Trabalho de Sistemas Operativos
Processamento de um notebook

Sérgio Oliveira
a62134

Pedro Dias
a63389

21 de Maio de 2018

Conteúdo

Capítulo 1

Introdução

Com este relatório iremos demonstrar o desenvolvimento de um programa que faz o processamento de ficheiros de formato *notebook*, formato designado pelo trabalho prático.

Ao ser executado, o argumento terá de ter um caminho válido para um ficheiro *notebook* para funcionar corretamente.

Relacionado aos temas que falamos, apresentaremos fragmentos de código ao longo do relatório, para que haja melhor compreensão do que estamos a explicar.

Este relatório está dividido em várias secções que correspondem aos pontos mais importantes do trabalho prático, como vemos na página de conteúdos.

Capítulo 2

Execucão

2.1 Noções básicas

A funcionalidade básica do nosso programa está em ler o nosso *notebook* linha a linha e termina a sua execução até que o ficheiro passado como argumento tenha sido lido pelo *while* até à linha final.

Este cíclo é o motor de todo o nosso programa, que está a ser controlado através de flags e da função read, que ao ser executada, retorna o valor de bytes que foram lidos. Se este valor retornado for negativo, então o *system call* está a retornar um erro. Esta é então a nossa maneira de parar o cíclo, executando-o até que o nosso read fique sem mais bytes para ler.

```
    while ((read = getline(&line, &len, fp)) != -1 && flagErrorFork
       ==1 && (running)) {
```

Antes de guardarmos numa variável, a linha lida é processada para que reconheça o conjunto de caratéres especiais para o correto processamento do ficheiro:

- $;
- $|;
- $(número)|;
- >>> e <<<

2.2 Sinais

Uma das flags é usada para depuração de erros de qualquer fork criado. Se o fork retorna um *status* diferente de sucesso, pode significar que a linha lida contém um comando errado. Neste caso, teriamos então de cancelar a execução do nosso *notebook*.

```
int forkError(int status,char *b){
```

No entanto também pode haver ação humana e, para isso, o sinal *SIGINT* é enviado para o programa (sinal normalmente relacionado com a combinação de botões *Control+C*). Nesse caso, a nossa variável global *running* irá determinar o estado de execução do nosso programa.

```
static volatile int running = 1;
void handler(int dummy){
    running = 0;
}
```

A detecção do sinal deve ser inicializada na main pela função de sistema signal.

Uma linha delimitada por qualquer outra expressão diferente dos itens em cima irá ser ignorada e não interpretada como comando.

2.3 Re-processamento

A funcionalidade normal do nosso programa será sempre executar e inserir os nossos resultados entre >>> e <<<. Haverá no entanto alturas em que faremos alterações ao nosso sistema (criamos um ficheiro novo, o *word count* de determinado ficheiro é agora maior, o estado de X dispositivo foi alterado, entre outros).

```
void re_processamento(char * file){
```

Nestes casos, necessitamos então de voltar a executar as linhas de comando do nosso *notebook*.

A função re-processamento abre então um ficheiro temporário para podermos fazer o *parsing* do nosso ficheiro de entrada, fazendo com que o original não seja imediatamente alterado.

```
    FILE *REDO;
    REDO = fopen("REDO.txt", "wr+");
```

Executamos então outro ciclo da mesma natureza do *while* da função main, lendo linha a linha e, desta vez, ignorando todo o conteudo entre >>> e <<< do ficheiro de entrada.

Após isto tudo, usámos a função rename para relocarmos o ficheiro temporário para o original.

```
    rename("REDO.txt",file);
```

2.4 Detecção de erros / Interrupção

Capítulo 3

Outras Funcionalidades

3.1 Histórico de comandos anteriores

3.2 Execução de conjuntos de comandos

Capítulo 4

Conclusão