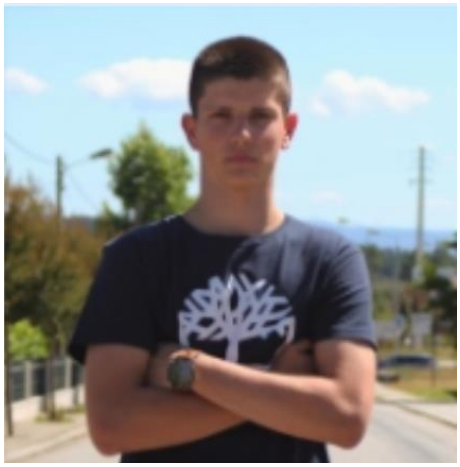
Universidade do Minho

Licenciatura em Engenharia Informática

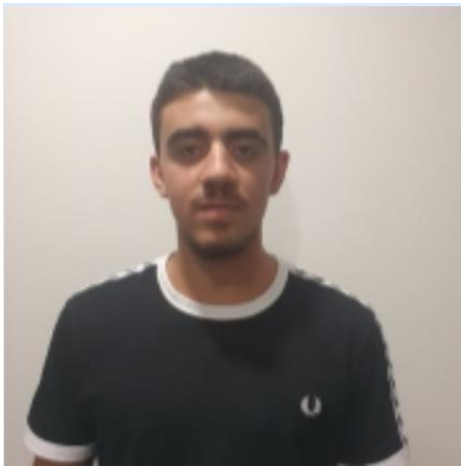
Sistemas Operativos

SDStore: Armazenamento Eficiente e Seguro de Ficheiros

Grupo 70



Afonso Bessa (A95225)



João Barroso (A95195)

Ricardo Gomes (A93785)

2021/2022

Índice

Introdução

O presente relatório descreve o trabalho prático: "*SDStore*: Armazenamento Eficiente e Seguro de Ficheiros" realizado no âmbito da cadeira de Sistemas Operativos (SO), ao longo do segundo semestre do segundo ano, da Licenciatura em Engenharia Informática da Universidade do Minho.

O objetivo do projeto foi implementar um serviço que permita aos utilizadores armazenar uma cópia dos seus ficheiros de forma segura, mas também eficiente, poupando desse modo espaço no disco. Para tal, o serviço disponibiliza funcionalidades de compressão e cifragem dos ficheiros a serem armazenados.

O serviço permite a submissão de pedidos para processar e armazenar novos ficheiros, bem como, recuperar o conteúdo original de ficheiros guardados previamente. É possível ainda consultar as tarefas de processamento de ficheiros a serem efetuadas num dado momento e estatísticas sobre as mesmas.

Neste documento procuramos descrever sucintamente o sistema desenvolvido e as suas principais funções, bem como as decisões tomadas durante a realização do projeto.

Descrição do Problema

a) Programas Cliente e Servidor

O projeto proposto implicou desenvolver:

- Um **cliente** - *sdstore* - que oferece uma interface com o utilizador via linha de comando. O utilizador pode agir sobre o servidor através dos argumentos especificados na linha de comando deste cliente.
- Um **servidor** - *sdstored* - mantido em memória a informação relevante para suportar as funcionalidades pedidas.

O *standard output* é usado pelo cliente para apresentar o estado do serviço ou o estado de processamento do pedido - *pending*, *processing*, *concluded*.

b) Transformações

Existem quatro diferentes tipos de transformações que podem ser aplicadas:

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

Resolução da Solução

Para a concretização do trabalho prático, dividimos o nosso programa em três ficheiros, cada um com uma determinada tarefa, o que contribuiu para uma melhor organização, gestão e manutenção do projeto.

Foi implementada uma arquitetura Cliente-Servidor com o intuito de distribuir processamento da informação entre o fornecedor do serviço – servidor - e os requerentes do serviço - clientes. Assim, os clientes enviam pedidos para o servidor e este, por sua vez, processa e envia os resultados dos respetivos pedidos. Note-se, que os clientes e servidor comunicam através de *pipes* com nome.

Desta forma, um dos pontos centrais serão os mecanismos de comunicação entre processos (IPC), assim como a utilização das diferentes *System Calls*.

Para controlo de erros foi também utilizada diversas vezes a função ***perror*** que imprime, se for o caso, uma mensagem de erro para o ***stderr***.

1. Cliente - Sdstore.c

1. Se o cliente receber o comando *status* ele passa ao servidor através de um pipe com nome a informação e o pid do processo do cliente -> (status pid).

```
sprintf(buffer, "%s %d", status, pid);

write(servidor_fifo, aux, strlen(aux)); // escreve no servidor fifo "status PID"
```

2. Se receber o comando *proc-file* passa ao servidor duas possibilidades dependendo se recebe a *flag -p*:
   a. *proc-file pid numTransformações prioridade source destination transformações*

```
if(strcmp(argv[2], "-p") == 0){
    numTransformacoes = argc - 6;

    for(int i = 6; i < argc; i++){
        strcat(trans, argv[i]);
        strcat(trans, " ");
    }

    sprintf(buffer, "proc-file %d %d %s %s %s ", pid, numTransformacoes, argv[3], argv[4], argv[5]);
```

   b. *proc-file pid numTransformações 0 source destination transformações*

```
numTransformacoes = argc - 5;

for(int i = 5; i < argc; i++){
    strcat(trans, argv[i]);
    strcat(trans, " ");
}

sprintf(buffer, "proc-file %d %d %s %s %s ", pid, numTransformacoes, "0", argv[3], argv[4]);
}
```

No primeiro caso a), a prioridade está definida, podendo ter um qualquer valor natural visto que temos a *flag -p*. Porém, no segundo caso b), está predefinida a sua prioridade sendo ela igual a zero independentemente do valor que receber como prioridade.

3. Servidor - Sdstored.c

O servidor inicialmente recebe o ficheiro de configuração e faz *parse* desse mesmo através da função *parse_config*. Após fazer parse das transformações irá guardar o nome da transformação e da sua capacidade máxima nas *struct* Transformação. Depois de realizar o parse do ficheiro de configuração, o servidor irá fazer o *parse* dos pedidos dos clientes através da função *parse_pedidos*, que irá receber pedidos através de um *pipe* com nome vindos dos clientes. Para executar os pedidos, o servidor irá ler toda a informação enviada pelo cliente, e executar o pedido dependentemente do tipo de informação recebida. O servidor termina quando receber um *signal* do tipo SIGTERM, tendo para esse propósito a função *sigterm_handler*.

4. Estruturas.c

Para além da Estruturas de Dados, Listas Ligadas, abordada no tópico seguinte, o Ficheiro Estruturas.c tem presente funções de manuseamento, inserção, criação de listas, como, por exemplo, criar Pedido ou Transformação, *createPedido* e *createTransf*, entre outras.

Estrutura de Dados

A Estrutura de Dados escolhida pelo grupo de forma a guardar as Transformações e/ ou os Pedidos foi **Listas Ligadas**.

```
typedef struct Transf {
    char *transf;
    int ocorr;
    int ocorrmax;
} * Transformacao;
```

Figura 1 - Transformações

A variável **transf** armazena o nome de uma Transformação. A variável **occorr** guarda o número atual de instâncias de uma Transformação. A variável **occorrmax** guarda o número máximo de instâncias de uma certa Transformação que pode executar concorrentemente num determinado período de tempo.

```
typedef struct Lista {
    Transformacao t;
    struct Lista *prox;
}* Lista;
```

Figura 2 - Lista de Transformações

A variável **t** armazena uma Transformação com a estrutura supramencionada. Como é uma lista ligada, temos também um apontador para o próximo nodo da lista de Transformações.

```
typedef struct Pedidos {
    char *mutacoes;
    int nTransf;
    int idCliente;
    int pid;
    int prioridade;
    char *source;
    char *destination;
} * Pedidos;
```

Figura 3 - Pedidos

A variável **mutações** corresponde a uma *strings* de Transformações separadas por espaços. A variável **nTransf**, tal como o próprio nome indica, **guarda** o número de Transformações existentes. A variável **idCliente** armazena o id do Cliente em questão. A variável **pid** armazena o *pid* do pedido. A variável **prioridade** indica a prioridade do Pedido. A variável **source** corresponde à origem do Pedido. A variável **destination** corresponde ao destino do Pedido.

```
typedef struct ListaPedidos {
    Pedido p;
    struct ListaPedidos *prox;
}* ListaPedidos;
```

Figura 4 - Lista de Pedidos

A variável **p** armazena um Pedido com a estrutura em cima abordada. Como é uma lista ligada, temos também um apontador para o próximo nodo da lista de Pedidos.

Funcionalidades

O serviço suporta as seguintes funcionalidades:

- O programa servidor recebe dois argumentos pela linha de comando:
  1. O primeiro, com o **identificador** da transformação que corresponde ao caminho para um ficheiro de configuração e o **número máximo** de instâncias de uma certa transformação que podem executar concorrentemente num determinado período de tempo.
  2. O segundo, corresponde ao **caminho** para a pasta onde os executáveis das transformações estão guardados.
- Cada cliente tem a opção de solicitar o processamento e armazenamento de um ficheiro - *proc-file* - através da aplicação no servidor de uma sequência de transformações.
- O utilizador é informado, caso o pedido tenha ficado **pendente**, quando entra em **processamento**, e quando é **concluído**. O comando termina quando o resultado está disponível.
- O servidor suporta o processamento concorrente de pedidos. Quando um pedido começa a executar, todas as transformações a serem usadas pelo mesmo devem ser identificadas como estando em utilização. O processamento de cada pedido não poderá ser iniciado enquanto, para alguma das transformações necessárias para a sua execução, o número de instâncias a correr esteja no limite máximo.
- O cliente tem a opção de consultar, através do comando ***status*** e a qualquer instante, o estado de funcionamento do servidor.
- Quando um pedido termina, é reportado ao cliente o número de bytes recebidos e o número de bytes produzidos por operação.
- Se receber o sinal *SIGTERM*, o servidor termina de forma graciosa, deixando primeiro terminar os pedidos em processamento ou pendentes. Atribuímos por *default* um máximo de 5 pedidos, ou seja, após esses pedidos é enviado um sinal *SIGTERM* ao servidor.
- O servidor dá prevalência a pedidos com maior prioridade. Para realizar esta funcionalidade, cada operação é acompanhada da sua prioridade.

Testes

- Resultado do *status*:

```
srcast@srcast-Nitro-AN515-51:~/Desktop/SO_2122/bin$ ./sdstore status
task #1: proc-file -1 ../samples/sample2.txt ../outputs/resposta2.bin bcompress
nop gcompress
task #2: proc-file -1 ../samples/1200mb-file ../outputs/resposta1.bin bcompress
nop gcompress
transf nop: 0/3 (running/max)
transf bcompress: 0/4 (running/max)
transf bdecompress: 0/4 (running/max)
transf gcompress: 0/2 (running/max)
transf gdecompress: 0/2 (running/max)
transf encrypt: 0/2 (running/max)
transf decrypt: 0/2 (running/max)
```

- Resultado de Comprimir e Descomprimir um Ficheiro:

```
srcast@srcast-Nitro-AN515-51:~/Desktop/SO_2122/bin$ ./sdstore proc-file 3 ../sam
ples/sample2.txt ../outputs/resposta2.bin bcompress nop gcompress
Pending
Processing
Concluded (bytes-input: 2311, bytes-output: 189)
```

```
srcast@srcast-Nitro-AN515-51:~/Desktop/SO_2122/bin$ ./sdstore proc-file 3 ../out
puts/resposta2.bin ../outputs/resposta3.bin gdecompress nop bdecompress
Pending
Processing
Concluded (bytes-input: 189, bytes-output: 2311)
```

- Interface do Servidor:

```
srcast@srcast-Nitro-AN515-51:~/Desktop/SO_2122/bin$ ./sdstored ../etc/transf.con
fig ../bin/sdstore-transformations/
SERVIDOR INICIADO COM O PID 12802

Transformacoes e a sua capacidade:
    nop: 3
    bcompress: 4
    bdecompress: 4
    gcompress: 2
    gdecompress: 2
    encrypt: 2
    decrypt: 2
Pedido referente ao PID 12811 concluído com sucesso
Pedido referente ao PID 12824 concluído com sucesso
Pedido referente ao PID 12836 concluído com sucesso
Pedido referente ao PID 12845 concluído com sucesso
Pedido referente ao PID 12855 concluído com sucesso

Recebido o sinal SIGTERM

SERVIDOR ENCERRADO
```

Conclusão

Face ao problema apresentado, e analisando criticamente a solução esperada, concluímos que cumprimos as tarefas propostas, conseguindo atingir os objetivos definidos, uma vez que foram implementadas todas as funcionalidades, quer básicas, quer avançadas.

Assim, após a elaboração deste trabalho prático, pode-se afirmar que aprimoramos o nosso domínio sobre a Linguagem de Programação C, nomeadamente os conceitos apresentados nas aulas da unidade curricular, tais como, as *system calls*, *pipes*, *fork*, *exec*, *read*, *write*, entre outras, tendo sempre em consideração o melhor desempenho possível dos programas.

Em suma, entregamos este trabalho com o sentimento de dever cumprido, pois, após várias e longas horas de trabalho e muitas dores de cabeça atingimos o melhor resultado possível.