Relatório Sistemas Operativos

Luís Brito
Gil Gonçalves
Pedro Duarte
{a54056, a67738, a61071}@alunos.uminho.pt

May 20, 2016

Contents

1 Introdução

Este relatório descreve a implementação do trabalho prático da unidade curricular de Sistemas Operativos. Para este trabalho, foi pedido que se desenvolvesse um sistema eficiente de cópias de segurança(backup) para salvaguardar ficheiros de um utilizador. Na implementação do sistema tivemos em conta o requisito de eficiência nomeadamente a questão do espaço, não tendo por isso ficheiros duplicados e o requisito de privacidade, impedindo que o utilizador aceda as pastas diretamente. Procuramos sempre utilizar as chamadas ao sistema leccionadas na unidade curricular.

2 Comunicação Cliente-Servidor

Sendo o Cliente e o Servidor programas independentes é necessário estabelecer uma forma de comunicação entre eles, Isto poderia ser feito de várias maneiras, sendo os `Named Pipes` a tecnologia usada.

2.1 Named Pipes

Foi utilizada a `System Call`

```
int mkfifo(const char*,mode_t)
```

de forma a criar um ficheiro que servirá de meio de comunicação entre o cliente e o servidor. Este ficheiro foi chamado de `toServidor`. Este ficheiro é inicialmente criado pelo servidor, pelo que se o cliente tenta escrever no Pipe sem este estar criado, não e feita qualquer ação e é ignorada a instrução do cliente, caso o ficheiro toServidor exista, mas por algum motivo o servidor não esteja ativo, o servidor não consegue arrancar antes que este seja apagado. No caso do servidor, este cria o ficheiro e fica a espera que um cliente envie informação para o pipe.

3 Funcionalidades

De forma a ser possível criar as cópias de segurança e restaurar os ficheiros indicados,foram criadas duas funcionalidades, nomeadamente `Backup` e `Restore`.

3.1 Backup

A funcionalidade `Backup` é utilizada para criar uma cópia do nosso ficheiro. Para isso, é-lhe passado como argumento o nome do ficheiro que queremos copiar:

```
~$./Cliente backup teste.txt
```

Depois é feito o sha1sum, que cria uma string com o `digest` relativo ao ficheiro e com o nome do ficheiro `(ex:"bc38e2f249dbf7c74eaa8dh38dh88d3233h teste.txt")`. Depois é feito o `gzip`, ao nome do ficheiro e este é colocado na sub-diretoria `data/` com o nome do seu `digest`.Por fim é feito o comando `ln` que cria na sub-diretoria `metadata/` uma ligação com o nome original do ficheiro para o seu conteúdo comprimido na sub-directoria `data/`.

3.2 Restore

A funcionalidade `Restore` é utilizada para restaurar o ficheiro original. Para isso, é-lhe passado como argumento o nome do ficheiro que queremos restaurar:

```
~$./cliente restore teste.txt
```

Depois é feito o `ls -l` para verificar se o ficheiro existe, se o ficheiro existir, é usado `awk` para imprimir o `digest` do ficheiro, guardámos esse `digest` e removemos esse ficheiro usando o `rm`. Depois vamos a sub-diretoria `data/` fazemos o `cp` desse `digest` para a nossa diretoria e finalmente extrai-mos o ficheiro usando o `gnuzip` desse digest.

3.3 Delete

A funcionalidade `Delete`, dado um nome do ficheiro presente na sub-directoria `data/`, apaga esse ficheiro se ele não estiver a ser utilizado na sub-directoria `metadata/` :

```
~$./cliente delete "digest"
```

É feito o `ls -l` na sub-directoria `metadata/` para listar os ficheiros e respetivos links, depois usa-se o `grep` para encontrar esse ficheiro e depois utilizamos o `wc -l` para contar as ocorrências desse ficheiro, se o resultado devolvido for 0 apaga-se usando o comando `rm "data/nomeficheiro"`, caso contrario não.

3.4 GC

A funcionalidade `GC`, acede a sub-directoria `data/`, guarda os nomes dos ficheiros e depois acede a sub-directoria `metadata/` e apaga os ficheiros que não estão a ser utilizados:

```
~$./cliente gc
```

É feito o `ls -1` na sub-directoria `data/` para listar os ficheiros, guardamos o nome dos ficheiros numa estrutura de dados e a medida que é percorrida a estrutura é feito o `ls -l` na sub-diretoria `metadata/` para listar os ficheiros e respetivos links, depois usa-se o `grep` para encontrar esse ficheiro e depois utilizamos o `wc -l` para contar as ocorrências desse ficheiro, se o resultado devolvido for 0 apaga-se usando o comando `rm data/"nomeficheiro"`, caso contrario não.

4 Servidor

4.1 Estruturas de Dados

De forma a podermos guardar o `digest` para verificar se existe algum link associado na sub-directoria metadada, criamos uma estrutura de dados, `Lista Ligada` :

```
typedef struct listaligada{
char *nomeDiguest;
struct listaligada *prox;
}*ListaLigada;
```

As estruturas acima são inicializadas quando o `cliente` usa o comando `gc`, que acede `data/` e guarda os nomes dos ficheiros. Na sub-directoria `metadata` efectuamos o `ls -l` para obter o link associado ao ficheiro da sub-directoria `data/` e depois usamos o comando `grep` para verificar se os link que estão na estrutura de dados, são os mesmos que estão no link, senão for os ficheiros são apagados.

4.2 Comunicação Pai-Filhos

Quando o servidor recebe pedidos de `backup`,`restore`,`delete` e `gc` este tem de delegar os pedidos para processos filhos. Para isso é necessário um meio de comunicação entre o servidor e os seus filhos.

4.2.1 Pipes Anónimos

O meio de comunicação escolhido para delegar os pedidos foi o uso de pipes anónimos. Criamos um pipe entre o servidor e processos usando a seguinte chamada ao sistema

```
int pipe(pid[2])
```

De seguida, é necessário ter o cuidado de fechar todos os descritores que não devem ser utilizados pelos processos errados, sendo que o processo pai lê, logo fecham-se todos os descritores de escrita e os processos filhos escrevem,logo fecham-se todos os descritores de leitura. Depois redireciona-se os descritores padrão `0-stdin` e `1-stdout` para o respectivo descritor do pipe e logo de seguida fecham-se os respetivos descritores. Se não tivéssemos este cuidado, isto poderia criar varias situações que conduziriam a um comportamento errado do sistema. Por exemplo, no caso de um processo filho morrer e o respectivo descritor continuasse aberto em outros processos.

4.3 Sinais do Sistema

Na realização deste trabalho foi usado sinais para a comunicação entre o servidor e o cliente, ou seja o servidor responde ao cliente através de sinais. Foi usado o sinal `SIGUSR1` para avisar o utilizador que as operações que ele efectou correram bem e usamos o sinal `SIGUSR2` para informar o utilizador que as operações correram mal.

```
int copiou=-10; /* variável global usada no sinal */

void recebeSinais(int sinal){

    switch (sinal) {
        case SIGUSR1:
           copiou=1;
            break;
        case SIGUSR2:
            copiou=0;
            break;
          default:
            break;
    }
}
```

4.4 Funções auxiliares

Foi criado funções auxiliares de modo a garantir um maior controlo dos dados. Visto que nas varias funções auxiliares é feito no minimo um `fork` se não fosse usadas funções auxiliares não iria ser tão fácil a percepção do codigo.

4.5 Funcionamento

Quando o servidor é iniciado ele fica a espera de comandos escritos pelo cliente, se o cliente escrever `backup` terá de passar no minimo um ficheiro que se encontra na sua directoria local. Supondo que o cliente escolhe passar ficheiros com a extenção .txt. Se o cliente preferir poderá fazer o seguinte comando `backup *.txt`, onde é passado ao servidor todos os ficheiros com a extenção .txt. O servidor recebe os ficheiros um a um e depois irá comprimir o ficheiro e guarda-lo numa sub-directoria `data` e na sub-directoria `metadata` guarda um ficheiro com o nome teste.txt que irá ser o link para o seu `digest`. Se o cliente quiser restaurar o ficheiro tem de escrever o seguinte comando `restore teste.txt` e o que o servidor irá fazer é ir a sub-directoria `metadata` ver se o ficheiro existe e se o ficheiro existir o servidor retira o seu `digest`,acede a sub-directoria `data` e copia esse ficheiro para a directoria local, na directoria local extrai o ficheiro e muda-lhe o nome para o nome que foi passado como argumento. Se o cliente escrever o comando `delete nomeDigest` o servidor acede a sub-directoria `metadata` e verifica se existe algum ficheiro com o link para esse `digest`, através

do comando `ls -l`, se existir ele não irá apagar o ficheiro, senão apaga-o. Se o cliente escrever `gc` o servidor irá guardar o nome dos ficheiros que estão na sub-directoria `data` e depois efectua os mesmos passos que o comando `delete`.

4.6 Makefile

De forma a evitar a compilação de arquivos desnecessários e automatizar as tarefas foi criada uma makefile:

```
default:all

clear:
    clear

clean:
    rm -f *.o cliente servidor toServidor

dire:
    rm -r data metadata

listaLigada.o: listaLigada.c listaLigada.h
    gcc -c listaLigada.c -Wall

servidor: servidor.c listaLigada.o
    gcc -o servidor servidor.c listaLigada.o -Wall

cliente: cliente.c
    gcc -o cliente cliente.c -Wall

all: servidor cliente
```

5 Dados de teste

Criamos vários ficheiros .txt para verificar o correto funcionamento do programa. Esses ficheiros foram guardados com informação diferente para verificar se o seu digest era diferente. Na pasta do trabalho temos quatro ficheiros para teste, nomeadamente teste1.txt, teste2.txt, teste3.txt e teste4.txt. O teste1.txt e teste2.txt contêm o mesmo conteúdo, enquanto o teste3.txt e teste4.txt são diferentes dos restantes.

6 Conclusão

Este trabalho constituiu um desafio desde o inicio até a sua conclusão. Os obstáculos que apareceram pelo caminho contribuíram para o crescimento da nossa capacidade de raciocínio e para a melhoria na compreensão das matérias dadas. Umas das dificuldades que mais apareceu foi o facto de quando chamávamos uma função auxiliar essa função não voltava para a main, isto porque ao executar o exclp o programa "morria", logo nunca iria regressar a função que a chamou. Outra dificuldade foi a parte dos sinais, porque não sabíamos como mandar os sinais para o cliente. A medida que realizava-mos o trabalho tivemos a noção que certos aspectos poderiam ser melhorados e só não foram melhorados devido a falta de tempo. Para trabalho futuro fica a implementação do servidor não efectuar mais do que cinco processos ao mesmo tempo.