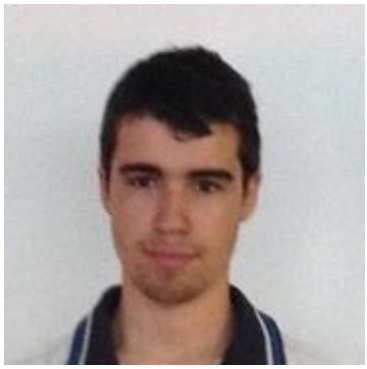
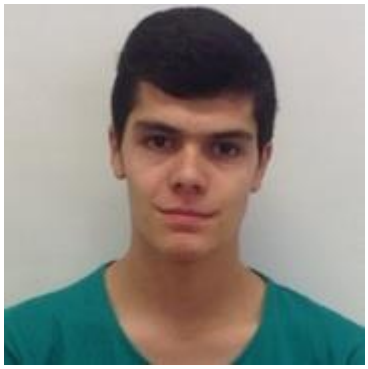
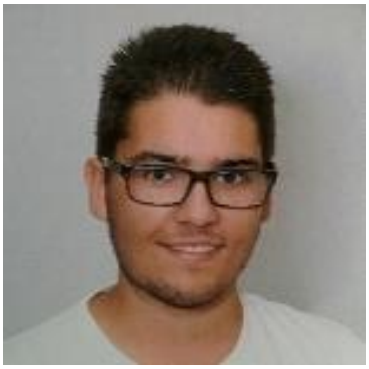
Backup Eficiente

TRABALHO PRÁTICO SISTEMAS OPERATIVOS 2016

Bruno Miguel Salgado Machado	A74941
Fábio Luís Baião da Silva	A75662
João Rui de Sousa Miguel	A74237

Índice

Conteúdo

O que é pedido? **1**

Utilização **2**

Como realizar um backup? 2

Como realizar um restauro? 2

Como apagar um ficheiro do backup? 3

Em Detalhe **4**

Ficheiro sobusrv.c 4

Ficheiro sobucli.c 8

O que é pedido?

Este trabalho pede-nos que seja construído um sistema eficiente de cópias de segurança (Backup), para guardar e recuperar os ficheiros do seu utilizador. Para que tal seja possível é pedido que sejam implementados dois métodos distintos:

Backup: Método responsável por criar cópias dos ficheiros que o utilizador atual pretende guardar.

Restore: Método responsável pelo restauro dos ficheiros/pastas, anteriormente guardados pelo método Backup, para as suas diretorias de origem.

Foram ainda pedidos requisitos adicionais, nomeadamente a eficiência do programa desenvolvido e a privacidade dos dados guardados. Para que este seja o mais eficiente possível é proposto a compressão dos dados lidos, e a eliminação da redundância dentro de cada ficheiro, e duplicação entre ficheiros. No que toca à privacidade dos dados, criou-se uma arquitetura cliente/servidor, para que não seja permitido ao utilizador, o acesso direto à diretoria de Backup.

Utilização

Antes de mais, para que se consigam efetuar backups e restauros, é necessário iniciar o servidor, para isto devemos executar o comando “make sobu” seguido de “./sobusrv”. O comando “make sobu” servirá para criar os ficheiros possíveis de correr no terminal, tanto para o cliente como para o servidor, e ainda criar as pastas que serão usadas para guardar todas as informações de backup. O outro comando, “./sobusrv”, serve para iniciar o servidor, criando um pipe que servirá como base de todo o sistema, e um processo filho (que corre em background), e que lida com os comandos introduzidos pelos processos clientes.

Como realizar um backup?

Para que se consiga realizar um backup é apenas necessário executar o programa sobucli, para tal dever-se-á introduzir no terminal “./sobucli backup [ficheiro para backup] [ficheiro2] [ficheiro3] …”.

Exemplo: $ ./sobucli backup home/User/Documents/lista_compras.txt
Deste modo, estaríamos a efetuar o backup do ficheiro “lista_compras.txt”.

Como realizar um restauro?

Para realizar um restauro de ficheiros é necessário executar o programa sobucli novamente, no entanto, desta vez o primeiro parâmetro do programa é diferente, tendo o utilizador de inserir no terminal: “./sobucli restore [ficheiro a restaurar] [ficheiro2] [ficheiro3] …”.

Exemplo: $ ./sobucli restore lista_compras.txt
Assim, conclui-se o restauro do ficheiro “lista_compras.txt” para a diretoria onde o utilizador esta a executar o programa.

Como apagar um ficheiro do backup?

Existe ainda um método extra, utilizado apenas para apagar ficheiros do backup. Isto é, o utilizador tem a possibilidade de apagar um ficheiro que não seja novamente necessário, ou até que não pretenda fazer restauro. Para isto deverá ser utilizado o comando: "./sobucli delete [nome do ficheiro a apagar] [ficheiro2] [ficheiro3] …".

Exemplo: $ ./sobucli delete lista_compras.txt
Deste modo apagaríamos o ficheiro "lista_compras.txt" do nosso Backup. Esta ferramenta adicional poderá ser útil caso ocorra um engano a fazer um backup (selecionando os ficheiros incorretos).

Em Detalhe

Ficheiro sobusrv.c

char* getDigest (char* path);

A função getDigest é invocada pela função backup, recebe um char* “path” com o caminho do ficheiro que se pretende fazer backup, aplicando o comando execlp ("sha1sum", "sha1sum", path, NULL), e tendo alterado previamente os descritores necessários, podemos executar seguidamente o readln (p[0], result, MAX_BUF), permitindo que a variável result contenha o digest do ficheiro em path.

Retorna um char* contendo o digest do ficheiro selecionado pelo utilizador.

char* separateLast (char* path);

A função separateLast é invocada pelas funções backup e restore, recebe um char* “path” contendo um caminho do ficheiro que o utilizador introduziu. O que esta função faz é retirar o nome do ficheiro que se pretende efetuar o backup ou o restauro, de toda a linha do caminho. Para tal, esta função pesquisa toda a linha do caminho até encontrar um “/”, utiliza-se depois outro ciclo para copiar a string relativa ao ficheiro para uma variável que vai ser retornada. Finalmente faz-se path[i] = 0; file[j] = 0; (equivalente a “\n”), para assinalar o endereço final da string.

Exemplo: Dado um caminho “~/Documents/lista_compras.txt”, a função devolve apenas uma string contendo “lista_compras.txt”. E a string contida na variável path = “~/Documents/”.

void backup (char* path);

A função backup é invocada pela main, após o utilizador dizer ao servidor que pretende efetuar um backup.

Recebe um char* “path” contendo, também, o caminho do ficheiro introduzido pelo utilizador. Esta função começa por invocar a getDigest de modo a obter um digest do ficheiro que o utilizador está a fazer backup. De seguida, abre-se, ou cria-se um ficheiro com o nome igual a digest. Ficheiro este que terá o conteúdo comprimido do ficheiro lido de path.

Para que tal seja possível, o ficheiro existente em path é escrito para um pipe, considerando a prévia abertura/criação deste, e assumindo todos os redireccionamentos

necessários, usando um cat que será executado por um processo filho. Posteriormente, o gzip, também com a ajuda de um processo filho, recebe através desse mesmo pipe a informação contida no ficheiro inicial, sendo assim comprimida e escrita no ficheiro correspondente em "~/.Backup/data".

Concluída a leitura e compressão do ficheiro, cria-se finalmente uma ligação entre o ficheiro criado e um outro ficheiro existente na pasta ".Backup/metadata" com o nome do ficheiro original. Para tal, utiliza-se um processo filho para executar a função execlp ("ln", "ln", "-sf", data, metadata, NULL), fazendo com que a bash execute o programa ln com as flags –sf, sendo o –f indicador de que se já existirem ficheiros de destino, estes serão apagados, e o –s para que o link efetuado seja simbólico, isto é, aponte para o ficheiro em data e não para os dados para o qual este aponta. Este link será então guardado em "~/.Backup/data" com o nome igual ao nome do ficheiro que o utilizador pretendia guardar.

Exemplo: data = "../data/jh324b53m543t"
metadata = "metadata/lista_compras.txt"
ln data metadata

int restore (char* path);

A função restore, tal como a backup, é invocada pela main, após o utilizador inserir o comando para efetuar um restauro de um ficheiro. Recebe um char* "path" com o caminho do ficheiro que se pretende restaurar.

Com a ajuda do linker criado com o backup, linker este existente na pasta metadata, asseguramo-nos que efetua-se a descompressão do ficheiro existente em data para a diretoria pretendida, sendo assim o ficheiro restaurado.

Para que tal aconteça, abre-se o linker associado ao ficheiro que se pretende recuperar cria-se um processo filho e altera-se os seus descritores, processo este que fica responsável por executar a linha de código execlp("gzip", "gzip", "-fd", NULL), que fará com que o conteúdo do ficheiro apontado pelo linker (o respetivo ficheiro em "/.Backup/data"), seja descomprimido para a diretoria pretendida. Para efetuar a descompressão utiliza-se a flag –d (decompress).

int delete (char* arg);

A função delete, recebe um char* "arg", contendo apenas o nome do ficheiro que se pretende remover do sistema de backup.

Esta função apenas utiliza um processo filho para executar um comando em bash, sendo este um "rm". Deste modo, assegura-se que a remoção do ficheiro explícito em "arg" será concluída com sucesso.

Para que tal se suceda, tenta-se abrir o ficheiro na pasta metadata que pretende-mos apagar, se tal ação for possível, significa que o ficheiro existe, podendo o utilizador proceder à sua remoção, a qual será feita por um processo filho, pela execução da função execlp ("rm", "rm", path, NULL), apagando o ficheiro existente em metadata.

void changeDirectory (void);

A função changeDirectory, não recebe nem devolve qualquer tipo de variável. Esta função tem como principal objetivo, criar um caminho que será iniciado com “~/.Backup”, para que toda a execução do programa tenha este caminho para que mais facilmente sejam guardados os ficheiros compressos em metadata e os respetivos linkers em data.

int main();

A função main do ficheiro sobusrv.c tem como objetivo fundamental, interpretar os comandos que vão ser introduzidos pelo cliente. Para que tal aconteça, atualiza-se primeiro cria-se um processo filho, depois atualiza-se a diretoria para a pasta “.Backup” com a função changeDirectory. De seguida, esse filho cria um pipe com o comando mkfifo ("fifo", 0666) na diretoria “/.Backup/”, e executa um ciclo infinito, cujos descritores de leitura são iguais aos de escrita do cliente, de modo a que este, interprete o comando introduzido pelo utilizador, asseguramos assim que o servidor está sempre a ser executado.

O servidor fica assim à espera de ler qualquer coisa dada pelo cliente [readln (fifo, arg, MAX_BUF)]. Após a leitura, preenchemos a variável arg com uma linha de texto, por exemplo, arg = “312 backup Documents/lista_compras.txt”. Guardamos para uma variável o process id do cliente que originou essa linha, com o uso de um tok = strtok(arg, “ “) seguido do comando pid = atoi(tok), sendo tok a string com o numero de PID.

Estipulado o PID, está na altura de saber o que opção o cliente pretende executar, tanto pode ser backup como um restore ou delete. Para tal fazemos novamente tok = strtok(NULL, “ “), e obtemos a segunda string.

Contendo em tok a opção (backup, restore ou delete) que pretendemos executar, usamos um strcmp para verificar o que o utilizador pretende fazer. Depois de identificada a sua opção, caso esta seja um backup, será executado a função backup() e posteriormente será enviado um sinal ao processo do cliente com um “sinal para continuar” para que este possa continuar a sua execução. Na eventualidade de o utilizador querer executar um restore ou um delete, será invocada a função respetiva (ou restore() ou delete()), sendo depois verificado se ocorreram erros na execução da mesma (caso os ficheiros não existissem), originando um código de erro que será enviado para o processo do utilizador com a linha de código kill (pid,

SIGUSR1). Posteriormente será enviado um sinal ao processo do cliente, tal como para a função backup(), para que este possa continuar a sua execução, para tal utiliza-se a função kill (pid, SIGCONT).

Ficheiro sobucli.c

char* getPWD (void);

A função getPWD não recebe nenhuma variável, no entanto, retorna um char* contendo a diretoria onde nos encontramos. Para que tal seja possível, cria-se um processo filho que executa a função execlp ("pwd", "pwd", NULL), apos modificar os descritores de escrita para o pipe, (dup2 (p[1], 1);). A diretoria será lida no processo pai com o uso de n = readln (p[0], path, MAX_BUF), guardando no path a diretoria que o cliente se encontra.

Exemplo: getPWD() = “/home/User/Desktop/Cliente/”

Notar que é adicionado um “/” ao fim da pwd, tal é feito com (path[n] = '/'; path[n+1] = 0;).

void naoexiste ();

A função naoexiste é usada como handler, no caso em que pretendemos restaurar, ou apagar um ficheiro que não existe.

void handler_restore ();

A função handler_restore é usada como handler, no caso em que efetuamos, com sucesso, o restauro de um ficheiro.

void handler_delete ();

A função handler_delete é usada como handler, no caso em que apagamos um ficheiro da nossa diretoria de backups com sucesso.

int main(int argc, char const *argv[]);

A função main do ficheiro sobucli.c tem como principal tarefa, a interpretação direta do que o utilizador introduziu.

No inicio da main, abre o pipe criado pelo servidor, para que haja uma ligação entre ambos os processos, fifo = open ("/.Backup/fifo", O_WRONLY).

Constrói-se ainda uma string que contem o PID deste processo e a opção que o utilizador pretende executar, exemplo, header = “312 backup”.

Caso o cliente pretenda executar um backup, é criado um processo filho que será responsável por verificar se é possível interpretar todos os ficheiros da linha introduzida pelo utilizador, dando os caminhos de todos os ficheiros com o comando execlp ("find", "find", argv[i], "-type", "f", NULL), guardando os seus caminhos para uma string, por exemplo, vamos assumir que o utilizador introduziu a linha "./sobucli backup ~/Documents/lista_compras_*", as iterações do ciclo while correspondente a while (readln (p[0], file, MAX_BUF)), vão colocar no char* file o caminho da todos os ficheiros que correspondam aquele padrão. Imaginemos ainda que existem dois ficheiros, o "lista_compras_comida.txt" e "lista_compras_roupa.txt", a primeira iteração do ciclo poria em file = "~/Documents/lista_compras_comida.txt" e a segunda iteração colocaria file = "~/Documents/lista_compras_roupa.txt", feito isto o ciclo terminaria.

A cada iteração deste ciclo será ainda criado outra string aux com a junção do header anterior e a diretoria de um dos ficheiros a fazer backup, ficamos assim com aux = "312 backup ~/Documents/lista_compras_comida.txt". Enviamos posteriormente esta linha ao servidor com o comando writeln (fifo, aux, strlen (aux) + 1), esperamos pela resposta do mesmo para que consigamos avisar o cliente de que o seu ficheiro foi copiado com sucesso, realizado com pause().

Caso o que o cliente pretenda executar seja um um restauro, será construída uma string aux com o pid, a operação desejada e o primeiro elemento a restaurar, por exemplo, aux = "313 restore ~/Documents/lista_compras.txt". Esta string, tal como no caso em que se pretende fazer um backup, é enviada para o servidor com um writeln (fifo, aux, strlen (aux) + 1), ficando o cliente à espera de um sinal para que possa continuar a execução do programa.

Finalmente, se pretendermos fazer delete de alguma coisa que tenhamos efetuado anteriormente backup, necessitamos da opção para o delete. Caso o cliente pretenda remover vários ficheiros do seu backup deverá inserir, por exemplo, "./sobucli delete lista_compras_comida.txt lista_compras_roupa.txt". Neste caso o ciclo for (i = 2; i < argc; i++), assegurar-se-á que uma string aux será preenchida a cada iteração com o nome do ficheiro que se pretende remover, sendo que na primeira iteração aux = "314 delete lista_compras_comida.txt", seria enviado ao servidor esta string, para que ele efetuasse a sua remoção, tal é feito com o auxilio do comando writeln (fifo, aux, strlen (aux) + 1). O programa fica então à espera de resposta do servidor antes de proceder para a sua última iteração.