Trabalho prático nº1

Interpretador de comandos (shell) para Unix/Linux

Versão 6

030509107 André Simões Fernandes
090509136 João Sá Vinhas Gonçalves

Estrutura

Com o objectivo de implementar a versão 6 do trabalho prático, recorreu-se, tal como na primeira versão do trabalho, à construção de uma estrutura para guardar todos os tokens da linha de comandos. No entanto, uma vez que tinhamos de gravar vários comandos separados por *pipes*, dedicidimos construir uma lista duplamente ligada de estruturas do tipo:

```
struct command_info {
        char **arg;             // Lista de argumentos (terminada com NULL).
        char *infile;           // Fich. p/ onde redirecionar stdin ou NULL.
        char *outfile;          // Fich. p/ onde redirecionar stdout ou NULL.
        int background;         // 0 ou 1 (procedimento a executar em background).
        int fd_input[2];        // Descritores de leitura do pipdo actual.
        int fd_output[2];       // Descritores de escrita do actual.
        struct command_info * next;         // Aponta para a próxima estrutura.
        struct command_info * previous;     // Aponta para a estrutura anterior.
};

typedef struct command_info Command_Info_Pip;
```

cada uma das quais responsável por guardar a informação relativa a um comando.

Além da particularidade de ser uma lista ligada, por questões de facilidade de acesso às estruturas anteriores, deve notar-se que esta estrutura também fica responsável por guardar dois *arrays* de descritores. Estes *arrays* correspondem ao *pipes* de entrada e saída que um dado comando poderá ter de aceder.

Execução dos comandos

Por forma a fazer o *parsing* da linha de comandos e a construção da lista duplamente ligada recorreu-se a duas funções:

- Command_Info_Pip * parse_cmd_pip(char * cmd_line_without_pipes);
- Command_Info_Pip * parse_cmds_pip(char * cmd_line_with_pipes);

A primeira, destina-se a fazer o parsing de forma similar à função parse_cmd da primeira versão do trabalho. A segunda, chama a primeira, recursivamente, e liga as estruturas convenientemente.

Por forma a criar todos os *pipes* necessários à execução dos comandos criou-se uma função à parte, que recebe como primeiro argumento o apontador para a primeira estrutura:

- int create_pipes(Command_Info_Pip * cmd_info);

Para facilitar o fecho dos *pipes* utiliza-se:

- int close_pipes(Command_Info_Pip * cmd_info);

Finalmente, e mais importante, por forma a executar os comandos usa-se a função:

- exec_multiple(Command_Info_Pip * first_cmd_info)

esta função lança tantos processos filhos quanto o número de estruturas ligadas ao first_cmd_info. Garantindo que os filhos, fazem redireccionamento apropriado. Seja para ficheiros ou para os *pipes* devidos.