

Relatório do projeto de Sistemas Operativos

João Pimentel A80874 Jaime Leite A80757
Bruno Veloso A78352

Junho 2018

Universidade do Minho
Departamento de Informática
Sistemas Operativos
Grupo 1

Mestrado Integrado em Engenharia Informática



Conteúdo

1	Introdução	3
2	Funcionalidades	4
2.1	Execução de programas	4
2.2	Reprocessamento de um notebook	5
2.3	Deteção de erros e interrupção de execução	5
2.4	Compilação e Execução	6
3	Conclusão	7
4	Bibliografia	7

1 Introdução

Este projeto foi realizado no âmbito da unidade curricular de *Sistemas Operativos*, na qual era pretendido construir um sistema de processamento de *notebooks*, ou seja, recebendo um ficheiro com texto e comandos a executar, o seu *output* será direcionado para zonas específicas dentro do mesmo ficheiro.

De modo a implementar esta tarefa, foram utilizadas várias técnicas aprendidas nas aulas práticas, bem como conhecimentos teóricos relativos à cadeira, dando especial atenção ao uso de system calls como *pipes*, *forks*, *execs*, etc.

É notório que o uso das mesmas permite a realização de diversas tarefas anteriormente complexas e dispendiosas em termos de complexidade, com pouco esforço.

Neste relatório serão abordadas todas as formas de resolução dos problemas encontrados ao longo da realização do projeto.

2 Funcionalidades

Foram bastante simples de realizar, já que bastou aplicar métodos muito semelhantes aos utilizados na realização dos guiões laboratoriais das aulas. Sendo quase todas baseadas em ler conteúdo de um ficheiro, escrever noutro e executar comandos, direcionando esse conteúdo.

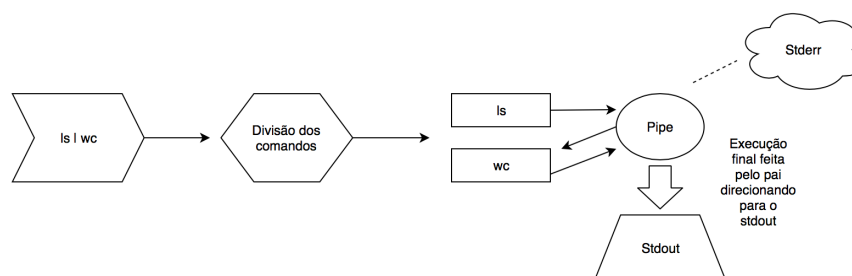
2.1 Execução de programas

Sabendo que apenas as linhas onde existe um carácter ‘\$’, no início da mesma, devem ser interpretadas como comandos possíveis a executar, a ideia passou por ler, linha a linha, usando uma função *getline*, o conteúdo do ficheiro.

Tendo em conta a possível ocorrência de erros, foi decidido pelos elementos do grupo que seriam usados dois ficheiros auxiliares, sendo um para direcionar os erros presentes no *stderr* (*cerr.log*) e outro onde seria escrita a parte útil do ficheiro do *input*, tal como a execução dos comandos necessários (*cout.log*).

De notar que o *output* da execução de cada comando está destacado pela presença de >>>, logo a seguir ao comando, e <<<, depois da execução do mesmo.

Como pode ocorrer encadeamento de comandos, o grupo optou por concatenar os comandos a executar, passando-os a uma função, em vez de guardar os offsets dos pipes onde poderiam ser executados num *array*, já que assim o método fica mais fácil de compreender. Esta função, que dado um comando do tipo “*ls | wc*”, é executado o “*ls*” por um “filho”, dentro de um *pipe*, sendo depois o seu *output* direcionado para o *stdout*, de modo a que o “pai” possa utilizar o mesmo como seu *input* de “*wc*”, devolvendo o resultado encadeado pretendido. O seguinte diagrama demonstra este processo:



Depois de executados todos os comandos, se não existir conteúdo no ficheiro *cerr.log*, ou seja, não existirem erros durante o processamento do *notebook*, o conteúdo de *cout.log* é transferido para o, já mencionado, ficheiro dado como *input*. Sendo que, no final, ambos os ficheiros auxiliares são apagados.

2.2 Reprocessamento de um notebook

Tendo em conta que é possível reutilizar um *notebook* já processado, fazendo alterações aos comandos a executar, foi necessário encontrar uma forma de estipular quando existia conteúdo não útil a escrever no ficheiro auxiliar. Dito isto, sabendo que este conteúdo, variável, é o existente entre `>>>` e `<<<`, bastou ter uma variável que indicasse se estava a ser analisada uma destas zonas, ignorando, por completo, o seu conteúdo, se fosse o caso.

```
1  Int nl = 1;
2  while(1) {
3      size_t n = readline(fd_1, buf, sizeof(buf));
4      if (n <= 0) break;
5
6      if (buf[0] == '>' && buf[1] == '>' && buf[2] == '>'
7          ) nl = 0;
8      if (buf[0] == '<' && buf[1] == '<' && buf[2] == '<')
9          {nl = 1; continue;}
10     if (nl == 0) continue;
11     ...
12 }
```

Assim, tornou-se simples reprocessar um *notebook* que já possuisse resultados aos seus comandos.

2.3 Detecção de erros e interrupção de execução

Como mencionado atrás, existem dois ficheiros auxiliares, sendo que um deles contém todos os erros que vão ocorrendo ao longo da execução do programa, quer isto dizer que é fácil de descobrir se o processamento deve ser encerrado. Assim, basta ler do ficheiro *cerr.log*, usando a *system call* **read**, em que, caso devolva o valor *zero*, o processamento pode prosseguir. Além disso, é possível terminar o programa, enviando um sinal de *SIGINT* (através do uso de *CTRL C*). Como este sinal deve terminar a execução em curso, devendo o *notebook* permanecer inalterado, foi necessário apagar os ficheiros temporários e, por precaução, escrever algo no ficheiro de erro, antes de apagar, para indicar a não finalização do programa.

Em seguida tem-se o excerto de código para ver se o ficheiro *cerr.log* se encontra vazio:

```
1  int err = open("cerr.log", O_RDWR | O_CREAT | O_APPEND, 0600);
2  if (-1 == err) {
3      perror("opening_cerr.log");
4      return 255;
5  }
6  int ler = read(err, buffer, sizeof(buffer));
7  close(err);
8
9  if (ler == 0) {
10     int x = cpy_file("cout.log", argv[1]);
11     if (x != 0) {
12         perror("Update_failed.");
13         return 266;
14     }
15 }
```

15 }

2.4 Compilação e Execução

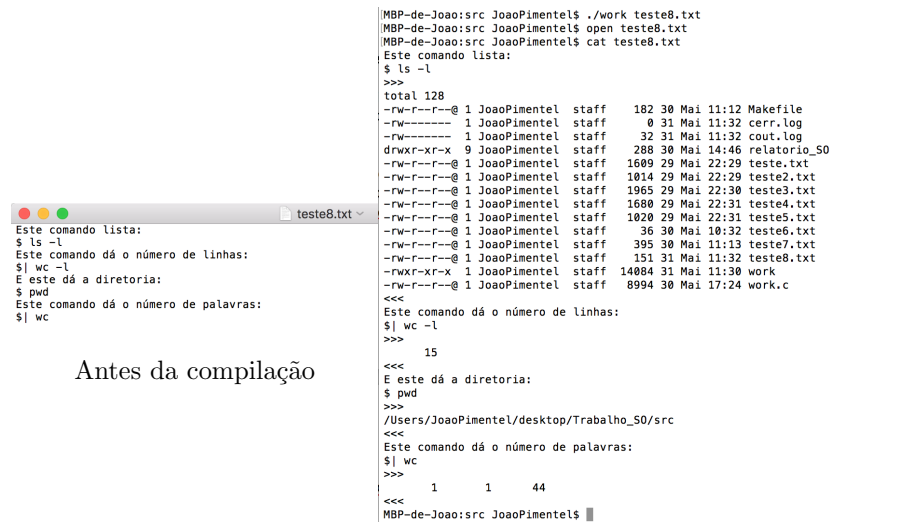
De modo a compilar o projeto na sua totalidade, apenas é necessário correr na *bash* o seguinte comando, estando na diretoria certa:

```
1 >> make compile
```

Já para executar, como é necessário um *notebook* como argumento, este tem que ser passado à função, tendo-se:

```
1 >> ./work notebook
```

Sendo que este *notebook* é o ficheiro em questão, existente na diretoria do trabalho. Por exemplo, dando o seguinte ficheiro como input, tem-se:



```
MBP-de-Joao:src JoaoPimentel$ ./work teste8.txt
MBP-de-Joao:src JoaoPimentel$ open teste8.txt
MBP-de-Joao:src JoaoPimentel$ cat teste8.txt
Este comando lista:
$ ls -l
>>>
total 128
-rw-r--r--@ 1 JoaoPimentel  staff   182 30 Mai 11:12 Makefile
-rw-----  1 JoaoPimentel  staff    0 31 Mai 11:32 cerr.log
-rw-----  1 JoaoPimentel  staff    32 31 Mai 11:32 cout.log
drwxr-xr-x  9 JoaoPimentel  staff  288 30 Mai 14:46 relatorio_S0
-rw-r--r--@ 1 JoaoPimentel  staff  1609 29 Mai 22:29 teste.txt
-rw-r--r--@ 1 JoaoPimentel  staff  1014 29 Mai 22:29 teste2.txt
-rw-r--r--@ 1 JoaoPimentel  staff  1965 29 Mai 22:30 teste3.txt
-rw-r--r--@ 1 JoaoPimentel  staff  1680 29 Mai 22:31 teste4.txt
-rw-r--r--@ 1 JoaoPimentel  staff  1020 29 Mai 22:31 teste5.txt
-rw-r--r--@ 1 JoaoPimentel  staff    36 30 Mai 10:32 teste6.txt
-rw-r--r--@ 1 JoaoPimentel  staff   395 30 Mai 11:13 teste7.txt
-rw-r--r--@ 1 JoaoPimentel  staff   151 31 Mai 11:32 teste8.txt
-rwxr-xr-x  1 JoaoPimentel  staff 14084 31 Mai 11:30 work
-rw-r--r--@ 1 JoaoPimentel  staff   8994 30 Mai 17:24 work.c
<<<
Este comando dá o número de linhas:
$| wc -l
>>>
15
<<<
E este dá a diretoria:
$ pwd
>>>
/Users/JoaoPimentel/desktop/Trabalho_S0/src
<<<
Este comando dá o número de palavras:
$| wc
>>>
1      1      44
<<<
MBP-de-Joao:src JoaoPimentel$
```

Antes da compilação

Depois da compilação

Para o caso de ser necessário apagar o executável do projeto, basta escrever:

```
1 >> make clean
```

3 Conclusão

Em suma, o projeto foi realizado com sucesso, apesar de terem sido encontrados diversos obstáculos que ajudaram a compreender realmente a utilidade de utilização de processos e necessidade de os utilizar com precaução.

Como foi necessária uma abordagem prática e por conta própria, todos os elementos do grupo foram capazes de compreender os panoramas lecionadas nesta *Unidade Curricular*, implementando as funcionalidades mencionadas ao longo do relatório.

No entanto, houve aspetos e funcionalidades que não foram implementadas, tal como a execução paralela e o acesso a resultados de comandos anteriores arbitrários, que podia ser facilmente obtido através da utilização de um *array* que contivesse todos os *offsets* dos *pipes* utilizados, por exemplo.

De destacar que este trabalho permitiu aumentar as capacidades de resolução de problemas, uma vez que obriga ao uso da “massa cinzenta” na sua totalidade, não sendo proveitoso utilizar métodos genéricos que em nada valorizam o espírito crítico e de desenvolvedor que os alunos procuram adquirir.

Para finalizar, após a conclusão do trabalho, todos os elementos deste grupo saem com uma perceção mais realista e clara da fragilidade na utilização de processos quando estes não são bem geridos. Além disso, foi claro que é de extrema importância a forma como se interage com ficheiros abertos e que o método de comunicação com *pipes* tem uma importância tremenda em termos de resultado esperado.

4 Bibliografia

- A. S. Tanenbaum, Modern Operating Systems, 2nd edition, Prentice Hall, 2001
- Guiões Sistemas Operativos, MIEI - UMinho, 2017/18
- Slides Sistemas Operativos, MIEI - UMinho, 2017/18

Mestrado Integrado em Engenharia Informática

