Universidade do Minho
Departamento de Informática

Sistemas Operativos

Mestrado Integrado em Engenharia Informática
2015/2016



Grupo 14

Bruno Pereira - 75135
Diogo Silva - 76407
Maria Ana de Brito - 73580

Introdução

Este projeto consistiu no desenvolvimento de um sistema eficiente de cópias de segurança (salvaguardar ficheiros de vários utilizadores). Para tal ser possível, este software requere dois comandos fundamentais: “backup” e “restore”. Salvaguardar e restaurar ficheiros, respetivamente. É pretendido que se mantenha privacidade entre o cliente e o servidor e, portanto, o servidor é a única entidade que efetua trabalho sobre os comandos inseridos do(s) utilizador(es). Além dos dois comandos suprarreferidos, para uma melhor gestão do conteúdo de “~/.Backup”, é permitido ao utilizador invocar dois comandos adicionais: “delete” e “gc”. Deste modo, “delete” encarrega-se de eliminar um ficheiro de um determinado utilizador de *metadata* e, se possível, de *data*. De uma forma muito semelhante, “gc” encarrega-se de eliminar todos os “backups” de data que já não sejam possíveis de recuperar, isto é, já não estão a ser utilizados. Ainda, de forma a sinalizar o sucesso (ou insucesso) da operação o servidor envia sinais ao cliente.

Este programa utiliza funções ao nível do sistema operativo, o que requere cuidados e conhecimentos adicionais na sua manipulação. Assim, o objetivo deste projeto é colocar em prática os conhecimentos adquiridos ao longo do semestre a Sistemas Operativos.

Programa Cliente

O cliente apresenta um sistema de “login” e registo de utilizadores. Para tal ser possível, os dados referentes a esses são guardados em memória na raiz do “backup”. Dados esses que apenas são removidos aquando a desinstalação do programa, ou seja, um utilizador pode, a qualquer momento, correr o programa cliente noutra diretoria sem que isso implique não conseguir restaurar os seus ficheiros/pastas anteriormente submetidos.

Assim, os ficheiros/pastas salvaguardados estão ligados ao nome do utilizador permitindo alguma flexibilidade ao próprio. Apesar de termos produzido um sistema de controlo de utilizadores muito simplificado, temos várias verificações para evitar que erros aconteçam: não podem existir utilizadores com o nome igual, não são permitidos espaços no nome de utilizador assim como na palavra-passe, os utilizadores não podem ser removidos através de qualquer comando, o máximo de caracteres para o nome de utilizador assim como para a password é de 32 caracteres.

Além da validação dos utilizadores, o cliente envia comandos ao servidor e recebe sinais provenientes do mesmo. Se, por exemplo, o utilizador introduzir o comando: “backup *.docx”, o cliente encarrega-se de fazer a busca dos ficheiros na sua diretoria e enviar o comando de cada ficheiro individual para o servidor. Para cada comando enviado, o cliente espera pela resposta do servidor antes de enviar o próximo.

Desenhamos o programa desta forma para transmitir ao utilizador algum *feedback* sobre quais os ficheiros que estão a ser bem guardados. Se, porventura, tivéssemos preferido que fosse o servidor a efetuar a busca dos ficheiros na diretoria do cliente e apenas depois os processar, o cliente não teria qualquer controlo em relação aos ficheiros que foram bem salvaguardados e aqueles que não foram.

Assim, o utilizador saberá se todos os ficheiros que pediu para serem salvaguardados foram bem processados ou se, porventura, ocorreu algum erro num deles. O utilizador encarrega-se da tomada da decisão após a resposta do servidor. Toda a comunicação do cliente para o servidor é efetuada através de um *pipe* com nome na mesma diretoria do servidor.

Um comando para o servidor apresenta a seguinte sintaxe: a primeira *string* é o nome do utilizador, a próxima, o comando, depois o ficheiro, a seguir temos o caminho do cliente e por fim, o *process id*.

A nossa estratégia limitou-nos a capacidade de concorrência de operações para um cliente singular. Como o cliente mantém-se à espera da resposta do servidor para enviar o próximo pedido implica que o envio de informação para o servidor seja sequencial.

Por outro lado, se tivermos vários clientes a correr ao mesmo tempo e a enviar comandos para o servidor, estes serão processados concorrentemente.

Algumas funções utilizadas no cliente:

void clearEcran(void);

Apenas executa um comando “clear” para limpar o terminal. É utilizada essencialmente no interpretador de comandos do cliente.

ssize_t readln(int fd, char *buffer, size_t nbyte);

Função que lê uma linha de caracteres de um ficheiro em memória. Não existe realocação de memória do “buffer”. É lido no máximo “nbyte” caracteres.

int existsUser(char *user);

Verifica se um dado utilizador consta no ficheiro de credenciais. Isto é, o conjunto de utilizadores acompanhados pelas respetivas palavras passe. Retorna 1 caso “user” já esteja registado, retorna 0 caso contrário.

char *getPath(void);

Determina qual é o caminho em que se encontra o atual utilizador. Utiliza o comando “pwd” para esse efeito. Retorna o resultado de executar “pwd”.

void sucess(int s);

“Handler” utilizado em “signal(SIGUSR1, success);” de forma a marcar a intenção de sucesso tal como sugere.

void non_sucess(int s);

“Handler” utilizado em “signal(SIGUSR2, non_success);” de forma a marcar a intenção de insucesso tal como sugere.

void access_server(char *client_name);

Quando um utilizador se “loga” com sucesso é-lhe permitido, então, enviar comandos ao servidor. Esta função serve esse efeito.

Interpretador de comandos do cliente:

Logo após o programa cliente arrancar, são apresentadas três opções ao utilizador: “logar-se”, registar-se ou sair do programa.

Caso seja um novo utilizador deve previamente registar-se. Não será possível registar-se com o nome de um utilizador que já exista, pelo que deve inserir um novo. Além do limite de 32 caracteres também não lhe é permitido um nome de utilizador com espaços, análogo para a palavra-passe.

Assim que o utilizador se “logar”, poderá introduzir um dos seguintes comandos (além de '0' para sair do menu):

Comando “backup”:

Ao utilizador é-lhe permitido inserir o comando acompanhado do nome do ficheiro, pasta, ou conjunto de ficheiros (através da utilização do caractere '*'). Se por exemplo o utilizador inserir “backup *.txt” então o primeiro passo passa por verificar se o ficheiro introduzido se trata de uma pasta, em caso afirmativo utilizará um método diferente.

Neste caso ao utilizar “test -d” sobre o ficheiro daria falso. Como neste caso não é uma pasta, devemos a seguir listar todos os ficheiros com essa extensão através do comando da *bash* “ls -1”. O output de do comando anteriormente referido é redirecionado para um ficheiro em memória denominado de “.temp”. De seguida, o processo pai encarrega-se de ler linha a linha, utilizando *readln*, todos os ficheiros de “.temp”. Após ter na sua posse um ficheiro é enviada uma “string” ao servidor através de um fifo. Essa string, além do comando e do ficheiro em questão, terá ainda o nome do utilizador assim como o caminho do mesmo e o pid do processo pai.

Neste ponto, antes de enviar o “próximo ficheiro” para o servidor, o cliente aguarda algum “feedback” desse através de um sinal. “SIGUSR1” assinala que o ficheiro foi salvaguardado com sucesso enquanto que “SIGUSR2” assinala o oposto.

Assim que o cliente obtém a resposta do servidor imediatamente comunica ao utilizador a tradução do sinal numa mensagem apresentada no ecrã e avança para o próximo ficheiro. Repete o conjunto de operações até não existir mais nenhum ficheiro para ser lido de “.temp”.

Além do “input” exemplo referido anteriormente, ao utilizador é-lhe permitido também inserir um comando “backup file.txt” de forma individual. O processo pelo qual este ficheiro individual passará é exatamente o mesmo quando o utilizador introduzir “backup *.txt”. Isto é possível porque quando efectuamos “ls -1” apenas será encontrado um ficheiro, “file.txt” e prosseguirá a efetuar trabalho da mesma forma.

Por fim, é ainda permitido ao utilizador pedir para se salvaguardarem pastas. Quando o teste “test -d <arg>” retornar verdade, então, é enviada uma “string” ao utilizador com o nome do comando diferente. Neste caso, “backup_f”, para indicar que queremos guardar uma pasta e não o ficheiro - o servidor encarrega-se de utilizar o método apropriado para o efeito. À semelhança para ficheiros, o cliente aguarda algum “feedback” do servidor antes de prosseguir.

Comando “restore”:

Quando o utilizador introduz “restore” acompanhado de algum ficheiro/pasta o cliente envia apenas uma “string” ao servidor com esses parâmetros. É o próprio servidor que se encarrega de testar se se trata de um ficheiro ou pasta. Após o envio da “string”, o cliente mantém-se à espera da resposta do servidor. Analogamente aos outros comandos, “SIGUSR1” marca uma resposta de sucesso e “SIGUSR2” o oposto. Os ficheiros restaurados são enviados para a diretoria onde se encontra o cliente independentemente de onde o utilizador se encontre ao fazer “login”.

Consideramos que esta forma de restaurar ficheiros/pastas é mais cómoda a um possível utilizador além de ser mais simples de implementar (e menos propensa a “bugs”).

Comando “delete”:

O cliente apenas encarrega-se de enviar o comando acompanhado do ficheiro/pasta introduzido pelo utilizador ao servidor. É o servidor que se encarrega de determinar se se trata efetivamente de um ficheiro ou de uma pasta e utilizar o método mais adequado. Denote-se que as entradas de “metadata” são sempre eliminadas, o que poderá não ser eliminado na eventualidade de ser partilhado é o conteúdo em “data”.

Comando “gc”:

Este comando é o único que não necessita de argumentos. Para contornar a situação sem alterar o código previamente definido assim como a sua estratégia, ao enviar ao servidor a “string” com o conjunto de parâmetros, adicionamos. logo após o comando, um “ficheiro” para ser ignorado após o processamento da “string” por parte do servidor. Neste caso, decidimos utilizar o caracter '.' para demarcar um ficheiro “nulo”.

Programa Servidor

Como sugerido, impusemos um limite de cinco operações de qualquer tipo ao servidor. Quando o limite é atingido, a próxima operação fica à espera da sua vez. A nossa estratégia foi utilizar uma variável global inicializada a 0 que representa o número de operações a serem efetuadas em simultâneo, atingindo o limite, a próxima operação fica em espera até receber um “SIGCHLD” indicando que já pode prosseguir e a variável global anteriormente referida é decrementada.

Comando backup (para ficheiros)

O servidor trabalha com ficheiros individuais provenientes do cliente no comando “backup”. A eficiência deste comando foi assegurada através da utilização de links simbólicos. Após o recebimento da informação, começa-se por calcular o *digest* do ficheiro com “sha1sum”. Este é guardado num ficheiro “tmp” para posteriormente ser utilizado.

A seguir, comprimimos o ficheiro com “gzip” que é posteriormente movido e renomeado com “mv” para /home/*user*/.Backup/data.

Já em ~/.Backup/data, o nome do ficheiro comprimido é alterado para o *digest* do ficheiro original. Nesta etapa, já temos o ficheiro comprimido assegurado, falta-nos, portanto, criar o seu link em *metadata* na pasta do utilizador correspondente.

Portanto, agora, com “ln” criamos um link simbólico em ~/.Backup/metadata/*utilizador* a apontar para o ficheiro comprimido. O link apresentará o nome do ficheiro original, isto é, assumindo que é pedido para o servidor efetuar um “backup” de “file.txt”, então em ~/.Backup/data teremos o mesmo ficheiro comprimido cujo nome é o *digest* do conteúdo não comprimido e em ~/.Backup/metadata/*utilizador* teremos um link chamado “file.txt”.

Se tudo correu conforme o planeado, a função “backup” chegará ao fim e retornará 0 como sinal de sucesso. Já fora da função, se o valor de retorno for 0 então é enviado ao servidor um “SIGUSR1” (correu tudo bem), caso contrário será enviado um “SIGUSR2”.

Comando backup (para pastas):

Quando o servidor recebe o comando “backup_f” vai proceder com um método ligeiramente diferente para salvaguardar a pasta indicada pelo utilizador. A nossa estratégia passa por obter o caminho de cada ficheiro que conste nessa pasta de forma a, posteriormente, proceder com um backup simples sobre esses ficheiros. O nome do “link” em “metadata” referente a um dos ficheiros da pasta a salvaguardar será o próprio caminho desse permitindo uma fácil restauração.

I) Primeiramente executamos “find <client_path>/<folder> -type f” e o resultado deste comando é o conjunto de ficheiros acompanhados do caminho respetivo que se encontram dentro de “<folder>” em “<client_path>”. Essa informação é enviada através de um *pipe* anónimo para o processo pai.

II)Já no processo pai, será lida uma linha por cada iteração do ciclo, ou seja, cada linha representará o caminho do ficheiro a salvaguardar. Como o “output” do comando efectuado em **I)** é da forma “<client_path>/<folder>/<file>” (é apenas um exemplo, o nosso

programa permite salvaguardar tanto uma pasta e um ficheiro singular como várias sub-pastas e ficheiros) então temos de avançar o endereço dessa “string” de forma a obtermos “<folder>/<file>”.

III) Por cada iteração do ciclo, chamamos a função “backup” para proceder com trabalho sobre cada ficheiro individual de “<folder>”. Denote-se que **não** é possível salvar ficheiros em que conste '/' no seu nome.

Para contornar essa situação, o último argumento de “backup” é um inteiro. Se esse inteiro for 1 então estamos perante um ficheiro proveniente de um pedido de “backup” de uma pasta. Assim, o “link” em “metadata” terá o nome do ficheiro com todos as '/' convertidas em ','. Caso esse parâmetro de “backup” seja 0, então o nome do ficheiro não passa por qualquer conversão, trata-se apenas de um ficheiro.

Comando restore (para ficheiros):

Este comando funciona no processo inverso do “backup”. O servidor recebe um ficheiro e começa por procurar a sua existência em *metadata*. Se não existir então a função *restore* retorna um inteiro de insucesso para ser posteriormente enviado um sinal ao cliente avisando que não foi possível recuperar.

Se, por outro lado, existir, então o primeiro passo é renomear a entrada em *metadata* apenas adicionando a extensão “.gz”, de outra forma “gzip” não permite descomprimir.

De seguida descomprimimos o ficheiro link. Devido à sua natureza, ao descomprimirmos o ficheiro *link* estaremos, na verdade, a descomprimir uma cópia do original. O uso de links facilitou imenso esta tarefa.

Por fim, apenas procedemos com um “mv” para a diretoria do cliente. Denote-se que todos os ficheiros recuperados são enviados para a diretoria onde se encontra atualmente o cliente e não para a diretoria original. Além de tornar o código mais simples, também dá acesso direto dos ficheiros ao utilizador que poderia perfeitamente se ter esquecido qual o caminho original do ficheiro.

Assumindo que o servidor recebe um comando do género “restore” “file.txt” então o próximo passo é ir a *metadata* e alterar o link respetivo para “file.txt.gz” onde, de seguida, é descomprimido ficando novamente “file.txt” (já se encontra com o conteúdo do ficheiro original) e por fim movido para a diretoria do cliente.

Comando restore (para pastas):

Ao contrário do *restore* simples para ficheiros. O *restore* de *folders* sofre um processo um pouco mais robusto. A nossa estratégia foi a seguinte:

I) Executar "find metadata/*user*/<folder>,*" para obtermos o conjunto de ficheiros dentro de “<folder>” pertencentes ao utilizador. Como todas as pastas no nome do ficheiro estão delimitadas por vírgulas, basta pedir para o *find* procurar por algum ficheiro cujo nome é composto pela pasta que o utilizador quer recuperar com uma vírgula a seguir e com o asterisco a indicar que a partir daí pode ter qualquer conjunto de caracteres. Esta informação é comunicada ao processo pai através de um *pipe* anónimo.

II) Para cada ficheiro resultado do comando “find”, o processo pai encarrega-se de lê-lo à “saída” do *pipe*. Após o ler com “readln”, cujo resultado fica guardado em “buffer”,

vai recriar todas as pastas que compõem o caminho do ficheiro com “mkdir”. Isto é possível porque vamos ler caractere a caractere de “buffer” e sempre que chegamos a uma vírgula, restauramos a pasta anterior e avançamos para a próxima.

III) Assim que tivermos todas as pastas restauradas é apenas necessário descomprimir e mover o ficheiro para a pasta respetiva. Processo semelhante a um “restore” simples para ficheiros.

Comando delete (para ficheiros):

O comando delete serve para eliminar um determinado ficheiro em “data”, assim como a sua entrada correspondente em “metadata”. Contudo temos de ser cautelosos, pois vários utilizadores podem ter submetido o mesmo ficheiro implicando este ser partilhado.

Logo, o primeiro procedimento passa por verificar se existe algum outro link que esteja a apontar para o ficheiro em “data”. Isto é, primeiramente, vamos à pasta do utilizador respetivo em *metadata* e determinamos qual o ficheiro apontado.

Tendo esta informação, agora vamos a todos os outros utilizadores verificar se existe algum que tenha algum ficheiro a apontar para o ficheiro a ser eliminado. Se se encontrar pelo menos um utilizador que tenha submetido o mesmo ficheiro, então não eliminamos esse em data. Contudo removemos a entrada em *metadata* do utilizador que convocou o comando delete. Se, por outro lado, mais nenhum utilizador tiver submetido o mesmo ficheiro então não é partilhado e pode ser removido com segurança.

Comando delete (para pastas):

A única diferença entre o comando “delete” de ficheiros para pastas é essencialmente o número de ficheiros a serem eliminados. Neste caso, um “delete” de uma pasta pode precisar de eliminar um grande conjunto de ficheiros de “metadata” e, se possível, de “data”.

Portanto, a nossa estratégia começa por listar todos os ficheiros pertencentes à pasta que o utilizador quer remover das suas submissões. Utilizamos o comando “find” para este efeito onde, para cada ficheiro, chamaremos o comando “delete”.

Comando gc:

O comando gc é utilizado para remover ficheiros de “~/.Backup/data” que já não estejam a ser utilizados por nenhum utilizador. Podemos pensar neste comando como uma forma de eliminar “backups” antigos. Assim, a única imposição é precisamente a referida anteriormente, apenas não pode existir nenhum “link” que esteja a apontar para este ficheiro de “backup” em “~/.Backup/data”. A estratégia para a implementação deste algoritmo foi a seguinte:

I) Listar todos os utilizadores de forma a consultar os seus “links” simbólicos posteriormente. Neste contexto, por utilizador entenda-se a pasta com o seu nome que existe em “metadata”. O ouput de “ls -1 metadata” é redirecionado para um ficheiro em disco denominado por “.users”. Portanto, o resultado de “ls -1 metadata” fica

salvaguardado.

II) O seguinte passo é verificar se para cada chave (nome do ficheiro em “data”) não existe nenhum “link” a apontar para a ele. Criamos um processo filho e redirecionamos o *ouput* de "ls -1 data" para um *pipe*. Após sairmos do processo filho, já no processo pai, vamos ler o conteúdo do *pipe* linha a linha.

III) Por cada linha lida, entenda-se, chave resultada de efetuarmos o comando “ls”, vamos abrir “.users” processado em I) e verificar se não existe qualquer “link” a apontar para esse ficheiro., isto é, por cada “readln” de “.users”, vamos criar um subciclo em que constará o nome de todos os “links” do utilizador respetivo.

IV) Enquanto nenhum “link” apontar para a chave que estamos a olhar, utilizamos a função “readlink” para obter a diretoria apontada pelo “link” a ser processado e comparamos o ficheiro apontado pelo “link” com a chave. Se forem iguais, o ficheiro do *digest* respetivo não pode ser eliminado.

V) Caso se chegue ao fim do ciclo em **III)**, significando que não existiu nenhum “link” a apontar para esse *digest*, então esse ficheiro é eliminado de “data”. Caso contrário, não o eliminamos.

Por fim, avançar para a próxima iteração do ciclo em **II)** para processar os outros ficheiros.

Conclusão

No desenvolvimento deste projeto fomos convidados a fortalecer várias técnicas de programação e raciocínios relacionados com sistemas operativos. Técnicas essas que se refletiram nomeadamente na utilização de “links”, acabando por nos facilitar imenso o algoritmo, quer do comando “backup” quer do comando “restore”. Sem ignorar, claramente, a sua contribuição para os comandos “delete” e “gc”.

No início do desenvolvimento do projeto, o grupo ainda não estava a utilizar “links”. Cada utilizador tinha direito a um ficheiro em disco com as correspondências *digest* e nome do ficheiro. Correspondências essas armazenadas em “~/.Backup/metadata”. Como já era de se esperar, encontramos dificuldades na implementação desta estratégia. Após pararmos para repensar no nosso algoritmo, apercebemo-nos que muito certamente tínhamos interpretado erroneamente o enunciado do projeto.

Acabamos, portanto, por utilizar “links”. Notamos o considerável impacto na dificuldade da implementação ao passar de escritas em ficheiros em disco para “links”.

De uma forma geral, foi uma das experiências que mais nos “marcou” no desenvolvimento do projeto de sistemas operativos.

Contudo, seria até questionável se tivéssemos apenas melhorado os nossos conhecimentos no âmbito da utilização de “links”. Já fora desse contexto, tivemos a oportunidade de utilizar quer *pipes* anónimos, quer *pipes* com nome e notar a diferença entre ambos. Apercebemo-nos da facilidade que existe em transmitir informação de um processo filho para um processo pai através da utilização de *pipes* anónimos.

Verificamos, também, que o redireccionamento de descritores nos retira várias limitações que antes tomávamos por inultrapassáveis. Não obstante, também tornam o código muito mais flexível. Um outro exemplo que pensamos que mereça ser mencionado é o facto de termos tido a oportunidade de pôr em prática comunicação com sinais e melhorar a nossa compreensão acerca do seu mecanismo. Reconhecemos que tomávamos a sua utilização por muito mais complexa do que efetivamente é.

Em suma, cremos ter fortalecido vários conhecimentos na temática voltada aos sistemas operativos. Por fim, sem este projeto encontraríamos futuramente dificuldades que foram agora apreendidas e ultrapassadas no seu desenvolvimento.