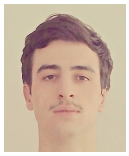
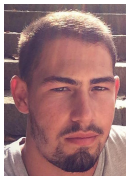
Backup Eficiente
Universidade do Minho
Licênciatura em Ciências da Computação

21 de Maio de 2016


Hugo Fernandes, a68683


João Gomes, a70400


Paulo Barbosa, a70073

Resumo

Este relatório foi realizado no âmbito da unidade curricular de Sistemas Operativos, no curso Ciências da Computação da Universidade do Minho, sendo o trabalho proposto a componente prática da mesma, que envolve o desenvolvimento de um sistema eficiente de cópias de segurança (backup), para salvaguardar ficheiros de um utilizador. Pretende-se, assim, dar uma ideia de todo o trabalho envolvido, nomeadamente da preparação, do modo de abordagem do problema e de alguns conteúdos, de modo a auxiliarem a compreensão deste trabalho.

Conteúdo

1 Introdução

Este trabalho foi realizado no âmbito da unidade curricular de Sistemas Operativos, do 2º ano da Licenciatura em Ciências da Computação da Universidade do Minho, com o objetivo de se desenvolver um sistema eficiente de cópias de segurança, semelhante à conhecida ***Dropbox***. Este sistema, desenvolvido na linguagem **C**, teria que realizar as seguintes tarefas:

- Ser capaz de efetuar o backup de ficheiros
- Ser capaz de efetuar o restore dos ficheiros guardados no backup

2 Comunicação

A comunicação entre o cliente e o servidor foi implementada através de um pipe com nome, usando a função **mkfifo** de *Unix*, como se pode observar de seguida:

```
if (access("fifo",F_OK)==-1){
    int f = mkfifo("fifo",0666);
    if(f<0){
      perror("Erro no fifo.\n");
      exit(-1);
    }
  }
```

Neste caso, podemos observar que o pipe com nome (fifo) apenas é criado caso não exista na diretoria onde o nosso programa se encontre. A função responsável

por fazer esse teste é a função **access**, cujo valor de retorno é -1 caso já exista um pipe com esse nome; diferente de -1 caso contrário. Após garantirmos que o pipe não existe, criamos o pipe, e damos permissão de escrita e de leitura.

3 Cliente

O cliente é a parte do sistema que é responsável por interagir com o utilizador, ou seja, o utilizador apenas tem permissão para dar instruções ao cliente, instruções essas que vão ser enviadas, através do pipe com nome mencionado acima, para o servidor e que iremos explicar de seguida.

4 Servidor

Após iniciado o servidor, são criadas duas diretorias (Data e MetaData) que serão explicadas mais à frente, e aberto o pipe. O programa bloqueia enquanto o cliente não enviar nada para o servidor (pelo pipe). Depois do cliente enviar esses comandos, o servidor guarda a informação da seguinte forma:

1. Começa por guardar a linha de comandos num array;
2. Percorre o array dividindo a string recebida por espaços brancos, com a função **wordexp**, guardando cada substring num array;

4.1 Backup

Como mencionado no ponto número **3**, existem vários tipos de comandos que o utilizador pode enviar pelo cliente, sendo um deles, precisamente, o **backup**. Neste caso, o objetivo é efetuar a cópia de segurança do/dos ficheiro/ficheiros em questão. O comando **backup** chama-se da seguinte maneira (exemplo):

```
./cliente backup a.txt b.jpg
```

E, após todos os passos descritos atrás, obtemos o seguinte array de strings:

```
w = {"./cliente","backup","a.txt","b.jpg"};
```

4.2 Restore

Outro tipo de comando que o utilizador pode enviar pelo cliente é o **restore**. Neste caso, o objetivo é restabelecer o conteúdo salvaguardado no sistema, de um ou mais ficheiros em simultâneo. O comando **restore** chama-se da seguinte maneira (exemplo):

```
./cliente restore a.txt b.jpg
```

E, após todos os passos descritos atrás, obtemos o seguinte array de strings:

```
w = {"./cliente","restore","a.txt","b.jpg"};
```

4.3 Delete

Por fim temos o **delete** no qual o objetivo é apagar a uma ou mais cópias de segurança já efetuadas, removemos o link associado ao ficheiro que queremos remover e removemos também o ficheiro em si. O comando **delete** chama-se da seguinte maneira (exemplo):

```
./cliente delete a.txt b.jpg
```

E, após todos os passos descritos atrás, obtemos o seguinte array de strings:

```
w = {"./cliente","delete","a.txt","b.jpg"};
```

4.4 Data

Nesta diretoria guardamos o ficheiro comprimido, cujo nome é o resultado da chamada do programa **sha1sum** sobre o ficheiro original. No momento em que queres fazer o restore de um ficheiro já previamente salvaguardado, descomprimimos nessa diretoria e usamos o comando **mv** para mover esse ficheiro descompactado para a diretoria original.

4.5 MetaData

Esta diretoria vai conter os ficheiros produzidos pela função **symlink**, que garantem a correspondência entre o ficheiro original e o caminho completo para o seu ficheiro comprimido, respectivo, que está guardado na pasta Data. Estes links servem para quando estivermos a fazer um **restore** ou um **delete**, vermos qual é o ficheiro (na pasta Data) correspondente ao que queremos apagar/restaurar.

5 Conclusão

Neste trabalho conseguimos por em prática os conhecimentos adquiridos nas aulas, de forma a fortalecer o esse conhecimento para uma melhor preparação para o teste final da disciplina. Um trabalho desta dimensão claro que ainda tem

alguns pontos a melhorar, mas tendo em conta que perdemos tempo com erros "óbvios", não conseguimos cumprir todos os pontos que nos foram propostos. No entanto, achamos que sem este trabalho ia ser muito mais complicado fazermos um bom estudo para esta unidade curricular.