Implementação de um interpretador de comandos

Bárbara Boechat, Eduarda Goulart, Juliana Araújo

Universidade Federal de São João del-Rei, Abril 2018

1 Especificações do problema

Este trabalho tem o objetivo de implementar um interpretador de comandos (chamado no Unix/Linux de shell). Para isso, são disparados processos que se comunicam por meio de pipes. Ademais, são utilizados pipes e manipulação de processos para criar um par produtor/consumidor por envio de mensagens que trabalhará dentro da shell para "alimentar" programas disparados com dados, ou para receber dados de um programa.

2 Implementação

O interpretador foi implementado em linguagem C, e pode ser iniciado de duas formas:

- Sem argumentos de linha de comando (interativo): os comandos são lidos da entrada padrão, e no início de cada linha é escrito um prompt ($). Mensagens de erro e o resultado dos programas são exibidos na saída padrão.
- Com apenas um argumento de linha de comando: o parâmetro fornecido é interpretado como o nome de um arquivo de onde comandos serão lidos. Nesse caso, apenas o resultado dos comandos é exibido na saída padrão, sem a exibição de prompts nem o nome dos comandos executados.

Cada linha deve conter um ou mais comandos para ser(em) executado(s). Exemplo de execução:

$ ls -l
$ ls -laR => arquivo
$ wc -l <= arquivo
$ cat -n <= arquivo => arquivonumerado
$ cat -n <= arquivo | wc -l => numerodelinhas
$ cat -n <= arquivo | wc -l => numerodelinhas
$ fim

O tratamento da entrada foi feito utilizando a função strtok(), separando os comandos e guardando aqueles que de fato serão executados. Os demais

símbolos, ao serem encontrados, são relacionados a uma label. O interpretador deve estar protegido de todos os erros que pode ocorrer no novo comando, portanto é criado um novo processo para cada novo comando através da função fork(). Para executar o programa foi utilizada a chamada de sistema execvp(), de modo que o processo pai é sobrescrito pelo processo filho, disparado sempre que há um novo comando de entrada.

O símbolo "=>" indica que a saída padrão do processo é substituída pelo arquivo indicado. Para isso, utilizou-se a função freopen(), responsável por redirecionar o fluxo padrão (stdout) e então escrever no arquivo especificado. Já o símbolo "<=" indica que a entrada padrão do processo é substituída pelo arquivo indicado. Nesse caso, também utilizou-se a função freopen() para substituir a entrada padrão (stdin) pelo arquivo.

Ao haver mais de um programa a ser executado na linha de comando eles serão encadeados por pipes (|) , indicando que a saída do programa à esquerda deve ser associada à entrada do programa à direita. Para criar o pipe, foi utilizada a chamada do sistema pipe(). Essa chamada necessita de um único argumento, que é um array de dois inteiros, que irá conter dois novos descritores de arquivos que serão usadas para o pipeline. O primeiro inteiro no array (elemento 0) é configurado e aberto para leitura, enquanto o segundo inteiro (elemento 1) está configurado e aberto para gravação. Cada um dos dois comandos será um filho, e os descritores no filho são duplicados na entrada ou saída padrão, utilizando a função dup2(), criando um canal de comunicação entre os dois processos. No primeiro processo a ser executado, indicando o primeiro comando, duplicamos a saída (fd[1]) e fechamos a entrada (fd[0]), assim ele herda o lado de saída do pipe como sua saída padrão. Já para o segundo filho, fechamos a saída padrão (fd[1]) e duplicamos a entrada padrão, fazendo com que o processo faça a leitura dos caracteres da entrada.

Um novo prompt só é exibido e um novo comando só é lido quando o comando da linha anterior termina sua execução, exceto quando o programa deve ser executado em background, representado pelo símbolo "&" ao final da linha de comando. Por default, foi utilizada as chamadas de sistema waitpid(), responsável por suspender a execução do processo de chamada até que um filho especificado pelo argumento pid tenha mudado de estado. Porém, se o processo deve ser executado em background a função waitpid() não é chamada.

3 Execução

O interpretador necessita que determinadas condições sejam satisfeitas para sua obter a execução correta. Por exemplo, caso ele seja executado com um argumento de linha de comando, é necessário que esse arquivo de entrada exista. Caso isso não aconteça, uma mensagem de erro é escrita na tela, e a execução é finalizada. O mesmo ocorre caso o interpretador receba mais de um parâmetro como entrada.

Para que o pipe seja executado corretamente, no caso da linha incluir mais de um comando, é necessário que o primeiro programa contenha o símbolo

“<=”, e que não contenha o símbolo ”=>”. Já o segundo comando pode ou não ter a ocorrência do símbolo ”=>”, indicando se a saída do mesmo será em arquivo. Desse modo, é possível associar a saída do primeiro processo à entrada do segundo.

Também é possível que o interpretador receba comandos vazios, e nesse caso a execução do mesmo continua, aguardando um novo comando. A execução é terminada ao receber como entrada a string ”fim”, um comando ”ctrl-d” ou ”ctrl-c”, ou ao chegar ao final da leitura do arquivo de entrada.