Implementação de Comandos com Pipes

Pedro Chambino e Eva Marques

2010-04-17

1 Primeiras Tentativas

Inicialmente tentamos desenvolver uma solução recursiva em que um comando verificava se tinha um próximo comando e de seguida criava um pipe para ligar a sua saída ao pipe e passava este na chamada recursiva para o próximo comando que depois o ligava à sua entrada e depois voltava a verificar se tinha outro próximo comando até não ter um próximo comando.

No entanto, esta solução levou a alguns problemas e devido à dificuldade de fazer o debugging acabamos por desenvolver uma solução iterativa sugerida pelo Professor José Alves da Silva.

2 Solução Desenvolvida na Versão Final

Para podermos executar vários comandos ligados por pipes:

- modificamos a estrutura do Command_Info de forma a ter um elemento que aponta para o próximo comando a executar ligado por um pipe;
- criamos uma função exec_pipe() para executar o comando com pipes em que segue os seguintes passos:
  - conta o numero de comandos a executar;
  - inicializa um array terminado por 0 (zero) para guardar as PIDs do processos filhos, que é devolvido pela função;
  - inicializa um array terminado por NULL para guardar os pipes. Os pipes são inicializados logo de seguida;
  - executa-se o primeiro comando numa função chamada exec_pipe_first_cmd(), que:
    - faz o fork() e retorna a PID do processo ou -1 em caso de erro;

    * o processo filho depois verifica se o comando é em background:
    * se sim: muda o grupo do processo para não receber os sinais da shell e se não redireccionar a entrada para um ficheiro então redirecciona para /dev/null;
    * se não: simplesmente verifica se tem de redireccionar a entrada para um ficheiro.
    * verifica se tem um pipe para redireccionar a saída:
    * se não: simplesmente verifica se tem de redireccionar a saída para um ficheiro. Isto permite executar comandos simples (sem pipes);
    * se sim: fecha a leitura do pipe. Se tiver um ficheiro para redireccionar a saída então também fecha a escrita no pipe, se não, redirecciona a saída para o pipe.
    * e finalmente executa o comando.
- depois num ciclo passando pelos próximos comandos estes são executados numa função chamada exec_pipe_next_cmd() semelhante à função exec_pipe_first_cmd():
    * mas que não verifica se o comando está em background, pois isso é verificado no primeiro comando;
    * começa por fechar a escrita no pipe anterior. Se tiver um ficheiro para redireccionar a entrada então também fecha a leitura do pipe anterior, se não, redirecciona a entrada para o pipe;
    * o redireccionamento da saída é feito como em exec_pipe_first_cmd(), mas para o pipe seguinte;
    * e finalmente executa o comando.
- por fim os pipes no processo pai (na shell) são fechados e o seu array libertado, retornando o array dos PIDs dos processos filhos.
- Em caso de erro nos fork() dos processos filhos, a funçao retorna NULL depois de fechar os pipes e libertar os arrays inicializados.