



Universidade do Minho

# Sistemas Operativos

MIEI - 2º ANO - 2º SEMESTRE

UNIVERSIDADE DO MINHO

## Sistema de Backup

BRUNO CANCELINHA

A75428

MARCELO MIRANDA

A74817

RUI VIEIRA

A74658

21 de Maio de 2016

# *Conteúdo*

|          |                                     |          |
|----------|-------------------------------------|----------|
| <b>1</b> | <b>Introdução</b>                   | <b>2</b> |
| <b>2</b> | <b>Funcionalidades</b>              | <b>3</b> |
| 2.1      | Backup . . . . .                    | 3        |
| 2.2      | Restore . . . . .                   | 3        |
| 2.3      | Delete . . . . .                    | 3        |
| 2.4      | Global Clean . . . . .              | 3        |
| <b>3</b> | <b>Makefile</b>                     | <b>4</b> |
| 3.1      | make . . . . .                      | 4        |
| 3.2      | make clean . . . . .                | 4        |
| 3.3      | make stop . . . . .                 | 4        |
| 3.4      | make install . . . . .              | 4        |
| 3.5      | make uninstall . . . . .            | 4        |
| <b>4</b> | <b>Raiz do Backup</b>               | <b>5</b> |
| 4.1      | data/ . . . . .                     | 5        |
| 4.2      | metadata/ . . . . .                 | 5        |
| 4.3      | paths/ . . . . .                    | 5        |
| <b>5</b> | <b>Comunicação cliente/servidor</b> | <b>6</b> |
| 5.1      | Pipe do servidor . . . . .          | 6        |
| 5.2      | Sinais . . . . .                    | 6        |
| 5.3      | Pipe do cliente . . . . .           | 6        |
| <b>6</b> | <b>Mensagem</b>                     | <b>7</b> |
| 6.1      | A estrutura . . . . .               | 7        |
| 6.2      | empty_message . . . . .             | 8        |
| 6.3      | init_message . . . . .              | 8        |
| 6.4      | change_message . . . . .            | 8        |
| 6.5      | freeMessage . . . . .               | 8        |
| <b>7</b> | <b>Conclusão</b>                    | <b>9</b> |

# 1. *Introdução*

Este projeto foi realizado no âmbito da disciplina de *Sistemas Operativos* e tem como objetivo a criação de um sistema de cópias eficiente, que guarda ficheiros dados por um utilizador. Estes são então comprimidos, reduzindo o espaço por eles ocupados. Temos também de considerar a privacidade de dados mantendo uma arquitetura cliente/servidor impedindo o acesso direto do cliente à pasta de backup.

O trabalho parecia ser fácil à primeira vista mas o nosso grupo não estava a considerar toda a dificuldade duma arquitetura com processos concorrentes, o que nos levou a adquirir uma nova maneira de pensar.

O projeto tem então duas funcionalidades principais. O ***backup*** que se responsabiliza por comprimir os ficheiros e salva-los na pasta que viremos a chamar *raiz do backup*. O ***restore*** que simplesmente descomprime o ficheiro e o devolve na sua diretoria original. Vamos de seguida explicar estes dois com mais profundidade.

## 2. Funcionalidades

### 2.1 Backup

A funcionalidade *backup* é a principal de todo o trabalho. Aos olhos do utilizador apenas guarda o ficheiro ou todo o conteúdo de uma pasta, mas visto de mais perto, é bem mais complexo.

Primeiro, o cliente terá que enviar todo o conteúdo do ficheiro a salvar em blocos de 4kbytes até o ficheiro estar completamente transferido para o servidor. Este terá então de lhe atribuir um *digest* gerado pelo *sha1sum*, comprimi-lo na pasta *data* usando o comando *gzip* e alterar o seu nome para esse *digest*. É também guardado na pasta *metadata* um *link simbólico* com o nome original do ficheiro ligado ao ficheiro correspondente em *data*, para além de criar outro *link simbólico* na pasta *paths* ligado à diretoria original desse ficheiro com o *path* original do ficheiro, para que este possa ser corretamente recuperado mais tarde. Quando o backup estiver concluído o servidor envia um sinal de sucesso ou de erro ao cliente.

### 2.2 Restore

Esta funcionalidade complementa o backup, permitindo-nos reaver os ficheiros guardados.

O restore começa por ler dos *links simbólicos* do *metadata* o *digest* correspondente ao conteúdo que pretendemos recuperar da pasta *data*. A partir deste, é criada uma cópia do conteúdo para que possamos descomprimir o ficheiro sem comprometer futuros *restores*. Após ser descomprimido, o conteúdo é enviado para o cliente, em conjunto com o *path* original do ficheiro, este é por fim, lá montado.

### 2.3 Delete

O comando *delete* apaga a entrada do ficheiro da *raiz do backup*. Para isso, apenas apaga o ficheiro de *metadata/* e de *paths*, mantendo o conteúdo comprimido em *data/*.

### 2.4 Global Clean

O *Global Clean* é chamado pelo nome de *gc*, remove todos os conteúdos em *data/* que não estão a ser ligados por nenhum ficheiro em *metadata/*.

## 3. *Makefile*

### 3.1 **make**

Compila o cliente para o ficheiro *client* e servidor para *server*.

### 3.2 **make clean**

Apenas limpa os executáveis criados pelo *make*.

### 3.3 **make stop**

Para todos os processos de *sobusrv*.

### 3.4 **make install**

Instala o *sobucli* e *sobusrv*. Este comando irá necessitar de permissões *sudo* pois instala estes executáveis diretamente na pasta */bin*. O nosso grupo questionou-se sobre colocar na diretoria */bin* ou alterar o *PATH* de *.bashrc*, acabamos por escolher a primeira pois a segunda alternativa não iria funcionar com nenhum de nós visto que o nosso path está guardado em *.zshrc* do *zsh*, uma shell alternativa à *bash* habitual. Mantemos então o instalador a copiar para a pasta */bin* para manter compatibilidade.

### 3.5 **make uninstall**

Pareceu-nos importante, depois de ter um instalador, ter também um desinstalador. O *make uninstall* apenas remove *sobusrv* e *sobucli* da pasta */bin*.

## 4. *Raiz do Backup*

A *raiz do backup* é uma pasta que se encontra na *home* do utilizador que corre o servidor. Dentro dela, encontra-se o *pipe* que servirá de comunicação cliente/servidor e, ocasionalmente, um conjunto de outros *pipes* para servir de comunicação servidor/cliente necessária para o comando *restore*, também uma pasta *data/*, *metadata/* e *paths*.

### 4.1 *data/*

Na pasta *data/* encontra-se todos os ficheiros comprimidos com o comando *gzip*, para além disso, o nome dos ficheiros são o seu *digest* criado por *sha1sum*. Deste modo é fácil descobrir ficheiros repetidos quando é chamado um novo *backup*. O comando *delete* não apaga nenhum destes ficheiros, para isso encontra-se designado o *gc* que limpa todos os ficheiros de *data/* que não estão ligados por *metadata/*.

### 4.2 *metadata/*

Na pasta *metadata/* estão guardados os *links simbólicos* que ligam o nome do ficheiro ao seu conteúdo em *data/*. O comando *delete* apenas apaga estes *links*.

### 4.3 *paths/*

Tal como a pasta *metadata/*, a pasta *data/* contém *links simbólicos* que ligam o nome do ficheiro ao seu *path* original para depois ser usado no *restore*.

## 5. Comunicação cliente/servidor

### 5.1 Pipe do servidor

O servidor cria um pipe que se encontra na *raiz do backup* com o nome *sobupipe*. O servidor fica numa espera passiva até que o cliente transmita as operações que pretende que o primeiro execute. Estes pedidos são enviados em forma de *mensagem*, uma estrutura que iremos discutir de seguida.

### 5.2 Sinais

Usamos sinais principalmente para o servidor notificar o cliente que o seu ficheiro já foi processado. Para sucesso, o servidor envia *SIGUSR1* para erro envia *SIGUSR2*. O utilizador ao receber cada um dos sinais escreve no ecrã a mensagem correspondente. Conforme o exemplo:

```
a.txt: copiado  
b.txt: erro ao copiar
```

### 5.3 Pipe do cliente

Ao fazer o *restore* é necessário um outro *pipe* que transfira, com estruturas do tipo mensagem, o documento descomprimido para o cliente, que o monta na sua localização original. Será criado na *raiz do backup* pelo servidor com o nome que do *pid* do processo que lhe enviou o pedido. Este *pipe* é imediatamente removido depois de já não ser necessário.

## 6. *Mensagem*

```
#define CHUNK_SIZE 4096
#define PATH_SIZE 1024

#define BACKUP 0
#define RESTORE 1
#define DELETE 2
#define GC 3

#define NOT_FNSHD 1
#define FINISHED 0
#define ERROR -1

typedef struct message {
    char chunk[CHUNK_SIZE];
    char file_path[PATH_SIZE];
    int operation;
    int status;
    int chunk_size;
    pid_t pid;
    uid_t uid;
} *MESSAGE;
```

### 6.1 A estrutura

Um ***chunk*** é um *array* de 4kbytes que terá ***chunk\_size*** bytes do ficheiro a transferir pelos *pipes*.

O *path* do ficheiro está guardado na *String* ***file\_path***.

A operação a efetuar está especificada no inteiro ***operation***. Existem defines para cada tipo de operação, portanto o ***operation*** pode estar para *BACKUP*, *RESTORE*, *DELETE* ou *GC*.

O inteiro ***status*** comunica o estado do ficheiro, este pode ser *NOT\_FINISHED* caso falem mais *chunks* para carregar o ficheiro, *FINISHED* caso tenha terminado de carregar todo o ficheiro, e *ERROR* caso tenha ocorrido um erro na leitura do ficheiro.

Finalmente temos o ***pid*** do processo que mandou a mensagem.



## **6.2    `empty_message`**

Apenas aloca o espaço para uma nova mensagem. Os campos desta nova mensagem não estarão tratados, sendo, por tanto, valores aleatórios.

## **6.3    `init_message`**

Para além de alocar uma nova mensagem, preenche todos os seus campos com os valores passados nos argumentos.

## **6.4    `change_message`**

Altera uma mensagem dada. Para tal, a mensagem passada nos argumentos deve estar inicializada, sendo assim populada com os valores passados pelos argumentos.

## **6.5    `freeMessage`**

Liberta o espaço alocado em memória pela mensagem.

## 7. Conclusão

Este projeto desenvolveu-nos uma capacidade de pensar numa arquitetura com vários processos concorrentes para além de nos deixar mais à vontade com o sistema Unix e a sua interação com a linguagem C.

Apesar de não estar em desagrado com o nosso trabalho, há algumas funcionalidades que gostaríamos de ter implementado.

A habilidade de poder gravar ficheiros com o mesmo nome, por exemplo, o ficheiro *a.txt* da pasta */Desktop* e outro como o mesmo nome em */Documents*. Que não nos foi possível efetuar devido à ambiguidade que provinha de chamar *restore* no tal ficheiro *a.txt*, mantendo a simplicidade do comando.

Seria também interessante a possibilidade de manter as várias versões do mesmo ficheiro, apresentando um histórico ao utilizador cada vez que este fosse correr o *restore* nesse ficheiro, tornando este mais poderoso. Não se concretizou para manter a simplicidade do programa.

Finalmente, o cliente deveria calcular o *shasum*, assim não seria necessário transferir o ficheiro todo para o servidor para de seguida concluir que já existe uma versão deste lá comprimida. A princípio pensamos que talvez não fosse correto o cliente executar comandos (neste caso o *shasum*), quando decidimos que o devia fazer e enviar para o servidor o *digest*, já foi muito em cima da hora e decidimo-nos focar em aspetos mais importantes.

Foi sem dúvida um trabalho bastante interessante que nos despertou um grande interesse no ramo de *Sistemas Operativos*.