

UNIVERSIDADE DO MINHO

MESTRADO INTEGRADO EM ENGENHARIA
INFORMÁTICA

SISTEMAS OPERATIVOS

Processamento de Notebooks

2 de junho de 2018

Membros do Grupo 66

Bruno Martins A80410 Leonardo Neri A80056 Márcio
Sousa A82400

1 Introdução

Este projeto teve como objetivo criar um programa na linguagem de programação C que permitisse o processamento de *notebooks*. Um *notebook* é um ficheiro de texto que contém comentários e comandos e que depois do seu processamento também contém o resultado dos comandos. O conteúdo do ficheiro, como referido anteriormente, contém um comando, que começa obrigatoriamente com o símbolo \$, e o seu output está delimitado entre > > output < < < . O *notebook* também poderá ter nalguns casos um número logo após o símbolo \$ referente ao output do *n* comando anterior que deverá ser um argumento do próximo comando a ser executado.

2 Estruturas de Dados

2.1 COMMAND

Para o armazenamento do conteúdo dos *notebooks* foi criada uma estrutura *COMMAND* que contém quatro campos. Uma *String input* que contém o comando a executar, uma *String output* que contém o output do comando, uma *String comment* que contém o comentário que está imediatamente antes de comando, e um *inteiro* que se refere ao número do output anterior que deve ser utilizado como *input* no comando atual.

2.2 LIST

Para assegurar uma conexão entre todas as células *COMMAND* foi utilizada uma lista ligada. Esta lista está ordenada tendo em conta a organização do *notebook* a ser processado.

3 Execução

3.1 Leitura e parsing do ficheiro

A execução do programa começa pela abertura do ficheiro passado como argumento com a *system call open*, de seguida é lido o texto que lá está dentro, linha a linha, um caracter de cada vez usando o *read*. Logo após ser finalizada a leitura da linha, o conteúdo dela é processada por uma função de

parsing que verifica se é um comando, um comentário, um output e também o *inteiro* a seguir ao *\$* e insere no campo designado da estrutura *COMMAND*.

3.2 Execução de Comandos

No que toca à execução dos comandos propriamente dita, o programa cria dois pipes, para que na criação de um processo filho que executa os comandos, este possa comunicar com o processo pai de forma a redirecionar inputs e outputs. Seguidamente é feito um *fork* que cria um processo filho e onde os comandos são executados usando *execvp*. Os outputs do comando são redirecionados para um pipe anteriormente criado e guardados na respetiva posição da estrutura. Na eventualidade de um comando necessitar de um output que não seja o imediatamente anterior a ele, é encontrado esse output e escrito no *file descriptor* de onde o processo vai ler os argumentos para execução. Finalmente no fim de todo este processo os resultados são guardados na estrutura. No caso de ser necessário um output de um comando que não seja o imediatamente anterior, é percorrida a estrutura *LIST* até ele ser encontrado e é escrito no pipe de onde o comando vai ler antes de executar.

3.3 Escrita no ficheiro

Visto que a estrutura em que está guardado o ficheiro e os outputs dos comandos dentro dele está ordenada segundo a organização do ficheiro, a escrita no ficheiro é feita utilizando a função *creat*, criando assim um ficheiro novo igual ao inicialmente passado mas agora sem conteúdo, são percorridos todos os elementos da estrutura *LIST* criada e escritas todas as componentes dessa estrutura.

4 Controlo de Erros

Quanto ao controlo de erros neste programa têm de ser considerados alguns cenários que possam ocorrer. Numa primeira instância pode não ser passado nenhum argumento ao programa bem como mais que um, nesse caso o programa é interrompido. Outro caso possível é, a executar um comando cujo input seja o output de um *N* comando anterior, o número seja maior que o número de comandos já executados logo não pode ser executado esse comando tendo o programa de terminar. Erros de leitura e/ou escrita em

ficheiros foram também tidos em conta bem como na alocação de memória para as estruturas. Na criação de pipes é verificado se estes são gerados sem erros e aquando da chamada da função *fork* também ela sofre controlo de erros.

5 Conclusão

Este projeto foi uma excelente oportunidade para aplicar conhecimentos relativos a Sistemas Operativos, tendo desta forma ficado mais vincado como e o porquê de um sistema operativo executar processos de forma eficiente. Quanto a melhoramentos possíveis ao programa acima descrito poderia ser melhorado através da execução concorrente de comandos. Esta arquitetura necessitaria no entanto de outro tipo de cuidados tal como a verificação de dependências de execução (um comando para executar necessitar do output de um comando que ainda não foi executado). Contudo, na opinião dos integrantes do grupo, os objetivos do trabalho foram atingidos pois todas as funcionalidades do programa executavam em conformidade com o requerido.