

**Universidade do Minho**

Escola de Engenharia

Mestrado Integrado em Engenharia Informática

**Sistemas Operativos**

Ano Letivo de 2015/2016

**Backup Eficiente**

**A27748 - Gustavo José Afonso Andrez**

**A74634 - Rogério Gomes Lopes Moreira**

**A67664 - Samuel Gonçalves Ferreira**

20 Maio 2016

# Índice

Índice ii

1. Introdução 1

2. Descrição dos ficheiros desenvolvidos 2

2.1. Inicialização (init.sh) 2

2.2. Cliente (sobucli.c) 2

2.2.1 Pedido de *backup* 3

2.2.2 Pedido de *restore* 4

2.2.3 Pedido de *delete* 5

2.3. Server (sobusrv.c) 5

2.3.1 *Backup* 6

2.3.2 *Restore* 8

2.3.3 *Delete* 8

2.4. Makefile 9

2.5. Comentários 9

2.6. Resultados 9

3. Conclusão 11

1. Introdução

O objetivo deste trabalho é, recorrendo às competências desenvolvidas nas várias disciplinas do curso, em particular , na discilina de Sistemas Operativos, conceber uma aplicação que permita ao utilizador efetuar, de um modo eficiente, cópias de segurança (*backup*) de ficheiros ou pastas bem como restaurar (*restore*) ficheiros/pastas salvaguardados.

O sistema desenvolvido deverá apresentar características de privacidade dos dados.

1. Descrição dos ficheiros desenvolvidos

Neste ponto do trabalho pretendemos apresentar de um modo sintético mas completo, as funcionalidades desenvolvidas no projeto de forma a que o código gerado seja mais facilmente interpretado.

* 1. Inicialização (init.sh)

Para primeira utilização do programa é necessário executar o script de instalação através do comando:

sh init.sh

Com esta execução é criada no diretório home/<user>/. a pasta .backup/ e, dentro desta, as pastas data/ e metadata/. É também criado em .backup/ o *pipe* que permite a comunicação entre o cliente e o servidor. O código é apresentado em seguida:

#!/bin/bash

mkdir ~/.backup

mkdir ~/.backup/data

mkdir ~/.backup/metadata

mkdir ~/.backup/pipe

* 1. Cliente (sobucli.c)

O cliente pode solicitar ao server um *backup*, *restore* ou *delete* de um ficheiro. Abaixo são explicadas as implementações destes pedidos por parte do cliente.

Estas solicitações têm de seguir os formatos:

sobucli backup <filePath(s)>

sobucli restore <filename(s)>

sobucli delete <filename(s)>

Neste programa, no caso de o número de argumentos fornecidos (argc) ser menor que 3 o pedido não está corretamente enunciado. Esta é a primeira verificação e, neste caso, é solicitado ao utilizador reintrodução do pedido no formato correto.

No caso em que o número de argumentos fornecidos (argc) for maior ou igual a três, mas com o segundo argumento inválido (diferente de '*backup*', '*restore*' e ’*delete*’), o utilizador recebe a informação que o comando é inválido e também o formato correto do pedido.

A comunicação do servidor para o cliente é feita através dos seguintes sinais:

*30 - sucesso*

*10 - ficheiro não encontrado*

*6 - já existe backup do ficheiro*

Assim, o cliente está preparado para receber os três sinais da seguinte forma:

signal(30,hand);

signal(10,hand);

signal(6,hand);

A função *hand* recebe o sinal e imprime a informação correspondente.

* + 1. Pedido de *backup*

Para testar se o pedido do utilizador corresponde a um *backup* é comparada a string “backup” com o segundo argumento fornecido (argv[1]).

As verificações e processos descritos em seguida são aplicadas a cada um dos argumentos restantes (os ficheiros que se pretende copiar).

A verificação da existência do ficheiro é feita através da variável int status que guarda o resultado do stat do ficheiro.

status = stat (argv[i],&st\_buf);

Se status==0, o utilizador é informado que o ficheiro não existe e, portanto, não poderá ser feito o *backup*.

Caso contrário, é feito um teste para concluir se trata-se de uma diretoria ou de um ficheiro:

if (S\_ISDIR (st\_buf.st\_mode))

Assim, no caso de ser uma diretoria, o utilizador é avisado que terá de fornecer um ficheiro e não um diretório.

No caso do argumento ser um ficheiro, o pedido de backup é enviado ao servidor (nesta altura já se tem a garantia da existência do ficheiro em questão).

O pedido de *backup* pelo cliente ao servidor é realizado através do *pipe* sendo enviada uma string com o seguinte formato:

"dirFicheiro" B "myPID"

em que:

dirFicheiro – caminho (path) do ficheiro a ser guardado;

B – informa que o pedido é de *backup*;

myPID – pid do processo do cliente que faz o pedido;

Depois de escrita a linha no *pipe*, o processo entra em espera ( pause() ) até receber um sinal do servidor – sinal este que indica o estado da operação. Os sinais possíveis na situação de backup são :

s==30 – foi feito o *backup* corretamente;

s==6 – não foi realizado *backup* por já existir um ficheiro com o mesmo nome;

* + 1. Pedido de *restore*

Para testar se o pedido do utilizador corresponde a um *restore* é comparada a string “restore” com o segundo argumento fornecido (argv[1]).

Uma vez que o cliente não tem acesso ao diretório do *backup*, cabe ao servidor testar a existência do ficheiro solicitado, então, é simplesmente escrita no *pipe* uma string com o seguinte formato:

"nomeFicheiro" R "myPID"

em que:

nomeFicheiro – nome do ficheiro a ser guardado;

R – informa que o pedido é de *restore*;

myPID – pid do processo do cliente que faz o pedido;

Depois de feita a escrita no pipe, o processo entra em espera (pause()) até receber um sinal do servidor – sinal este que indica o estado da operação. Os sinais possíveis na situação de restore são:

s==30 – foi feito o *restore* corretamente;

s==10 – não foi realizado *restore* por não existir o ficheiro solicitado;

* + 1. Pedido de *delete*

A implementação do pedido de *delete* é em tudo idêntica ao pedido de *restore* sendo a informação enviada ao server pelo *pipe*:

"nomeFicheiro" D "myPID"

em que:

nomeFicheiro – nome do ficheiro a ser apagado;

D – informa que o pedido é de *delete*;

myPID – pid do processo do cliente que faz o pedido;

* 1. Server (sobusrv.c)

Neste programa são declaradas diversas variáveis necessárias para o armazenamento temporário e/ou comunicação de dados entre o processo principal e os seus processos-filho;

Depois de declaradas as variáveis, o servidor entra num ciclo (while (1)) de leitura do *pipe*.

Como já referido, a informação a que poderá aceder é do formato:

"dirFicheiro" X "PID"

em que:

dirFicheiro – nome/caminho do ficheiro a ser processado

X – corresponde ao caracter 'B' (*backup*), 'R' (*restore*) ou 'D' (*delete*)

PID – pid do processo do cliente que faz o pedido;

Quando recebida informação do *pipe*, esta é desmembrada e guardada nas seguintes variáveis:

char dir – dirFicheiro

char op – X ('B', 'R' ou 'D')

int pid – myPID

* + 1. *Backup*

Este processo é iniciado no caso de op=='B'.

Em seguida é calculado o *digest* do ficheiro recorrendo ao programa *sha1sum* e, através do comando *cut*, é retirada apenas a informação necessária (*digest*). Este procedimento é feito através do comando (execlp):

sha1sum "dirFicheiro" | cut -d ' ' -f1

Para testar se já existe um ficheiro com a mesma designação na diretoria *metadata*, é executado o seguinte comando:

find "metadata" -name "nomeFicheiro"

Em caso positivo, encerra-se (*kill*) o processo (*pid*) e é enviado o sinal “6” para o cliente, que corresponde à informação de que já existe um ficheiro com o mesmo nome:

kill(pid,6)

Em caso negativo, é feita a verificação se já existe *backup* do ficheiro no diretório *data*. Esta verificação é feita através da função:

temBackup (int digest)

Caso ainda não exista (*digest* não presente), é feita a compressão do ficheiro através do programa *gzip* e o ficheiro *.gz* gerado é guardado na diretoria *data*, através do comando:

gzip -k -c "dirFicheiro" > /home/<user>/.backup/data/"digestFicheiro".gz

O ficheiro é guardado tendo como nome o *digest* gerado.

Em seguida é criada uma ligação para o ficheiro com o nome original do ficheiro e esta é guardada na diretoria *metadata*.

ln -s /home/<user>/.backup/data/"digestFicheiro".gz /home/<user>/.backup/metadata/"nomeFicheiro"

No caso de já existir um *backup* na diretoria data de um ficheiro com o mesmo *digest*, apenas é criado o ficheiro ligação para esse ficheiro.

Finalmente encerra-se (*kill*) o processo (*pid*) e é enviado o sinal “30” para o cliente, que corresponde à informação de que o backup foi executado com sucesso:

kill(pid,30)

Caso não o pedido não seja de *backup*, terá de ser necessariamente de *restore* ou *delete*. Em ambos os casos terá de ser feita uma verificação se o ficheiro existe. Assim, é feita uma verificação se o nome do ficheiro existe na diretoria metadata/ através do comando:

find /home/<user>/.backup/metadata -name "nomeFicheiro"

No caso em que o ficheiro não existir, a variável int aux guarda o valor nulo. Nesta situação não é possível fazer o *restore* ou *delete* e encerra-se (*kill*) o processo (*pid*) e é enviado o sinal “10” para o cliente, que corresponde à informação que o ficheiro pretendido não existe:

kill(pid,10)

No caso em que o ficheiro existe (aux!=0) é guardada na variável char digest o *digest* correspondente do ficheiro. Este processo é realizado através do comando:

ls -l /home/<user>/.backup/metadata | grep "nomeFicheiro" |

cut -d ' ' -f11

Nesta altura é feita a verificação se se trata de um pedido de *restore* (op=’R’). Em caso negativo, o pedido será de *delete*.

* + 1. *Restore*

No caso em que a string lida tem como segundo parâmetro o caracter 'R', é então iniciado o processo de *restore*.

É realizada a descompressão do ficheiro e em seguida restaurado para a diretoria do cliente através do comando:

gunzip < /home/<user>/.backup/metadata/"digestFicheiro" > ./"nomeFicheiro

Encerra-se (*kill*) o processo (*pid*) e é enviado o sinal “30” para o cliente, que corresponde à informação de que o restore foi executado com sucesso:

kill(pid,30)

* + 1. *Delete*

No caso em que a string lida tem como segundo parâmetro o caracter 'D', é então iniciado o processo de *delete*.

Neste caso a ligação na diretoria metadata/ será removida e, caso seja a única ligação para o ficheiro com o nome do digest do ficheiro, a cópia de segurança também será apagada. Assim, é necessário calcular inicialmente quantas ligações existem para a cópia (digest.gz) em .backup/data/. Este cálculo é feito através do comando:

ls -l /home/<user>/.backup/metadata | grep "digest".gz | wc -l

e o número de ligações é guardado na variável int aux.

No caso em que existe apenas uma ligação para a cópia de segurança (aux==1), a cópia é removida através do comando:

rm /home/<user>/.backup/data/"digestFicheiro".gz

Independentemente do valor de aux, em seguida procede-se à remoção do ficheiro ligação na diretoria .backup/metadata/ através do comando:

rm /home/<user>/.backup/metadata/"nome"

Por fim encerra-se (*kill*) o processo (*pid*) e é enviado o sinal “30” para o cliente, que corresponde à informação de que o delete foi executado com sucesso:

kill(pid,30)

* 1. Makefile

Apresenta-se em seguida o código da makefile:

SRCS = $(wildcard \*.c)

CFLAGS= -Wall -pedantic

PROGS = $(patsubst %.c,%,$(SRCS))

all: $(PROGS)

%: %.c

$(CC) $(CFLAGS) -o $@ $<

* 1. Comentários

Tanto no server como no cliente está implementada a função readln, desenvolvida nas sessões práticas, para ler uma linha de um file descriptor.

Foi também criado um script – clean.sh - com a funcionalidade de limpar os diretórios de *backup*.

Outros. Melhorias…..

* 1. Resultados

Uma das características desejáveis do sistema desenvolvido era a eficiência respeitante ao espaço dedicado para o *backup*. Este ponto foi cumprido pois, antes de os ficheiros serem salvaguardados, estes são comprimidos com recurso ao programa gzip. Não menos importante, quando o utilizador solicita o *backup* de um ficheiro que já tenha sido alvo de *backup* com um nome diferente, é criado uma ligação para o *backup* já existente não havendo por isso cópias de ficheiros duplicados.

Cada vez que um ficheiro é apagado, é executada uma remoção dessa informação do *backup*, ou seja, o sistema elimina da diretoria metadata/. a ligação desse ficheiro e faz-se a verificação se existem outras ligações ativas para a cópia correspondente. Caso haja outras ligações a cópia é mantida em data/., caso contrário é eliminada. Deste modo nunca existem ligações sem cópias associadas ou vice-versa. Por isto não faz sentido a implementação de um comando gc().

1. Conclusão

Como conclusão salientamos os seguintes aspetos:

- o sistema funciona como previsto apresentando funcionalidades de backup, restore e delete de ficheiros.

- toda a informação é guardada no diretório .backup

- a privacidade é garantida pois o cliente não tem acesso direto à informação salvaguardada

- o cliente e servidor comunicam através de um pipe e a informação que nele circula está perfeitamente definida (3 strings e 3 sinais)

- a implementação é eficiente pois não existem cópias duplicadas nem cópias que não tenham ligações em na diretoria metadata/.

Pelo referido, consideramos que no geral os objetivos do trabalho foram cumpridos.