Universidade do Minho

Backup Eficiente

Mestrado Integrado em Engenharia Informática

Sistemas Operativos
(2ºSemestre/2015-2016)

Grupo 27

A71015 Beatriz Aarão

A61887 Carlos Pereira

A61799 Rafael Braga

Braga
20 de Maio de 2016

Resumo

O presente relatório serve de apoio ao trabalho prático da unidade curricular de Sistemas Operativos.

Com este trabalho, pretende-se construir um sistema eficiente de cópias de segurança (backup), para salvaguardar ficheiros de um utilizador.

O sistema será composto por dois programas: um programa cliente que servirá para dar comandos ao sistema e um programa servidor que efetuará as operações de cópia e reposição propriamente ditas.

Ao longo deste relatório, serão abordados todos os objetivos propostos no enunciado, incluindo os objetivos de valorização.

Palavras Chave: Backup. Restore. Eficiência. Privacidade. Servidor. Cliente. Pipe. Digest. Data. Metadata. Sinais.

Conteúdo

Capítulo 1

Introdução

1.1 Descrição do Problema

Como já foi referido anteriormente, este trabalho tem como objetivo criar um sistema de cópias de segurança de ficheiros, implementado em C, usando as primitivas do sistema operativo e os programas utilitários Unix indicados. Este sistema deverá possuir duas operações principais, as quais poderão ser invocadas pelo utilizador, sendo estas:

- **Backup:** Responsável pela criação de cópias de segurança dos ficheiros e/ ou pastas indicados pelo utilizador.
- **Restore:** Responsável pelo restauro de ficheiros e/ou pastas, repondo-os no local original.

Este trabalho possui ainda alguns requisitos de eficiência e privacidade, nomeadamente a minimização do espaço em disco ocupado pelo *backup*, utilizando para isso eliminação da redundância e reduplicação dos dados, e a utilização de uma arquitetura cliente/-servidor que impedirá o acesso direto por parte do utilizador à pasta de Backup.

O **Servidor** desenvolvido deverá ser capaz de:

- Permanecer em execução, aguardando pedidos do cliente para operações de cópia e recuperação de ficheiros.
- Guardar todos os dados numa única diretoria, à qual chamamos a raiz do backup. Será usada a diretoria */home/user/.Backup/*.
- Enviar respostas ao cliente, para indicar a conclusão das operações ou erros, usando apenas sinais.

- Nunca executar mais do 5 operações (de cópia ou recuperação) em simultâneo, de forma a não sobrecarregar o sistema.

O **Cliente** desenvolvido deverá ser capaz de:

- Nunca escrever nem ler ficheiros diretamente na raiz do *backup*. Deverá enviar comandos para copiar e recuperar ficheiros ao servidor usando um *FIFO /named pipe*[1] que se encontra na raiz do *backup*.
- Executar concorrentemente várias instâncias do mesmo sem que isso cause problemas.
- Esperar pela confirmação do servidor que cada ficheiro está efetivamente copiado ou recuperado antes de imprimir a mensagem correspondente. Só terminar quando todas as operações solicitadas estiverem concluídas

Os ficheiros deverão ser guardados em duas sub-diretorias da raiz do backup com os nomes data e metadata da seguinte forma:

- O conteúdo do ficheiro deve ser comprimido com o programa gzip e guardado na sub-diretoria /data/ num ficheiro, cujo nome é o *digest*[2] do seu conteúdo não comprimido, calculado pelo programa *sha1sum*.
- Deve ser criado na sub-diretoria metadata/ uma ligação com o nome original do ficheiro para o seu conteúdo comprimido na sub-diretoria data/.

[1] canal de comunicação unidirecional utilizado na comunicação entre processos
[2] Chave gerada a partir do conteúdo de um ficheiro

Capítulo 2

Desenvolvimento

2.1 Script de Instalação

O *script* de instalação deve ser executado em primeiro lugar. A sua execução deve ser acompanhada por todos os ficheiros: *sobusrv.c*, *sobucli.c* e *makefile*. É responsável pela criação de todas as diretorias necessárias para o funcionamento do Servidor e do Cliente. Cria as diretorias *.Backup*, *.Backup/data* e *.Backup/metadata*. No final, executa a makefile de instalação dos programas *sobucli* e *sobusrv*.

2.2 Makefile

A *makefile* compila os ficheiros *sobucli.c* e *sobusrv.c* com as *flags Wall*, *Wconversion*, *pedantic*, *Wextra* e *o*. Também elimina os ficheiros objeto criados durante a compilação. O resultado final são os executáveis *sobucli* e *sobusrv* que correspondem, respetivamente, ao Cliente e Servidor.

2.3 Servidor

O Servidor pode ser considerado como o programa principal do projeto, uma vez que é o primeiro programa a ser executado.É responsável por inicializar o *FIFO* necessário à comunicação entre Servidor e Clientes.

Este é também responsável por executar os comandos que o Cliente envia pelo *FIFO p_request*. Este *FIFO* encontra-se na diretoria raiz do Servidor: */.Backup/*. O servidor é capaz de executar os seguintes comandos:

- *backup* - criação de uma cópia de segurança
- *restore* - recuperação de um ficheiro com cópia de segurança

- *delete* - eliminação da cópia de segurança de um ficheiro
- *gc* - eliminação dos ficheiros na diretoria */.Backup/data/* que não tenham links

Todos os comandos devem receber pelo menos o nome de um ficheiro (ou diretoria), exceto o gc. O Servidor não executa mais do que cinco operações (ou comandos) ao mesmo tempo. Após a execução de um comando, envia ao Cliente desse comando, um sinal a indicar sucesso ou insucesso na execução.

2.3.1 backup

O comando *backup* faz a cópia de segurança de um ficheiro. A função principal deste comando (backup), recebe como argumentos o nome do ficheiro e o ID do processo que enviou o comando. O processo em causa foi criado pelo Cliente enquanto lia os argumentos de *argv[ ][ ]*.

A função backup começa por calcular o *digest* do ficheiro, caso o ficheiro exista, chamando depois duas funções auxiliares: *zipFile* e *createMetadata*.

- *zipFile* - trata da compressão do ficheiro, guardando o resultado na diretoria /.Backup/data/, dando-lhe o nome do digest.
- *createMetadata* - faz a ligação entre o ficheiro comprimido, que se encontra em */.Backup/data/*, e um link que será guardado na diretoria */.Backup/metadata/*. Os nomes de todos os links aqui guardados correspondem ao caminho original do ficheiro. Por exemplo, se fosse feito o backup de um ficheiro de caminho absoluto */home/usr/exemplo.txt*, o nome do link seria *>home>usr>exemplo.txt*.

Optou-se por utilizar esta técnica, de modo a simplificar a solução da função *restore*, já que sabemos desde o início a localização original do ficheiro.

Aquando do término da função principal do backup, ou das suas auxiliares, é enviado o sinal *SIGUSR1* caso seja sucedido e *SIGUSR2* caso contrário. Como temos desde o início o *ID* do processo que mandou executar o backup, já sabemos para onde devemos enviar estes sinais. Depois, o Cliente fica responsável por informar o utilizador o resultado da execução.

2.3.2 restore

O comando *restore* recupera um ficheiro de uma diretoria (caso tenham cópias de segurança) para a sua localização original.

A função principal deste comando (*restore*), recebe o caminho absoluto original do ficheiro ou diretoria a descomprimir, como também o *ID* do processo que enviou o comando. O processo em causa foi criado pelo Cliente enquanto este lia os argumentos de *argv[ ][ ]*.

A função *restore*, acede a todos os links na diretoria */.Backup/metadata/* e compara os seus nomes com o do caminho recebido anteriormente. Caso o *link*, que já contém o caminho original do ficheiro, tenha os primeiros n carateres iguais ao caminho recebido como parâmetro, significa que o *link* tem uma ligação com o ficheiro (ou diretoria) pretendido. Caso isto se verifique, é chamada a função *restoreFile* que efetua a descompressão de ficheiros na diretoria */.Backup/data/*.

Esta função *restoreFile* recorre à função *createFolders* que cria as diretorias do caminho original do ficheiro a ser descomprimido. Só depois de ter as diretorias criadas é que a *restoreFile* irá descomprimir o ficheiro para o lugar original.

A recuperação de mais do que um ficheiro é feita sequencialmente, já que não sabemos desde logo quantos ficheiros iremos recuperar. O Servidor considera esta sequência como uma só operação.

A função *restore* e a função *restoreFile* recebem como parâmetro o ID do processo criado pelo Cliente que mandou ao Servidor executar o comando *restore*. Deste modo, quando estas funções terminam, é enviado um sinal que avisa este processo se o comando foi bem executado ou não. Depois disto, o Cliente fica responsável por informar o utilizador do resultado da execução.

2.3.3 delete

O comando *delete* elimina a cópia de segurança de um ficheiro. Isto é, remove o *link* de um ficheiro da diretoria */.Backup/metadata/*. Desta forma, garante-se a existência de dados (na diretoria */.Backup/data/* que sejam partilhados por mais do que um ficheiro.

A função principal deste comando é *delFile* que recebe o caminho absoluto original do ficheiro a remover e o *ID* do processo que enviou o comando. O processo foi criado pelo Cliente enquanto lia os argumentos de *argv[ ][ ]*.

Os nomes dos *links* em */.Backup/metadata/* representam as localizações originais de ficheiros com cópia de segurança. Sabendo o nome do ficheiro a remover, o Servidor apenas precisa de aplicar a função *unlink* a um *link* com esse nome.

A função *delFile* recebe como parâmetro o *ID* do processo criado pelo Cliente que mandou ao Servidor executar o comando *delete*. Deste modo, quando esta função termina, é enviado um sinal que avisa este processo se o comando foi bem executado ou não. Depois disto, o Cliente fica responsável por informar o utilizador do resultado da execução.

2.3.4 gc

O comando *gc*, é responsável por eliminar todos os ficheiros existentes na sub-diretoria *data/* que não possuam ligação a nenhuma das entradas em *metadata/*. Desta forma, garante-se um uso eficiente de memória.

A função principal deste comando, é a *gc*, a qual recebe como parâmetro o *ID* do processo criado pelo Cliente que mandou ao Servidor executar o comando *gc*. A função *gc*, irá aceder a todos os ficheiros da diretoria *data/*, verificando posteriormente se estes possuem alguma ligação a alguma das entradas em *metadata/*. Para isso, percorrerá todas as entradas de *metadata/* recursivamente, até que seja encontrado um *link* ligado ao ficheiro a ser comparado, ou até que não existam mais entradas.

Esta função, recorre a várias funções auxiliares, entre as quais se encontram as funções *opendir* e *readdir*, responsáveis por aceder às diretorias requeridas. Para além destas duas funções auxiliares, é ainda utilizada a função *existFile*, responsável pela verificação das ligações entre os ficheiros da diretoria *data/* e *metadata/*. Para que esta verificação seja possível, é utilizada como auxiliar à função *existFile*, a função *readlink*, responsável por devolver o valor do link, para que posteriormente seja possível a utilização da *unlink*, para eliminar os ficheiros.

A função *gc* recebe como parâmetro o *ID* do processo criado pelo Cliente que mandou ao Servidor executar o comando *gc*. Deste modo, quando esta função termina, é enviado um sinal que avisa este processo se o comando foi bem executado ou não. Depois disto, o Cliente fica responsável por informar o utilizador do resultado da execução.

2.4 Cliente

O Cliente é responsável por receber comandos e ficheiros (quando necessários) do utilizador, enviar estas informações para o Servidor e ficar à espera de uma resposta do Servidor, de forma a informar o utilizador do sucesso (ou não) da execução do comando. O Cliente comunica com o Servidor através de um *FIFO* de nome *p_request*.

O Cliente deve receber o nome onde executar o comando como uma de duas maneiras, dependendo daquilo que o utilizador pretende:

1. Como caminho a partir da diretoria de onde se executou o ficheiro objeto gerado a partir de *sobucli.c*. Por exemplo,

   - Caso se tenha um ficheiro *a.txt* na diretoria atual, o comando ficaria *./sobucli "comando"a.txt*
   - Caso se tenha uma sub-diretoria *Dir* na diretoria atual, o comando ficaria *./sobucli "comando"Dir*

2. Como caminho absoluto, caso se encontre numa diretoria que não seja sub-diretoria desta. Por exemplo,

   - Caso se tenha um ficheiro *a.txt* no *Desktop*, o comando ficaria *./sobucli "comando"/home/user/Desktop/a.txt*
   - Caso se tenha uma sub-diretoria *Dir* no *Desktop*, o comando ficaria *./sobucli "comando"/home/user/Desktop/Dir*

Antes de escrever no *FIFO*, o Cliente verifica se o comando indicado pelo utilizador é válido. Este comando encontra-se em *argv[1]*.

A função de validação *verifyCommand* aceita as seguintes instruções:

- *backup* - comando que cria a cópia de segurança de um ficheiro. Apenas considera este comando como válido caso seja fornecido pelo menos o nome de um ficheiro (ou diretoria) para executar o comando ($argc > 2$).
- *restore* - comando que recupera um ficheiro (ou diretoria). Apenas considera este comando como válido caso seja fornecido pelo menos o nome de um ficheiro (ou diretoria) para executar o comando ($argc > 2$).
- *delete* - comando que elimina a cópia de segurança de um ficheiro (ou diretoria). Apenas considera este comando como válido caso seja fornecido pelo menos o nome de um ficheiro (ou diretoria) para executar o comando ($argc > 2$).
- *gc* - comando que elimina todos os ficheiros em */.Backup/data/* que não tenham ligações em */.Backup/metadata/*. Apenas considera este comando como válido quando o utilizador não fornece nome de ficheiros ($argc = 2$).

Após a validação do comando, o Cliente começa por abrir o descritor de *p_request* (criado pelo Servidor) para escrita. O Cliente cria um processo para cada ficheiro onde se vai executar o comando. Dentro deste processo, é feita a escrita em *p_request*. O processo envia os seguintes parâmetros, separados por espaços, como uma única linha:

- *ID* do processo criado e que envia o comando e o ficheiro;
- Nome do comando a executar;
- Nome do ficheiro a executar (exceto se o comando for *gc*).

O terminador desta linha é definido por defeito como $\backslash n$. O Servidor já sabe que a linha recebida termina quando encontrar este caractere. Depois disto, é escrito no FIFO uma linha com todas estas informações. Também se definem as respostas que o Cliente deverá apresentar ao utilizador a partir dos sinais enviados pelo Servidor:

- *SIGUSR1* caso o comando seja executado com sucesso.
- *SIGUSR2* caso o comando não seja executado com sucesso.

2.4.1 backup

O comando backup é responsável por fazer a cópia de segurança de ficheiros e de diretorias.

No Cliente, a função principal associada a este comando é a função *backup*, a qual recebe o ficheiro ou diretoria a copiar e o descritor para o *FIFO*. Esta função é responsável por "desconstruir"o local onde se vai aplicar o comando passado pelo cliente.

Caso o cliente passe uma diretoria para ser copiada, a função *backup* será chamada recursivamente sobre essa mesma diretoria, e suas sub-diretorias, até que se chegue aos ficheiros individualizados das mesmas. Se se tratar de um ficheiro, será chamada a função *backupFile*, a qual recebe o nome do ficheiro a copiar bem como o descritor do *FIFO*. Esta função, constrói o comando a inserir no *PIPE* que será enviado ao servidor.

Caso o *backup* do ficheiro seja feito com sucesso, será recebido o sinal *SIGUSR1*, o qual corresponde a imprimir no ecrã que o ficheiro foi copiado.

2.4.2 restore

O comando *restore*, é responsável por recuperar um ficheiro (ou diretoria).

No caso do comando fornecido pelo cliente seja válido, este é enviado ao servidor para que seja efetuado. Posteriormente, o Cliente irá verificar qual o comando enviado. Caso este comando diga respeito a um *restore*, o Cliente irá então imprimir no *standard output* uma mensagem correspondente ao sinal recebido.

Caso o *restore* do ficheiro seja feito com sucesso, será recebido o sinal *SIGUSR1*, o qual corresponde a imprimir no ecrã que o ficheiro foi recuperado. Caso contrário, será recebido o sinal *SIGUSR2*, o qual corresponde a imprimir no ecrã que o ficheiro não foi recuperado..

2.4.3 delete

O comando *delete*, é responsável por eliminar a cópia de segurança de um ficheiro (ou diretoria).

Caso o comando passado pelo cliente seja válido, este é enviado ao servidor para que seja efetuado. Posteriormente, o Cliente irá verificar qual o comando enviado. Caso este comando diga respeito a um *delete*, o Cliente irá então imprimir no *standard output* uma mensagem correspondente ao sinal recebido.

Caso o *delete* do ficheiro seja feito com sucesso, será recebido o sinal *SIGUSR1*, o qual corresponde a imprimir no ecrã que o ficheiro foi apagado. Caso contrário, será recebido o sinal *SIGUSR2*, o qual corresponde a imprimir no ecrã que o ficheiro não foi apagado.

2.4.4 gc

O comando *gc*, é responsável por eliminar todos os ficheiros em */.Backup/data/* que não tenham ligações em */.Backup/metadata/*.

Contrariamente aos outros comandos, o comando *gc* não recebe como argumento nenhum ficheiro.

Caso o comando passado pelo Cliente seja válido, este é enviado ao servidor para que seja efetuado. Posteriormente, o Cliente irá verificar qual o comando enviado. Caso este comando diga respeito a um *gc*, o Cliente irá então imprimir no *Standart Output* uma mensagem de sucesso, indicando que este foi feito com sucesso.

2.5 Eficiência e Privacidade

No enunciado do projeto, era pedido que tivéssemos em atenção a eficiência e privacidade do mesmo, nomeadamente minimização do espaço em disco ocupado pelos ficheiros, evitando a duplicação dos dados e impedindo o acesso direto do utilizador à pasta/dispositivo de backup.

Para garantir a minimização de ficheiros em disco, foi utilizado o *digest* de um ficheiro como nome do mesmo, aquando da sua compressão. Desta forma, conseguimos garantir que se estes tiverem todos nomes diferentes, o conteúdo do mesmo não é igual. Outra otimização consistiu em usar várias *system calls* provenientes da linguagem C para reduzir o número de chamadas ao sistema operativo. Exemplos destas funções são: *mkdir, unlink, readlink, ...* Estas funções evitam múltiplos *forks* e *execs* que são bastante ineficientes. Outra estratégia a nível de eficiência consistiu em usar as funções *opendir()* e *readdir()* como substitutas ao comando *find*. Embora o resultado do comando *find* para listar diretorias recursivamente seja muito mais fácil a nível de utilização, envolve várias chamadas ao sistema operativo. Ao recorremos às funções do *C* evita-se o uso de um *fork()* e um de *exec* (chamadas ao sistema operativo), à criação e direcionamento do resultado do comando *find* para um ficheiro (mais chamadas ao sistema operativo) e leitura de cada linha desse ficheiro (várias chamadas ao sistema operativo conforme o número de ficheiros presentes numa diretoria).

Para evitar sobrecarregar o sistema, o número de instruções que o servidor executa ao mesmo tempo é, no máximo, cinco. A implementação consistiu em guardar-se num contador global o número de processos filhos que estão a ser executados. Sempre que um filho é criado, a variável é incrementada. Um filho só é criado se a variável for menor que cinco. Quando um processo filho termina, o processo pai recebe o sinal *SIGCHLD*. Sempre que o pai recebe esse sinal, a variável correspondente ao número de processos filhos é decrementada. Com a mesma conta (mas diferentes terminais), foram realizados testes para verificar a integridade deste sistema.

No que diz respeito à privacidade do programa, isto é conseguido através do impedimento do acesso direto por parte do Cliente à pasta *.Backup/* do Servidor. Para garantir que isto é respeitado, o Cliente apenas envia os comandos pretendidos através de um *PIPE* para o Servidor, sendo este responsável pela sua execução. Devido à

estratégia de guardar o caminho completo dos ficheiros comprimidos na pasta relativa ao servidor, clientes com contas diferentes apenas conseguem fazer o *restore* dos seus ficheiros (embora esta funcionalidade não tenha sido completamente implementada).

Capítulo 3

Conclusão

Neste trabalho, o principal objetivo era a utilização da linguagem de programação *C*, para a criação de um programa de *Backup* que fosse eficiente.

Durante o processo de codificação do programa, pretendeu-se sempre como objetivo final um programa que fizesse sentido nas possibilidades que oferecesse ao utilizador. Um exemplo disto é o comando *restore*. Ao receber pelo menos um ficheiro válido, este comando recria (caso já não existam) as diretorias do caminho absoluto do ficheiro original. Outro exemplo deste objetivo e continuando no comando *restore*, é a possibilidade de recuperar uma diretoria inteira (e sua sub-diretorias). Em vez do utilizador ser obrigado a indicar cada um dos ficheiros que existiam na diretoria original, apenas tem que indicar o nome da diretoria.

Outro objetivo foi tentar utilizar o número mínimo de chamadas ao sistema operativo. Para isso, utilizaram-se *system calls* do próprio *C*.

A maior dificuldade deste trabalho foi a comunicação de vários clientes (de diferentes contas) ao mesmo servidor. Embora a nossa implementação permita a execução de comandos provenientes de vários clientes ao mesmo tempo, sem comprometer a integridade do sistema, e impeça o acesso aos ficheiros do servidor por parte dos clientes, surgiram alguns problemas. Nomeadamente, a alteração de permissões de utilizadores (que não sejam o Servidor) e, no fim da execução do programa, a reposição dessas permissões para o seu estado original. Por exemplo, no comando *restore*, seria necessário alterar as permissões do Cliente de maneira que o Servidor conseguisse escrever nele e no fim alterar essas permissões para o seu estado original.