

Universidade do Minho

Sistemas Operativos

MIEI - 2º ANO - 2º SEMESTRE

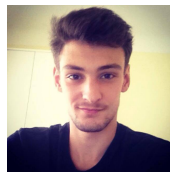
UNIVERSIDADE DO MINHO

BACKUP EFICIENTE

GRUPO



Dinis Peixoto
A75353



Ricardo Pereira
A74185



Marcelo Lima
A75210

16 de Novembro de 2017

Conteúdo

1	Introdução	2
2	Instalação/Desinstalação do programa	3
2.1	Instalação	3
2.2	Desinstalação	3
3	Makefile	4
3.1	Comandos disponíveis	4
4	Funcionalidades do programa	6
4.1	Backup	6
4.1.1	Estrutura do comando	6
4.1.2	Implementação	6
4.1.3	Exemplo de execução	6
4.2	Restore	7
4.2.1	Estrutura do comando	7
4.2.2	Implementação	7
4.2.3	Exemplo de execução	7
4.3	Delete	7
4.3.1	Estrutura do comando	7
4.3.2	Implementação	7
4.3.3	Exemplo de execução	7
4.4	GC	8
4.4.1	Estrutura do comando	8
4.4.2	Implementação	8
4.4.3	Exemplo de execução	8
5	Comunicação Servidor/Cliente	9
5.1	Servidor	9
5.2	Mensagem	9
5.3	Sinais	10
5.4	Cliente	11
6	Conclusão	12

1. *Introdução*

Este projeto foi nos solicitado pelos docentes da UC *Sistemas Operativos* e propõem a realização de um programa de fácil utilização e capaz de realizar um backup eficiente de qualquer ficheiro.

Tal como o professor docente refere no enunciado esta aplicação poderia ser feita com conhecimentos do primeiro ano do curso, não estaríamos no entanto a aplicar a matéria lecionada nesta unidade curricular, nem a manter níveis de eficiência e de privacidade de ficheiros.

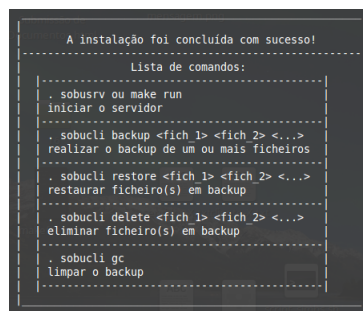
Todavia foram estes requisitos propostos que em conjunto com o limitado prazo de entrega aumentaram a dificuldade do trabalho.

2. Instalação/Desinstalação do programa

2.1 Instalação

Com o objectivo de melhorar e simplificar a utilização do nosso programa, criamos um script de instalação responsável por:

- adicionar os comandos à diretoria: `/home/usr/bin/`
- criar todas as diretorias necessárias: `/home/usr/.Backup/data` e `/home/usr/.Backup/metadata`
- fazer a listagem de comandos ao utilizador:

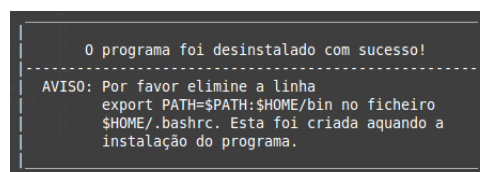


```
A instalação foi concluída com sucesso!
-----
Lista de comandos:
-----
. sobusrv ou make run
iniciar o servidor
-----
. sobucli backup <fich 1> <fich 2> <...>
realizar o backup de um ou mais ficheiros
-----
. sobucli restore <fich 1> <fich 2> <...>
restaurar ficheiro(s) em backup
-----
. sobucli delete <fich 1> <fich 2> <...>
eliminar ficheiro(s) em backup
-----
. sobucli gc
limpar o backup
```

2.2 Desinstalação

Criamos também um script de desinstalação para o caso do utilizador não desejar utilizar mais o programa, este é responsável por:

- remover os comandos da diretoria: `/home/usr/bin/`
- remover todas as diretorias criadas na instalação: `/home/usr/.Backup/data` e `/home/usr/.Backup/metadata`
- alertar o utilizador da pequena alteração feita durante a instalação no `home/usr/.bashrc`:



```
O programa foi desinstalado com sucesso!
-----
AVISO: Por favor elimine a linha
export PATH=$PATH:$HOME/bin no ficheiro
$HOME/.bashrc. Esta foi criada aquando a
instalação do programa.
```

3. *Makefile*

```
CC = gcc

all: servidor cliente

servidor: servidor.c info.c
    $(CC) servidor.c info.c $(CFLAGS) -o sobusrv

cliente: cliente.c info.c
    $(CC) cliente.c info.c $(CFLAGS) -o sobuccli

compile:
    $(CC) servidor.c info.c $(CFLAGS) -o sobusrv
    $(CC) cliente.c info.c $(CFLAGS) -o sobuccli

.PHONY: run
run:
    $(CC) servidor.c info.c $(CFLAGS) -o sobusrv
    $(CC) cliente.c info.c $(CFLAGS) -o sobuccli
    ./sobusrv

.PHONY: install
install: all
    chmod a+x install.sh
    ./install.sh

.PHONY: uninstall
uninstall:
    chmod a+x uninstall.sh
    ./uninstall.sh

.PHONY: clean
clean:
    rm -f sobusrv
    rm -f sobuccli

.PHONY: exit
exit:
    pkill -f sobusrv
```

A Makefile permite-nos compilar todo o nosso programa para que seja possível executá-lo.

3.1 Comandos disponíveis

- **make ou make compile**

Comando default do Makefile para compilar o nosso programa.

- **make run**

Comando para compilar e de seguida iniciar o servidor.

- **make install**

Comando que faz a instalação, chamando o respectivo script.

- **make uninstall**

Comando que desinstala o programa, chamando o respectivo script.

- **make clean**

Comando para remover os executáveis, sobusrv e sobucli, criados no acto de compilação.

- **make exit**

Comando para terminar o processo do servidor.

4. Funcionalidades do programa

4.1 Backup

4.1.1 Estrutura do comando

sobucli backup <nomes_dos_ficheiros>

4.1.2 Implementação

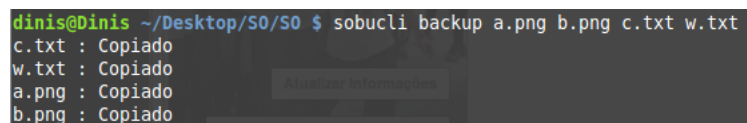
Esta parece uma função muito simples mas é a base de praticamente todo o trabalho. Quando um utilizador executa esta função com o comando acima referido, o cliente percorre o ficheiro ou ficheiros referidos no comando, guardando-o em blocos de **4K bytes** numa estrutura **info** com vários parâmetros que são também preenchidos.

O servidor recebe estes blocos até o ficheiro estar completamente transferido, quando isto acontece o servidor é capaz de o reconhecer através do parâmetro **info->fim**.

Estando o ficheiro completamente transferido, utilizamos o programa *shasum* para gerar um digest e atribuir ao respectivo ficheiro, que será posteriormente comprimido usando *gzip*, este ficheiro ficará na directoria */home/usr/.Backup/data*.

Existirá no entanto um ficheiro com o nome original na directoria */home/usr/.Backup/metadata* linkado simbólicamente ao ficheiro anterior em *data*, isto permitirá reconhecer o ficheiro quando for necessário fazer **restore** do mesmo.

4.1.3 Exemplo de execução



```
dinis@Dinis ~/Desktop/S0/S0 $ sobucli backup a.png b.png c.txt w.txt
c.txt : Copiado
w.txt : Copiado
a.png : Copiado
b.png : Copiado
```

4.2 Restore

4.2.1 Estrutura do comando

sobucli restore <nomes_dos_ficheiros >

4.2.2 Implementação

Esta funcionalidade é essencial, sem esta não fazia qualquer sentido existir a função **Backup**. Esta função começa por associar os nomes de ficheiros descritos no comando na directoria *metadata* e ver onde está o digest destes em *data*, nesta mesma directoria é feita uma cópia do ficheiro e só depois este é descomprimido usando *gunzip*.

De seguida o ficheiro é transferido pelo pipe para o cliente, em blocos de **4K bytes** numa estrutura **info**, tal como no comando **Backup**.

Para finalizar é feito *unlink* do ficheiro cópia em *data* e feita a remoção do mesmo.

4.2.3 Exemplo de execução

```
dinis@Dinis ~/Desktop/S0/S0 $ sobucli restore a.png b.png c.txt w.txt
c.txt : Recuperado.
w.txt : Recuperado.
a.png : Recuperado.
b.png : Recuperado.
```

4.3 Delete

4.3.1 Estrutura do comando

sobucli delete <nomes_dos_ficheiros >

4.3.2 Implementação

Esta é uma das funcionalidades opcionais propostas pelo docente da UC, responsável por eliminar os ficheiros que um utilizador tem em *backup*, já que quando é feito o restore de um ficheiro este continuará em backup.

O **delete** faz uma simples tarefa que é fazer o *unlink* do(s) ficheiro(s) mencionado(s) no comando na directoria *metadata*, eliminando-o assim desta directoria.

4.3.3 Exemplo de execução

```
dinis@Dinis ~/Desktop/S0/S0 $ sobucli delete a.png b.png c.txt w.txt
c.txt : Apagado
w.txt : Apagado
b.png : Apagado
a.png : Apagado
```


4.4 GC

4.4.1 Estrutura do comando

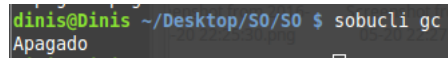
sobucli gc

4.4.2 Implementação

Esta é a segunda funcionalidade opcional proposta, esta tem a função de limpar o *backup*, isto é eliminar todos os ficheiros na directoria *data* que já não se encontram linkados a nenhum ficheiro em *metadata*, isto é, depois de se utilizar o comando **delete**.

A implementação deste comando é feita percorrendo todos os ficheiros na directoria *data* e ir verificando para cada um a existência ou não de um ficheiro linkado em *metadata*, quando não acontecer apaga o ficheiro em *data*.

4.4.3 Exemplo de execução



```
dinis@dinis ~/Desktop/S0/S0 $ sobucli gc
Apagado
```

5. Comunicação Servidor/Cliente

A comunicação entre o servidor e o cliente é realizada por pipes. Na diretoria `/home/usr/.Backup` encontra-se o pipe principal, que estabelece a comunicação entre o servidor e o cliente, e é por onde este último envia comandos ao servidor. Este pipe recebe uma mensagem com uma estrutura muito organizada, que contém o pid do processo que se está a ligar ao servidor, o nome do ficheiro, o comando a realizar e a localização do pipe, esta última não se refere à do pipe principal mas sim a um pipe secundário, que liga o servidor ao cliente e que é responsável pela transferência de informação (ficheiros). Desta forma é possível que cada ficheiro tenha um pipe único para ser transferido, permitindo que corram mais processos do cliente em simultâneo.

Os pipes secundários transferem a informação de forma bem estruturada e organizada, explicaremos como mais adiante neste capítulo.

5.1 Servidor

O servidor quando é iniciado fica a correr num processo filho, ficando assim em *background*. Desta forma, após ligar o servidor é possível que o utilizador se mantenha na mesma *shell* não tendo que executar uma nova para poder executar comandos com o **sobucli**. Além disto o servidor fica à espera de receber comunicações do cliente, esta é uma espera passiva de forma a que o CPU não esteja sobrecarregado. É também importante referir que para o servidor não ser sobrecarregado só aceita até 5 comandos em simultâneo.

5.2 Mensagem

```
#define BUFFER_SIZE 128
#define BLOCK_FILE_SIZE 4096
#define COMMAND_SIZE 16
#define CODE_SIZE 256
#define FILE_NAME_SIZE 512

typedef struct info {
    int pidProcesso;
    int fim;
    int tamanho;
    char Codigo[CODE_SIZE];
    char NomeFicheiro[FILE_NAME_SIZE];
    char comando[COMMAND_SIZE];
    char Ficheiro[BLOCK_FILE_SIZE];
}*INFO;
```

De cada vez que o cliente comunica com o servidor a mensagem é enviada com a forma da nossa estrutura **INFO*, sendo que todos os parâmetros são preenchidos pelo remetente, e interpretados pelo destinatário, isto permite que a comunicação cliente/servidor seja mais controlada, conseguindo assim menos *bugs* no nosso programa.

Parâmetros da estrutura:

- **pidProcesso**

Para haver conhecimento do processo utilizado

- **fim**

Informar se o ficheiro já está completamente transferido.

- **tamanho**

Número de bytes que a mensagem está a transferir.

- **Codigo[CODE_SIZE]**

Digest gerado pelo *sha1sum* do ficheiro.

- **NomeFicheiro[FILE_NAME_SIZE]**

Nome do ficheiro que está a ser transferido.

- **comando[COMMAND_SIZE]**

Nome do comando que foi utilizado.

- **Ficheiro[BLOCK_FILE_SIZE]**

Bloco de, no máximo, **4k bytes**, que é transferido.

5.3 Sinais

Os sinais oferecem uma ferramenta de comunicação entre os processos, nomeadamente entre o servidor e o cliente, extremamente útil para o desenvolvimento deste programa.

Quando é terminada a execução de um comando, o processo servidor envia um sinal ao processo cliente informando se esta foi concluída com sucesso ou não. Em caso positivo, o envio do sinal faz com que seja imprimido na *shell*:

- . *copiado* no caso do **backup**;
- . *recuperado* no caso do **restore**;
- . *apagado* no caso do **delete** e **gc**.

No entanto, caso tenha ocorrido um erro, ou em casos exceção são utilizados diferentes sinais para cada ocorrência, dentro dos quais *SIGUSR1*, *SIGUSR2*, *SIGINT*, etc.

5.4 Cliente

O Cliente é um utilizador do programa, logo do servidor. Para cada cliente que executa um comando, são criados processos conforme o número de ficheiros mencionados no comando do cliente. Ficheiros estes que serão tratados em simultâneo pelos diversos processos em paralelo do servidor. O cliente espera no entanto até ao último ficheiro estar processado para terminar o seu processo.

6. *Conclusão*

Este trabalho gerou muitas dificuldades para o grupo, apesar de ser um trabalho que exigia a matéria abordada na UC, exigiu também que usássemos o nosso espírito de autonomia procurando assim diferentes soluções para as diversas barreiras encontradas durante a realização do mesmo.

Dentro das dificuldades com que nos encontramos, a organizada comunicação entre cliente/servidor e a aplicação da possibilidade de concorrência de ficheiros, em conciliação com a preocupação do aproximar da data limite foram sem margem para dúvida as maiores dificuldades que o grupo teve de encarar e ultrapassar.

Reconhecemos também que com mais tempo teríamos conseguido aplicar mais funcionalidades ao trabalho, como as restantes funcionalidades opcionais propostas pelo professor docente, ou a realização de backup de ficheiros de diretorias diferentes conciliado à capacidade de ter ficheiros com o mesmo nome e conteúdos distintos, que era facilmente resolvido se tivéssemos relacionado os *paths* de cada um dos ficheiros de modo a ficarem distinguíveis.

Ainda assim consideramos que foi feito um bom trabalho, para o qual somos capazes de olhar com orgulho e consideramos que foi um trabalho muito enriquecedor para o nosso perfil como alunos de Engenharia Informática. Foram desenvolvidos conhecimentos importantes com base no funcionamento do sistema operativo que mais usamos (Linux) e percebemos o quanto ainda o podemos explorar.