Universidade do Minho

Sistemas Operativos
(2º ano, Curso LCC)
Relatório do Trabalho Pratico

João Siva (A70755) Leonel Gonçalves (AE3603)
Tiago Loureiro (A71191)
Grupo LCC-1

2 de Junho de 2018

Conteúdo

1 Introdução **2**

2 Main.c **3**

3 helpers.h **5**

4 helpers.c **6**
- 4.1 readln . . . 6
- 4.2 randomName . . . 6
- 4.3 analyse . . . 7
- 4.4 gatherArg . . . 7
- 4.5 execute . . . 7
- 4.6 executeNumPipe . . . 8
- 4.7 printline . . . 8
- 4.8 terminate . . . 8

5 Makefile **9**

6 Testes **10**
- 6.1 Teste 1 . . . 10
- 6.2 Teste 1.1 . . . 11
- 6.3 Teste 2 . . . 13

7 Conclusão **15**

Capítulo 1

Introdução

O propósito deste relatório é dar um resumo do funcionamento do trabalho que nos foi proposto na UC de Sistemas Operativos. O nosso trabalho é composto por 4 ficheiros:

```
main.c
helpers.h
helpers.c
Makefile
```

A estrutura do nosso relatório vai se basear exatamente nesta divisão do trabalho por estes ficheiros. No fim do relatório vai também em anexo o código dos ficheiros acima referidos.

Capítulo 2

Main.c

Como o nome do ficheiro leva a entender, este é o ficheiro principal do nosso trabalho.

No início quando começamos a idealizar a estrutura que o nosso projeto iria tomar, deparamo-nos como iriamos armazenar o resultado das execuções dos comandos, ou em caso de erro como iriamos fazer para o ficheiro a ser processado se mantivesse inalterado.

Na altura surgiram duas opções, tudo seria guardado em buffers ou tudo seria guardado em ficheiros. No entanto optamos por armazenar em ficheiros devido ao tamanho do output dos comandos não ser previsivel para ser possível fazer malloc de memória para os buffers. Então os ficheiros seriam mais fiaveis para a conceção do nosso projeto.

Mas o uso de ficheios também tem os seus precalços, temos que garantir que ao criar um ficheiro não existiria um ficheiro na mesma diretoria com o mesmo nome. Mais uma vez surgiram 2 opções ou os ficheiros eram criados com nomes aleatórios ou era criado uma diretoria com um nome aleatório dentro da *working directory* e os ficheiros eram criados dentro da nova diretoria.

Ora, o uso da diretoria é mais fiavel, pelo que embora a diretoria tenha um nome aleatório, o nome que dá-mos aos ficheiros que criamos pode ser escolhido por nós sem que haja colisão de nomes.

Resumidamente a nossa main.c no inicio cria os ficheiros, "tmp.txt" e "error.txt" e um FIFO com o nome "Pipeline". O ficheiro "tmp.txt" é um ficheiro temporário onde vai ser guardado o estado final (com o resultado dos comandos)

que depois será copiado para o ficheiro notebook que está a ser processado. O ficheiro "error.txt" é usado para redirecionar os todos os erros que ocorrerem durante a execução do programa. O fifo "Pipeline" será explicado mais à frente pelo que é usado por funções auxiliares definidas nos ficheiros helpers.h e helpers.c.

São feitos redirecionamentos com a função *dup2* do *stdin* para ficheiro que se recebe como argumento, e do stderror para o ficheiro "error.txt".

A nossa main.c começa a ler cada linha do ficheiro que recebou como argumento, guarda-a num buffer e envia-a para uma função (função *analyse*) que testa se a linha é um comando, se é apenas texto, ou se é o inicio ou fim do resulado de um comando(caso o ficheiro a ser processado já tivesse sido processado anteriormente). Essa mesma linha é copiada para o ficheiro "tmp.txt" a não ser que seja um resultado que um comando de um processamento anterior.

Com o resultado da função *analyse* pode acontecer uma de quatro possibilidades:

- Caso seja apenas uma linha será apenas copiada para o ficheiro "tmp.txt";
- Caso seja uma linha que comece pela palavra $, é um comando e a linha que está guardada no buffer será então guardada num array de strings (do genero do argv, separadas por palavras) para que possa ser enviada dentro da função *execute* para ser executada.
- Caso a linha começe por $|, é um comando que receberá como input o último comando executado e também será guardada num array de strings, e enviada para a função *executeNumPipe*.
- Caso a linha comece por $n|, é um comando que receberá como input o n-ésimo comando a contar do atual, e esta linha também será guardada num array de strings e enviada para a função *executeNumPipe*.
- Caso o ficheiro a ser processado tenha sido reprocessado e tenha resultados de comandos ainda no ficheiro, assim que seja detetado o início de um resultado de um comando (»>) é levantada uma flag, todas as linhas serão ignoradas, ou seja não copiadas até que seja encontrado o final de um output («<), e a partir daí as linhas voltam a ser processadas.

Após cada execução de cada linha é testado se foi escrito algo para o ficheiro de erros, caso tal aconteça será impresso um aviso no stdout que ocorreu um erro e que deve consultar o ficheiro de erros. Logo a seguir todos os ficheiros criados anteriormente são apagados menos o ficheiro "error.txt" e é terminado o programa com um *return -1*. Caso todas as linhas sejam processadas sem erros ou sem receber sinal de termínio por parte do utilizador (^C), o conteudo do ficheiro "tmp.txt" é copiado para o ficheiro a ser processado. No fim todos os ficheiros dentro da diretoria criada são apagados e posteriormente a diretoria é também apagada, terminando o programa com um *return 0*.

Capítulo 3

helpers.h

Este ficheiro tem apenas as assinaturas das funções que são utilizadas na main.c e que estão definidas no ficheiro helpers.c assim como os includes e um define da main.c.

Capítulo 4

helpers.c

Este ficheiro tem todas as funções auxilares utilizadas e definidas por nós que são utilizadas no nosso projeto.

Vamos agora explicar o funcionamento de cada uma:

4.1 readln

```
1 ssize_t readln(int fildes, void *buffer, size_t nbyte);
```

Função responsavel por ler uma linha apartir do *file descriptor* recebida como argumento na variavél fildes, guarda o que lê no array buffer, e lê nbytes de cada vez. Esta função devolve o tamanho do buffer. Usamos esta função para ler as linhas do ficheiro que vai ser processado.

4.2 randomName

```
1 void randomName(char* dir);
```

Função responsável por preencher a string dir com 8 letras minúsculas aletórias. Usamos esta função para com a string dir criar uma diretoria com um nome aleatório e depois criarmos lá o ficheiros durante a execução do programa.

4.3 analyse

```
1 int analyse(char* buffer, ssize_t size);
```

Função que lê os primeiros caracteres da string buffer, e devolve um de cinco valores:
• -1 , caso buffer comece por $.
• -2 , caso buffer comece por $|.
• -3 , caso buffer comece por »> (início de um output de um comando).
• -4 , caso buffer comece por «< (fim de um output de um comando).
• 0 , caso seja apenas uma linha para ser copiada.
• >0 , caso buffer comece por $n| e devolve n como um inteiro.

4.4 gatherArg

```
1 size_t gatherArg(char* arg[], char* buffer, size_t size);
```

Função responsável por receber uma string (buffer) com tamanho size, e guarda no arg que é um array de strings as palavas da string buffer. Esta função devolve o número de strings que foram guardadas em arg. Usamos esta função para depois usar o emphexecvp nas funções emphexecute e emphexecuteNumPipe.

4.5 execute

```
1 void execute(char* arg[], ssize_t num, char* dir, int execs);
```

Função responsável por executar o comando que está guardado em arg, este arg tem o tamanho num, dir é a directoria onde estão a ser criados os ficheiros, e execs é o número da execução atual. Este inteiro será o nome do ficheiro com o output deste comando. Usamos esta função para executar o comando que está armazenado em arg, o output do comando será guardado em dois ficheiros, no ficheiro própria para cada execução e para o ficheiro "tmp.txt" que vai ser "construido" para no fim ter o resultado final do processamento do ficheiro a que está a ser processado.

4.6 executeNumPipe

```
1 void executeNumPipe(char* arg[], ssize_t num, char* dir, int execs, int
      numexec);
```

Função responsável por executar o comando que está guardado em arg, este arg tem o tamanho num, dir é a directoria onde estão a ser criados os ficheiros, e execs é o número da execução atual. Este inteiro será o nome do ficheiro com o output deste comando. E numexec, será utilizado para calcular que ficheiro será usado como input. O ficheiro a ser utilizado como input é cálculado da seguinte formula: (execs – numexec). Usamos esta função para executar o comando que está armazenado em arg, o output do comando será guardado em dois ficheiros, no ficheiro própria para cada execução e para o ficheiro "tmp.txt" que vai ser "construido" para no fim ter o resultado final do processamento do ficheiro a que está a ser processado. Esta função permite-nos usar como input algum dos comandos anteriormente executados.

Para comandos do tipo, $| basta chamar esta função com a variável numexec com o valor 1.

Para comandos do tipo, $n| basta chamar esta função com a variável numexec com o valor n.

4.7 printline

```
1 void printline(char* buffer, size_t n, char* dir);
```

Esta função imprime o buffer com tamanho n para o ficheiro "tmp.txt" que está na diretoria dir. Usamos esta função para imprimir tudo que não sejam comandos, sem mudar redirecionamento que já foi feito na main.c.

4.8 terminate

```
1 void terminate(int signum);
```

Esta função é chamada logo no início da execução do programa e deteta sinais enviados pelo utilizador para terminar o programa. Esta função escreve para o stderror uma mensagem que o programa foi terminado através de um sinal e o ficheiro .nb mantém se inalterado, e por fim apaga todos os ficheiros até agora criados pela execução do nosso programa.

Capítulo 5

Makefile

A nossa makefile tem 5 opções:

• make , esta opção executa as opção compile (compilar o ficheiros main.c e lib/helpers.c) e a opção run que executa o executável notebook com um dos exemplos que nós criamos.

• compile , esta opção compila os ficheiros ficheiros main.c lib/helpers.c com a opção -Wall e dá o nome de notebook ao executável.

• run , esta opção executa o executável notebook com um dos nossos exemplos example.nb.

• run_all , esta opção executa o executável notebook com todos os notebooks, desde o seu nome tenha como estrutura example*.nb.

• clean , esta opção copia todos os ficheiros que estejam na pasta example_base para a diretoria atual, esta opção serve para caso os exemplos que nos criamos voltem ao seu estado original.

Capítulo 6

Testes

6.1 Teste 1

Um dos exemplos base, exemple.nb.

```
This command lists all files in the current directory:
$ ls -la
Now we can order that files:
$| sort
And from that ordered list fetch the first one:
$| head -1
Now we will count the number of words from the second command:
$2| wc
```

Resultado da execução do example.nb.

```
This command lists all files in the current directory:
$ ls -la
>>>
total 144
drwxrwxr-x 6 johnny johnny 4096 Jun 2 22:25 .
drwxr-xr-x 16 johnny johnny 4096 Jun 2 21:32 ..
-rw-rw-r-- 1 johnny johnny 74687 Mai 28 23:09 enunciado-so-2017-18.pdf
-rw-rw-r-- 1 johnny johnny 275 Jun 2 22:03 example2.nb
drwxrwxr-x 2 johnny johnny 4096 Jun 1 19:28 example_base
-rw-rw-r-- 1 johnny johnny 229 Jun 2 22:03 example.nb
drwxrwxr-x 8 johnny johnny 4096 Jun 2 22:02 .git
-rw-rw-r-- 1 johnny johnny 451 Mai 29 21:56 .gitignore
drwxrwxr-x 2 johnny johnny 4096 Jun 2 22:25 gltcflbi
drwxrwxr-x 2 johnny johnny 4096 Mai 29 02:14 lib
-rw-rw-r-- 1 johnny johnny 2772 Jun 2 22:02 main.c
```

```
-rw-rw-r-- 1 johnny johnny 270 Jun 2 22:03 Makefile
-rwxrwxr-x 1 johnny johnny 18264 Jun 2 22:25 notebook
-rw-rw-r-- 1 johnny johnny   8 Mai 28 23:09 README.md
<<<
Now we can order that files:
$| sort
>>>
drwxrwxr-x 2 johnny johnny 4096 Jun 1 19:28 example_base
drwxrwxr-x 2 johnny johnny 4096 Jun 2 22:25 gltcflbi
drwxrwxr-x 2 johnny johnny 4096 Mai 29 02:14 lib
drwxrwxr-x 6 johnny johnny 4096 Jun 2 22:25 .
drwxrwxr-x 8 johnny johnny 4096 Jun 2 22:02 .git
drwxr-xr-x 16 johnny johnny 4096 Jun 2 21:32 ..
-rw-rw-r-- 1 johnny johnny 229 Jun 2 22:03 example.nb
-rw-rw-r-- 1 johnny johnny 270 Jun 2 22:03 Makefile
-rw-rw-r-- 1 johnny johnny 275 Jun 2 22:03 example2.nb
-rw-rw-r-- 1 johnny johnny 2772 Jun 2 22:02 main.c
-rw-rw-r-- 1 johnny johnny 451 Mai 29 21:56 .gitignore
-rw-rw-r-- 1 johnny johnny 74687 Mai 28 23:09 enunciado-so-2017-18.pdf
-rw-rw-r-- 1 johnny johnny   8 Mai 28 23:09 README.md
-rwxrwxr-x 1 johnny johnny 18264 Jun 2 22:25 notebook
total 144
<<<
And from that ordered list fetch the first one:
$| head -1
>>>
drwxrwxr-x 2 johnny johnny 4096 Jun 1 19:28 example_base
<<<
Now we will count the number of words from the second command:
$2| wc
>>>
     15     128     798
<<<
```

6.2 Teste 1.1

Usamos o ficheiro que ja foi precessado e alteramos um dos comandos. E agora, iremos reprocessá-lo.

```
This command lists all files in the current directory:
$ ls
>>>
total 144
drwxrwxr-x 6 johnny johnny 4096 Jun 2 22:32 .
drwxr-xr-x 16 johnny johnny 4096 Jun 2 21:32 ..
-rw-rw-r-- 1 johnny johnny 74687 Mai 28 23:09 enunciado-so-2017-18.pdf
drwxrwxr-x 2 johnny johnny 4096 Jun 2 22:32 eokzvxno
```

```
-rw-rw-r-- 1 johnny johnny 275 Jun 2 22:30 example2.nb
drwxrwxr-x 2 johnny johnny 4096 Jun 1 19:28 example_base
-rw-rw-r-- 1 johnny johnny 229 Jun 2 22:30 example.nb
drwxrwxr-x 8 johnny johnny 4096 Jun 2 22:02 .git
-rw-rw-r-- 1 johnny johnny 451 Mai 29 21:56 .gitignore
drwxrwxr-x 2 johnny johnny 4096 Mai 29 02:14 lib
-rw-rw-r-- 1 johnny johnny 2772 Jun 2 22:02 main.c
-rw-rw-r-- 1 johnny johnny 270 Jun 2 22:03 Makefile
-rwxrwxr-x 1 johnny johnny 18264 Jun 2 22:25 notebook
-rw-rw-r-- 1 johnny johnny   8 Mai 28 23:09 README.md
<<<
Now we can order that files:
$| sort
>>>
drwxrwxr-x 2 johnny johnny 4096 Jun 1 19:28 example_base
drwxrwxr-x 2 johnny johnny 4096 Jun 2 22:32 eokzvxno
drwxrwxr-x 2 johnny johnny 4096 Mai 29 02:14 lib
drwxrwxr-x 6 johnny johnny 4096 Jun 2 22:32 .
drwxrwxr-x 8 johnny johnny 4096 Jun 2 22:02 .git
drwxr-xr-x 16 johnny johnny 4096 Jun 2 21:32 ..
-rw-rw-r-- 1 johnny johnny 229 Jun 2 22:30 example.nb
-rw-rw-r-- 1 johnny johnny 270 Jun 2 22:03 Makefile
-rw-rw-r-- 1 johnny johnny 275 Jun 2 22:30 example2.nb
-rw-rw-r-- 1 johnny johnny 2772 Jun 2 22:02 main.c
-rw-rw-r-- 1 johnny johnny 451 Mai 29 21:56 .gitignore
-rw-rw-r-- 1 johnny johnny 74687 Mai 28 23:09 enunciado-so-2017-18.pdf
-rw-rw-r-- 1 johnny johnny   8 Mai 28 23:09 README.md
-rwxrwxr-x 1 johnny johnny 18264 Jun 2 22:25 notebook
total 144
<<<
And from that ordered list fetch the first one:
$| head -1
>>>
drwxrwxr-x 2 johnny johnny 4096 Jun 1 19:28 example_base
<<<
Now we will count the number of words from the second command:
$2| wc
>>>
     15    128    798
<<<
```

Resultado de reprocessar um ficheiro com um comando diferente.

```
This command lists all files in the current directory:
$ ls
>>>
enunciado-so-2017-18.pdf
example2.nb
example_base
```

```
example.nb
hiszjmiy
lib
main.c
Makefile
notebook
README.md
<<<
Now we can order that files:
$| sort
>>>
enunciado-so-2017-18.pdf
example2.nb
example_base
example.nb
hiszjmiy
lib
main.c
Makefile
notebook
README.md
<<<
And from that ordered list fetch the first one:
$| head -1
>>>
enunciado-so-2017-18.pdf
<<<
Now we will count the number of words from the second command:
$2| wc
>>>
     10     10    109
<<<
```

6.3 Teste 2

Vamos processar um dos exemplos base, exemple2.nb.

```
This command will display the current active processes:
$ ps
Now we can select the lines that the word notebook appeared:
$| grep -w notebook
Lets check the working directory now:
$ pwd
The number of lines that contained the word notebook after the second
    command:
$2| wc -l
```

Resultado do processamento do exemple2.np.

```
This command will display the current active processes:
$ ps
>>>
  PID TTY          TIME CMD
18367 pts/2    00:00:00 bash
18689 pts/2    00:00:00 make
18699 pts/2    00:00:00 sh
18700 pts/2    00:00:00 find
18721 pts/2    00:00:00 notebook
18724 pts/2    00:00:00 notebook
18725 pts/2    00:00:00 ps
<<<
Now we can select the lines that the word notebook appeared:
$| grep -w notebook
>>>
18721 pts/2    00:00:00 notebook
18724 pts/2    00:00:00 notebook
<<<
Lets check the working directory now:
$ pwd
>>>
/home/johnny/Desktop/so1718
<<<
The number of lines that contained the word notebook after the second
    command:
$2| wc -l
>>>
2
<<<
```

Capítulo 7

Conclusão

Neste trabalho, pusemos em prática algumas das funcinalidades de um vasto leque de ferramentas do sistema opetarivo. De forma geral, abordamos as temáticas essenciais, que passa desde acesso a ficheiros, gestão de processos, execução de programas, redirecionamento de descritores de ficheiros, pipes anónimos, pipes com nome e sinais. Conseguimos com sucesso, analizar e resolver os sub-problemas inerentes as tarefas propostas. Este trabalho estava dividido em dois tipos de funcionalidades (basicas e avançadas), visto ser algo de extrema importancia para um profíssional da área, não ficamos pelas funcionalidades básicas. Com este trabalho ficamos a compreender melhor a estrutura e principais funções de um sistema operativo, bem como um modelo de programação subjacente.