompress: 1/4 (running/max)

transf gcompress: 1/2 (running/max)

transf gdecompress: 1/2 (running/max)

transf encrypt: 1/2 (running/max)

transf decrypt: 1/2 (running/max)

Obter informação de utilização do programa cliente:

$ ./sdstore

./sdstore status

./sdstore proc-file priority input-filename output-filename transformation-id-1 transformation-id-2 ...

Executar o servidor:

$ ./sdstored config-filename transformations-folder Por exemplo,

$ ./sdstored etc/sdstored.conf bin/sdstore-transformations

Cliente

Na implementação do programa cliente foi tomada em atenção a verificação dos pedidos efetuados por este, de forma a evitar o envio de pedidos indesejados para o servidor e assim evitar que este tivesse tempo de execução com pedidos inválidos.

Após esta verificação, caso o pedido seja um pedido inválido será apresentado ao cliente as funcionaldiades do programa (help) e o programa irá terminar.

Sendo o pedido um pedido válido o programa cliente irá comunicar com o programa servidor através de um pipe com nome "fifo"e após este envio irá criar um pipe com nome com o seu identificador de processo (pid), abrindo um descritor de escrita seguido de um descritor de leitura do mesmo.

Esta implementação foi definida de forma ao cliente bloquear na escrita e ser desbloqueado assim que o servidor abra este mesmo descritor na leitura, garantindo assim que a escrita para o pipe apenas é efetuada quando este encontra-se aberto e evitando que haja alguém a tentar escrever para o pipe quando este está fechado (SIGPIPE).

Assim que o seu pedido seja efetuado e este receber a reposta do servidor, esta é enviada para o standard output e estes descritores são fechados, sendo feito também o unlink do pipe criado.

Servidor

O programa servidor ao iniciar é feita a leitura das configurações, que encontram-se em ficheiro, do mesmo e é estas configurações são guardadas em memória. De forma a efetuar a gestão destes recursos, foi criada uma estrutura de dados "Configs", o programa irá alocar memória para duas estruturas, uma com a informação das capacidades máximas de cada transformação e uma outra com as que encontram-se em utilização.

De seguida irá criar um processo controlador que irá fazer a processar os pedidos recebidos pelo servidor.

Após as configurações serem estabelecidas o servidor cria um pipe com nome "fifo", sendo utilizada a mesma implementação que no programa cliente de forma a evitar SIGPIPE. Ficando assim a ler os diversos pedidos do cliente e comunicando para o controlador processar os mesmos, ficando assim novamente disponível e a aguardar um novo pedido.

O processo controlador irá ler do pipe anónimo o que o servidor escreveu, após a leitura é feito "parse"da mensagem e consoante o pedido o controlador irá criar um outro processo que irá processar o mesmo, ficando assim disponível para receber um novo pedido, tendo estes os seguintes diferentes comportamentos:

- Status: Faz a manipulação das mensagens a serem enviadas ao programa cliente, colocando as informações das tarefas em execução que encontram-se guardadas numa estrutura de dados assim como a informação dos recursos que encontram-se em utilização e os disponíveis. Assim que estas mensagens encontram-se finalizadas é enviado para o pipe com nome onde o cliente encontra-se à escuta e o processo termina.

- Help: A informação dos comandos/funcionalidades do servidor encontra-se na memória deste processo e as mesmas são enviadas pelo pipe com nome do respetivo cliente e o processo termina.

- Proc-file: Após o controlador ter feito o parse e guarada a tarefa, em memória, numa estrutura de dados (Queue). Este processo, dependendo do número de transformações a serem feitas, irá criar n-1 processos e criar n-1 pipes anónimos para estabelecer a comunicação entre estes. O primeiro processo irá abrir o ficheiro a ser transformado e irá redirecionar o standard input para esse descritor, fechando o extremo de leitura do pipe e redirecionar o standard output para o extremo de escrita do pipe. Os processos intermédios vão fechar o extremo de leitura do pipe n e fazem o redirecionamento do standard input para o extremo de leitura do pipe n-1 e o standard output para o extremo de escrita do pipe n. O último processo redireciona o standard input para o extremo de leitura do pipe n-1 e o standard output para o descritor de ficheiro que o cliente enviou no seu pedido.

  Após as tranformações serem aplicadas e o ficheiro guardado é enviada uma mensagem para o controlador atualizar os recursos disponíveis, enviar uma mensagem ao cliente pelo pipe com nome do respetivo cliente, do estado do seu pedido. O controlador irá verificar se existe alguma tarefa pendente na estrutura de dados de tarefas pendentes, caso haja alguma tarefa pendente este verifica se os recursos necessários, para a execução da mesma, encontram-se disponíveis. Se existirem recursos disponíveis esta é removida da estrutura de dados, os recursos são atualizados e o controlador cria um processo para a sua execução e informa o respetivo cliente através de um pipe com nome.

4 Conclusão

Ao longo do projeto fomos verificando que certas decisões tomadas para a implementação do programa servidor poderiam ter sido diferentes, por exemplo o pipe de comunicação entre o servidor e o controlador poderia ter sido implementado por um pipe com nome de forma a evitar a necessidade de um ciclo "while"para a leitura de pedidos do servidor.

A manipulação do pedido pelo controlador e gestão das variáveis/memória poderia ter sido colocada no processo que executa a tarefa de forma a libertar o processamento da mesma do controlador e este apenas criar o processo para execução da tarefa.

As verificações dos pedidos feita pelo programa cliente poderia contemplar mais casos de pedidos indesejados. Apesar de já vários pedidos indesejados estarem a ser rejeitados do lado do programa cliente.

De referir a importância da gestão de recursos, processos, memória, etc aquando da definição da arquitetura de programas a serem desenvolvidos, principalmente quando existe concorrência de pedidos.