Universidade do Minho
Escola de Engenharia

Sistemas Operativos (2º ano de MIEI)

Trabalho Prático

Relatório de Desenvolvimento

Grupo 47

Cesário Miguel Pereira Perneta (A73883)

Luís Miguel Bravo Ferraz (A70824)

Pedro Miguel Lopes Pereira (A70951)

2 de Junho de 2018

Resumo

Este trabalho foi-nos proposto no âmbito da disciplina de Sistemas Operativos e tem como objetivo a construção de um sistema para processamento de *notebooks*.

No processamento, pressupõe-se que as linhas começadas por $ sejam interpretas como comandos a serem executados, sendo o resultado desses comandos inserido imediatamente a seguir, delimitado por >>>e <<<.

As linhas começadas por $ | executam comandos que têm como *stdin* o resultado do comando anterior.

Conteúdo

Capítulo 1

Introducão

Neste trabalho foi-nos proposta a construção de um sistema para processamento de *notebooks*, com vista em aumentar a nossa experiência no uso das matérias lecionadas ao longo deste semestre. O mesmo é composto por 3 diferentes etapas:

- Interpretação dos comandos;
- Execução dos comandos;
- Acrescentar o resultado da execução ao ficheiro;

Foram impostas no enunciado as seguintes funcionalidades básicas:

- Execução de programas;
- Reprocessamento de um Notebook;
- Deteção de erros e interrupção da execução;

Bem como as seguintes funcionalidades avançadas:

- Acesso a resultados de comandos anteriores arbitrários;
- Execução de conjunto de comandos;

Esperamos com este relatório explicar e clarificar todas as decisões tomadas ao longo do trabalho bem como alcançar os objetivos traçados.

1.1 Implementação

O programa foi desenvolvido para ambientes Unix e foi desenvolvido na linguagem C. Ao longo deste capítulo serão descritas as principais componentes do programa bem como as suas funcionalidades

1.1.1 Estratégia Inicial

A estratégia inicial passou por guardar em ficheiros individuais o output gerado pelo comando de cada linha e o resultado final num ficheiro temporário que vai sendo completado aos poucos. Consequentemente, foi logo decidido que iria ser criado um ciclo com um readline em que o pai, sempre que encontrasse um comando, o enviasse através de um pipe, de modo a ser tratado pelo filho. Será também o pai a ter a função de escrever no ficheiro temporário geral. A partir deste ponto, começam então a ser tomadas decisões de forma a implementar as funcionalidades desejadas.

1.1.2 Principais Estrutura de dados e Variáveis

Neste ponto vamos então enumerar as estruturas de dados contidas no programa:

- Ficheiros
  - fd : File descriptor do notebook;
  - fd_err : File descriptor do ficheiro temporário de erros (redirecionado do stderr);
  - fd_final : File descriptor do ficheiro temporário que vai conter o resultado final;
  - fd_ant : File descriptor do ficheiro com o output gerado necessário para um comando com '—';
  - fd_filho : File descriptor do ficheiro gerado pelo filho;
  - fd_pf : File descriptor do ficheiro filho no processo pai;
- Buffers
  - buffer : contem a linha lida em cada iteraçao;
  - buffer_filho : contem a linha enviada pelo pai no filho;
  - buffer_ant : contem a linha lida dos ficheiros necessários para o execução da linha atual;
- Pipes
  - pd[2] : pipe que permite a comunicação entre o processo pai e o processo filho;
  - pd_ant[2] : pipe que permite a ao processo filho, comunicar com um filho gerado por si que lê um ficheiro anterior com o output necessário para um comando;
- Variáveis Globais
  - num_operacoes, variavel iniciada a um que conta o numero de processos criados pelo pai e utilizada para nomear os ficheiros com os outputs e aceder aos mesmos quando necessario.
  - fd, fd_err, fd_final, explicados no ponto "Ficheiros".

1.2 Funcionalidades Básicas

1.2.1 Execução de programas

Inicialmente, como já foi dito anteriormente, é utilizado um ciclo inicial que, utilizando o readline, vai ler linha a linha o notebook e vai reescrever tudo num ficheiro temporário. Posto isto, será então feita uma seleção onde distinguimos os comentários dos comandos (linhas começadas pelo carater '$').

Se a linha não começar por '$' então é logo escrita pelo pai no ficheiro temporário, caso contrário, o pai cria um filho e envia-lhe a linha com a ajuda de um pipe. É o filho que tem a função de tokenizar a linha por espaços de modo a conseguir construir um array de strings que será útil mais a frente na execução dos comandos, pois irá conter tanto o comando como os seus argumentos.

No fim da tokenização, vamos avaliar a primeira posiçao do array gerado. Se a mesma for só composta por um '$' então podemos logo executar o comando, caso contrário vamos ter de proceder à leitura do output do comando anterior. Para tal, dentro do filho criamos um novo filho (chamemos de filho2) que irá realizar a leitura do ficheiro que contem o output anterior. O filho2, à medida que lê, com a ajuda de um novo pipe, envia as linhas do ficheiro para o filho, que após a realização da leitura, executa o comando tendo como input o que recebe do filho2 guardando o resultado no ficheiro respetivo.

Posteriormente, o pai vai ler o ficheiro gerado pelo filho e irá escrever entre as linhas também colocadas por ele e ≪«"o resultado do exec do filho no ficheiro temporário final. No final, apaga-se o ficheiro dado como argumento, e faz-se o rename do ficheiro temporário para o nome desse mesmo ficheiro que está guardado em argv[1].

1.2.2 Reprocessamento do Notebook

Para esta funcionalidade, ao lermos um notebook previamente processado, foi necessário criar uma estratégia que nos permitisse ignorar os outputs presentes no mesmo. Foi então criada uma variavél designada flag que é inicializada com o valor 0. Sempre que a linha lida é '»¿' ela toma o valor de 1, sendo que só volta ao valor de zero após a leitura da linha '«¡'. O pretendido é o pai só verificar as linhas quando a flag está a 0, pois, desta forma, os outputs são então ignorados. Um ficheiro temporário é novamente criado, repetindo-se o ciclo descrito na section anterior.

1.2.3 Deteção de Erros e Interrupção da Execução

Relativamente à deteção de erros, são feitos 3 tipos:

- Ficheiros abrem/criam sem problemas ao ver o valor do file descriptor associado;
- Filhos acabam sem problemas, ao vermos o status com o WEXITSTATUS;
- Criamos um ficheiro temporário designado "std_err.txt"para o qual é redirecionado tudo que seria escrito no stderr. Se o WEXITSTATUS for igual a zero (o filho acabou bem) verificamos então se o "std_err.txt"está vazio para nos assegurarmos que não ocurreram erros durante o processo.

Relativamente à interrupção da execução é criado um signal para o SIGINT, sendo que será invocada a funçao remove_all_fail() que será explicada mais frente.

1.3 Funcionalidades Avançadas

1.3.1 Acesso a Comandos Anteriores Arbitrários

Para a realização desta funcionalidade, foi necessário, após testar se o primeiro elemento do array de strings resultante da tokenizaçao, verificar se ele continha algo mais que o "$". Nos casos em que tal ocorria, criou-se um ciclo que utilizava a funçao isdigit() para concatenar os carateres que retornavam True a esta função numa string, sendo que depois se fazia o parse para int da mesma. desta forma é nos possível abrir o ficheiro relativo ao N comando anterior, subtraindo à variavel global num_operaçoes esse valor.

1.4 Funções Auxiliares

1.4.1 remove_all_right()

Esta função recebe como argumento o nome do notebooke é utilizada quando o processo ocorre sem erros. Em consequência disso, faz-se os closes dos ficheiros iniciais, de erro e temporário, apaga-se o notebook e os ficheiros relativos aos outputs gerados e renomeia-se o nome do ficheiro temporário para o nome do notebook.

1.4.2 remove_all_fail()

Esta função é utilizada quando algo corre mal, removendo todos os ficheiros criados mantendo intacto o notebook.

1.5 Resultados

Notebook inicial :

```
Este comando lista os ficheiros:
\$ ls -la
Agora podemos ordenar estes ficheiros:
\$ echo "sistemas operativos, trabalho pratico"
Ordena
\$2| sort
```

Resultado:

```
 Este comando lista os ficheiros:
\$ ls -la
>>>
total 52
drwxrwxr-x 2 pedro pedro  4096 Jun  2 23:41 .
drwxrwxr-x 3 pedro pedro  4096 Jun  2 21:23 ..
-rw-rw-r-- 1 pedro pedro     0 Jun  2 23:41 ficheiro1.txt
-rwxrwxr-x 1 pedro pedro 14424 Jun  2 23:30 notebook
-rw-rw-r-- 1 pedro pedro 12234 Jun  2 23:23 notebook.c
-rw-rw-r-- 1 pedro pedro   399 Mai 22 17:22 readline.c
-rw-rw-r-- 1 pedro pedro   138 Mai 22 17:12 readline.h
-rw-rw-r-- 1 pedro pedro     0 Jun  2 23:41 std_err.txt
-rw-rw-r-- 1 pedro pedro    46 Jun  2 23:41 temporario.txt
-rw-rw-r-- 1 pedro pedro   144 Jun  2 23:40 teste3.txt
<<<
Agora podemos ordenar estes ficheiros:
\$ echo "sistemas operativos, trabalho pratico"
>>>
"sistemas operativos, trabalho pratico"
<<<
```

```
Ordena
\$2| sort
>>>
drwxrwxr-x 2 pedro pedro  4096 Jun  2 23:41 .
drwxrwxr-x 3 pedro pedro  4096 Jun  2 21:23 ..
-rw-rw-r-- 1 pedro pedro     0 Jun  2 23:41 ficheiro1.txt
-rw-rw-r-- 1 pedro pedro     0 Jun  2 23:41 std_err.txt
-rw-rw-r-- 1 pedro pedro 12234 Jun  2 23:23 notebook.c
-rw-rw-r-- 1 pedro pedro   138 Mai 22 17:12 readline.h
-rw-rw-r-- 1 pedro pedro   144 Jun  2 23:40 teste3.txt
-rw-rw-r-- 1 pedro pedro   399 Mai 22 17:22 readline.c
-rw-rw-r-- 1 pedro pedro    46 Jun  2 23:41 temporario.txt
-rwxrwxr-x 1 pedro pedro 14424 Jun  2 23:30 notebook
total 52
<<<
```

Capítulo 2

Conclusão

Este trabalho prático consistiu em aplicar os conhecimentos adquiridos ao longo do semestre num problema do dia - a -dia, que neste caso passa pelo processamento de um Notebook.

Perante todas as dificuldades obtidas durante o trabalho, pensamos que os resultados obtidos são satisfatórios. Muito dos problemas do grupo passaram por estar relacionados com as estratégias a utilizar para superar os mesmos. Como trabalho futuro, fica por completar a última funcionalidade avançada, mas por falta de tempo por termos ficado presos a alguns problemas não nos foi possivel realizar.

Por fim, sentimos que este trabalho serviu para consolidar bem os conhecimentos da disciplina, o que nos vai ser bastante útil na realização do teste da mesma.