Universidade do Minho

Mestrado Integrado em Engenharia Informática

2º ano 2º Semestre

Processamento de Notebooks

Autores:
João Palmeira 73864
José Dias 78494
Rafael Silva 74264

Universidade do Minho

2 de Junho de 2018

Conteúdo

1 Introdução

Neste trabalho pretende-se construir um sistema de processamento de *Notebooks*. Um *Notebook* é um ficheiro de *.nb* que contem vários comandos e que depois de processado será alterado de modo a conter os resultados desses comandos.

Numa fase mais avançada o programa deve ser capaz de aceder ao resultado de um comando anterior arbitrário e também capaz da execução de conjuntos de comandos.

Este projeto foi desenvolvido na linguagem C no âmbito da unidade curricular de Sistemas Operativos.

2 Desenvolvimento

2.1 Execução de programas

Neste projeto foram desenvolvidas várias funcionalidades, entre elas está a execução de programas, isto é, à medida que se lê uma linha do ficheiro de *input*, verifica-se se esse comando é inicializado por $ ou $|. Caso seja o primeiro ele irá executar esse comando e irá guardar esse output num ficheiro auxiliar para mais logo ser consultado e, ao mesmo tempo irá escrever o *output* dessa execução delimitado por >>> e <<< e começado pelo comando em questão.

De seguida temos um comando começado por $|, onde esse comando irá usar o output do comando executado anteriormente, ou seja, como é guardado sempre o *output* de cada comando executado num ficheiro temporário, será mais fácil de aceder a essa *input*, basta abrir o ficheiro e ler o seu conteúdo.

Para executar esta tarefa usamos *pipe's* anónimos, que fazem a escrita no ficheiro temporário para depois ser efetuada uma leitura e escrita da informação pretendida no ficheiro com o resultado final.

Resumidamente, a medida que vamos ler um comando do ficheiro *input* vamos ao mesmo tempo executar esse comando e escrever o seu resultado num ficheiro temporário e no ficheiro final.

2.2 Reprocessamento de um *Notebook*

Para executar esta funcionalidade da-mos ao ficheiro executável o nome do ficheiro com um resultado final e irá ler esse ficheiro e executar cada um dos comandos existente no ficheiro e guardar o ser output num ficheiro temporário, se verificar que executa todos os comandos sem nenhum erro, o programa renomeia esse ficheiro temporário para o novo ficheiro com o resultado final, caso encontre algum erro o programa não irá alterar o estado do ficheiro que contem o resultado inicial.

2.3 Controlo de Erros

No que toca a controlo de erros por parte do programas temos vários métodos para os detetar e lançar para o *stderr* o que se passou de errado com a execução de certa parte do código.

Para fazermos tal controlo, nos por cada *execvp* temos a seguir um tratamento de erros, primeiros temos um *kill* que nos assegura se o *execvp* der erro temos um sinal que tratar e da *kill* e diz-nos onde ocorreu o erro, através da seguinte função *signal(SIGUSR1,sigHandler)*.

De seguida temos vários *exit* diferentes para cada situação possível de saída inesperada do programa (de -1 a -7).

Para o tratamentos dos sinais temos uma variável chamada *flag* que nos indica se existiu algum erro com o *execvp*, caso não tivesse dado erro o programa irá escrever o *output* de *execvp* num ficheiro temporário e noutro ficheiro temporário de *output* por cada instrução executada, caso contrario iremos imprimir

o erro com a função *perror*.

Também temos tratamento de erros para abertura de ficheiros.

2.4 Acesso ao Resultado de um Comando Anterior Arbitrário

Esta é umas das funcionalidades avançadas e o que nos pede é que dado um comando que tenha atrás do mesmo o conjunto de $n|, em que n será um numero, isto é, o n-ésimo comando anterior. O que esta funcionalidade do pede é ir buscar o output do n-ésimo comando anterior e fornece-lo como *input* ao comando que contem $n|.

Como nos anteriormente na funcionalidade de *Execução de programas* nos servimos de vários ficheiros auxiliares para guardar o *output* individual de cada um dos comandos, nos podemos usar esses ficheiros para ir buscar o output pretendido e passa-lo através de *pipes* para o comando pretendido como *input* do mesmo, sendo assim mais eficiente e eficaz a executar estas instruções pois o *input* do mesmo já se encontra calculado e é só ir busca-lo.

3 Execução e Resultados

3.1 Instruções de Execução

```
$ make
make notebook
cc notebook.c -o notebook
$ ./notebook ficheiro
```

3.2 Resultados

Apresentamos agora dois exemplos de ficheiro *nb* passado como argumento ao Processador de um *Notebook*.

- **Primeiro exemplo:**

```
$ ls
$| sort
$1| head -2
$| head -4
$| head -2
$| head -1
$3| sort
```

- **Segundo exemplo:**

```
$ ls -l
$| sort
$1| head -4
$|head -2
```

- **Terceiro exemplo:**

```
$ ls -l
$| grep b
$| sort -r
$2| wc -l
```

- **Quarto exemplo:**

```
$ ps -l
$| sort
$| head -2
$| wc -l
```

Para executar o Processador é necessário no terminal correr o comando *make* (onde é invocado os comandos que estão presentes no ficheiro *Makefile*) para gerar o executável. Após isso invoca-se o executável ao qual é passado como argumento um ficheiro *nb* idêntico ao exemplo anterior. O Processador apaga esse ficheiro *nb* e gerar um novo com os resultados, tal como o que apresento a seguir (gerado através do exemplo acima):

Resultado do primeiro exemplo:

```
$ ls
>>>
exemplo1.nb
exemplo2.nb
exemplo3.nb
exemplo4.nb
makefile
notebook
notebook.c
result.nb
temp.nb
<<<
$| sort
>>>
exemplo1.nb
exemplo2.nb
exemplo3.nb
exemplo4.nb
makefile
notebook
notebook.c
result.nb
temp.nb
<<<
$1| head -2
>>>
exemplo1.nb
exemplo2.nb
<<<
$| head -4
>>>
exemplo1.nb
exemplo2.nb
<<<
$| head -2
>>>
exemplo1.nb
exemplo2.nb
```

```
<<<
$| head -1
>>>
exemplo1.nb
<<<
$3| sort
>>>
exemplo1.nb
exemplo2.nb
<<<
```

Resultado do segundo exemplo:

```
$ ls -l
>>>
total 104
-rw-r--r--@ 1 rafaelsilva  staff     67  1 Jun 22:03 exemplo1.nb
-rw-r--r--@ 1 rafaelsilva  staff     38  2 Jun 20:56 exemplo2.nb
-rw-r--r--@ 1 rafaelsilva  staff     39  2 Jun 21:06 exemplo3.nb
-rw-r--r--@ 1 rafaelsilva  staff     36  2 Jun 21:05 exemplo4.nb
-rw-r--r--@ 1 rafaelsilva  staff    103  1 Jun 22:00 makefile
-rwxr-xr-x  1 rafaelsilva  staff  13900  2 Jun 21:10 notebook
-rw-r--r--@ 1 rafaelsilva  staff   6803  2 Jun 21:10 notebook.c
-rw-------@ 1 rafaelsilva  staff    421  2 Jun 21:10 result.nb
-rw-------  1 rafaelsilva  staff     12  2 Jun 21:15 temp.nb
<<<
$| sort
>>>
-rw-------  1 rafaelsilva  staff     12  2 Jun 21:15 temp.nb
-rw-------@ 1 rafaelsilva  staff    421  2 Jun 21:10 result.nb
-rw-r--r--@ 1 rafaelsilva  staff     36  2 Jun 21:05 exemplo4.nb
-rw-r--r--@ 1 rafaelsilva  staff     38  2 Jun 20:56 exemplo2.nb
-rw-r--r--@ 1 rafaelsilva  staff     39  2 Jun 21:06 exemplo3.nb
-rw-r--r--@ 1 rafaelsilva  staff     67  1 Jun 22:03 exemplo1.nb
-rw-r--r--@ 1 rafaelsilva  staff    103  1 Jun 22:00 makefile
-rw-r--r--@ 1 rafaelsilva  staff   6803  2 Jun 21:10 notebook.c
-rwxr-xr-x  1 rafaelsilva  staff  13900  2 Jun 21:10 notebook
total 104
<<<
$1| head -4
>>>
-rw-------  1 rafaelsilva  staff     12  2 Jun 21:15 temp.nb
-rw-------@ 1 rafaelsilva  staff    421  2 Jun 21:10 result.nb
-rw-r--r--@ 1 rafaelsilva  staff     36  2 Jun 21:05 exemplo4.nb
-rw-r--r--@ 1 rafaelsilva  staff     38  2 Jun 20:56 exemplo2.nb
<<<
$|head -2
```

```
>>>
-rw-------  1 rafaelsilva  staff     12  2 Jun 21:15 temp.nb
-rw-------@ 1 rafaelsilva  staff    421  2 Jun 21:10 result.nb
<<<
```

Resultado do terceiro exemplo:

```
$ ls -l
>>>
total 104
-rw-r--r--@ 1 rafaelsilva  staff     67  1 Jun 22:03 exemplo1.nb
-rw-r--r--@ 1 rafaelsilva  staff     38  2 Jun 20:56 exemplo2.nb
-rw-r--r--@ 1 rafaelsilva  staff     39  2 Jun 21:06 exemplo3.nb
-rw-r--r--@ 1 rafaelsilva  staff     36  2 Jun 21:05 exemplo4.nb
-rw-r--r--@ 1 rafaelsilva  staff    103  1 Jun 22:00 makefile
-rwxr-xr-x  1 rafaelsilva  staff  13900  2 Jun 21:10 notebook
-rw-r--r--@ 1 rafaelsilva  staff   6803  2 Jun 21:10 notebook.c
-rw-------@ 1 rafaelsilva  staff   1612  2 Jun 21:15 result.nb
-rw-------  1 rafaelsilva  staff     12  2 Jun 21:17 temp.nb
<<<
$| grep b
>>>
-rw-r--r--@ 1 rafaelsilva  staff     67  1 Jun 22:03 exemplo1.nb
-rw-r--r--@ 1 rafaelsilva  staff     38  2 Jun 20:56 exemplo2.nb
-rw-r--r--@ 1 rafaelsilva  staff     39  2 Jun 21:06 exemplo3.nb
-rw-r--r--@ 1 rafaelsilva  staff     36  2 Jun 21:05 exemplo4.nb
-rwxr-xr-x  1 rafaelsilva  staff  13900  2 Jun 21:10 notebook
-rw-r--r--@ 1 rafaelsilva  staff   6803  2 Jun 21:10 notebook.c
-rw-------@ 1 rafaelsilva  staff   1612  2 Jun 21:15 result.nb
-rw-------  1 rafaelsilva  staff     12  2 Jun 21:17 temp.nb
<<<
$| sort -r
>>>
-rwxr-xr-x  1 rafaelsilva  staff  13900  2 Jun 21:10 notebook
-rw-r--r--@ 1 rafaelsilva  staff   6803  2 Jun 21:10 notebook.c
-rw-r--r--@ 1 rafaelsilva  staff     67  1 Jun 22:03 exemplo1.nb
-rw-r--r--@ 1 rafaelsilva  staff     39  2 Jun 21:06 exemplo3.nb
-rw-r--r--@ 1 rafaelsilva  staff     38  2 Jun 20:56 exemplo2.nb
-rw-r--r--@ 1 rafaelsilva  staff     36  2 Jun 21:05 exemplo4.nb
-rw-------@ 1 rafaelsilva  staff   1612  2 Jun 21:15 result.nb
-rw-------  1 rafaelsilva  staff     12  2 Jun 21:17 temp.nb
<<<
$2| wc -l
>>>
       8
<<<
```

Resultado do quarto exemplo:

```
$ ps -l
>>>
  UID   PID  PPID        F CPU PRI NI       SZ    RSS WCHAN     S    ADDR TTY   TIME CMD
  501  2480  2479     4006   0  31  0  4296240   1736 -       S  0 ttys000    0:00.27 -bash
  501  2726  2480     4006   0  31  0  4267732    804 -       S+     0 ttys000    0:00.00
<<<
$| sort
>>>
  501  2480  2479     4006   0  31  0  4296240   1736 -       S  0 ttys000    0:00.27 -bash
  501  2726  2480     4006   0  31  0  4267732    804 -       S+ 0 ttys000    0:00.00
                                                                        ./notebook exemplo4.nb
  UID   PID  PPID        F CPU PRI NI       SZ    RSS WCHAN     S    ADDR TTY   TIME CMD
<<<
$| head -2
>>>
  501  2480  2479     4006   0  31  0  4296240   1736 -       S   0 ttys000    0:00.27 -bash
  501  2726  2480     4006   0  31  0  4267732    804 -       S+  0 ttys000    0:00.00
                                                                        ./notebook exemplo4.nb
<<<
$| wc -l
>>>
       2
<<<
```

Resultado do reprocessamento do resultado final do terceiro exemplo:

- Foi mudado o *grep b* para *grep a*.

```
$ ls -l
>>>
total 96
-rw-r--r--@ 1 rafaelsilva  staff     38  2 Jun 20:56 exemplo2.nb
-rw-r--r--@ 1 rafaelsilva  staff     39  2 Jun 21:06 exemplo3.nb
-rw-r--r--@ 1 rafaelsilva  staff     36  2 Jun 21:05 exemplo4.nb
-rw-r--r--@ 1 rafaelsilva  staff    103  1 Jun 22:00 makefile
-rwxr-xr-x  1 rafaelsilva  staff  13900  2 Jun 22:05 notebook
-rw-r--r--@ 1 rafaelsilva  staff   6803  2 Jun 22:05 notebook.c
-rw-------@ 1 rafaelsilva  staff   1486  2 Jun 22:06 result.nb
-rw-------  1 rafaelsilva  staff     12  2 Jun 22:06 temp.nb
<<<
$| grep a
>>>
total 96
-rw-r--r--@ 1 rafaelsilva  staff     38  2 Jun 20:56 exemplo2.nb
-rw-r--r--@ 1 rafaelsilva  staff     39  2 Jun 21:06 exemplo3.nb
-rw-r--r--@ 1 rafaelsilva  staff     36  2 Jun 21:05 exemplo4.nb
-rw-r--r--@ 1 rafaelsilva  staff    103  1 Jun 22:00 makefile
-rwxr-xr-x  1 rafaelsilva  staff  13900  2 Jun 22:05 notebook
```

```
-rw-r--r--@ 1 rafaelsilva  staff   6803  2 Jun 22:05 notebook.c
-rw-------@ 1 rafaelsilva  staff   1486  2 Jun 22:06 result.nb
-rw-------  1 rafaelsilva  staff     12  2 Jun 22:06 temp.nb
<<<
$| sort -r
>>>
total 96
-rwxr-xr-x  1 rafaelsilva  staff  13900  2 Jun 22:05 notebook
-rw-r--r--@ 1 rafaelsilva  staff   6803  2 Jun 22:05 notebook.c
-rw-r--r--@ 1 rafaelsilva  staff    103  1 Jun 22:00 makefile
-rw-r--r--@ 1 rafaelsilva  staff     39  2 Jun 21:06 exemplo3.nb
-rw-r--r--@ 1 rafaelsilva  staff     38  2 Jun 20:56 exemplo2.nb
-rw-r--r--@ 1 rafaelsilva  staff     36  2 Jun 21:05 exemplo4.nb
-rw-------@ 1 rafaelsilva  staff   1486  2 Jun 22:06 result.nb
-rw-------  1 rafaelsilva  staff     12  2 Jun 22:06 temp.nb
<<<
$2| wc -l
>>>
       9
<<<
```

4 Conclusão

Este projeto foi desenvolvido no âmbito da unidade curricular de Sistemas Operativos. Ao longo do processo de desenvolvimento encontrámos algumas barreiras que dificultaram este projeto.

Um dos obstáculos com que nos deparamos foi o de conseguir produzir uma estrutura capaz de resolver o problema em causa, tendo em conta as condições impostas. O facto de o trabalho envolver vários componentes como *pipes* anónimos, sinais, descritores, *pipes* com nome, escrita/leitura de ficheiros, entre outros, dificultou bastante esta etapa quanto á sua organização.

Outra dificuldade do processo foi manter uma comunicação entre os *pipes*, isto é, manter uma sincronização ao longo dos processos entre os ficheiros de *output* temporários e a leitura e escrita dos mesmos através dos *pipe's* de forma correta. Servindo então para recolher a informação certa no local certo .

Em suma, achamos que o trabalho corresponde as expectativas e cumpre de modo geral os requisitos pedidos. Mais importante ainda, ajudou-nos a perceber como é feita a gestao, criação e comunicação entre processos, a escrita e leitura de ficheiros (e nao só), e a utilização de sinais.