Implementação de um Shell

Professor: Marcelo Honorato Marleta
SCE0136 - Sistemas Operacionais II
Instituto de Ciências Matemáticas e Computação – ICMC
Universidade de São Paulo – USP

Danilo Sacilotto de Souza (Nº USP 6792899)
Gabriel Andrade de Mello (Nº USP 5866141)

1) Introdução

1.1) Interpretador

Um comando do *Linux* é uma palavra especial que representa uma ou mais ações. Um interpretador de comandos também é conhecido como shell ou modo texto. Ele é o programa responsável por interpretar essas instruções enviadas pelo usuário e seus programas para o kernel. No *Linux*, você poderá ter vários interpretadores de comandos (ao contrário do que acontece no Windows que só tem o command.com).

O interpretador de comandos é que executa comandos lidos do teclado ou de um arquivo executável. É a principal ligação entre o usuário.

1.2) Resumo do Projeto

Este projeto trata-se da análise e implementação de um interpretador de comandos simples baseado naqueles geralmente distribuídos com sistemas Unix-like (Linux, *BSD, Solaris). O shell possui comandos integrados (built-in) que executam funções específicas e também é capaz de executar programas que existem no sistema (comandos externos), os quais devem executar em um outro processo que é criado após o usuário especificar a linha de comando. O foco deste projeto está nas características de job control (gerência dos processos em execução a partir do shell). Ao decorrer deste projeto nos tornamos mais familiarizados com a semântica das chamadas do sistema que controlam processos, especificamente criação, término, hierarquia e E/S em arquivos.

2) Características do Projeto

2.1) Totalmente Implementadas

- Um interpretador para linha de comando lidos do teclado;
- Suporte a execução de programas em segundo plano (“&”, *background*);
- Suporte para o comando jobs;
- Suporte a redirecionamento de entrada e saída;
- Suporte para controle de processo (job-control);

2.2) Parcialmente Implementadas

- Não houve nenhuma característica parcialmente implementada.

2.3) Não Implementadas

- Suporte a pipes.

3) Testes do Projeto

Abaixo será apresentada uma breve descrição de como foram realizados alguns dos testes para cada característica implementada.

- Para interpretador de linha de comando:
  Comando: $ echo TESTE
  Resultado: foi exibida na tela a palavra “TESTE”.

  Comando: $ man echo
  Resultado: foi aberta a documentação man referente ao comando echo.

- Para execução de programas em segundo plano:
  Comando: $ xeyes &
  Resultado: o programa xeyes foi aberto e o terminal ficou “desbloqueado”, ou seja, sendo possível qualquer manipulação com o mesmo.

- Para suporte ao comando jobs:
  Comando: $ jobs
  Resultado: todos os processos em execução e parados foram exibidos com seu respectivo *id.*

- Para redirecionamento de Entrada e Saída
  Comando: $ echo TESTE > teste.txt
  Resultado: a palavra “TESTE” foi adicionada ao arquivo “teste.txt”, sendo que o mesmo não existindo foi criado.

  Comando: $ ./teste < teste.txt
  Resultado: o conteúdo de “teste.txt” foi utilizado como entrada para o programa “teste”.

- Para controle de processo (job-control)
  Comando: Control+C
  Resultado: o processo (programa) que estava em execução foi finalizado.

  Comando: Control+Z
  Resultado: o processo (programa) que estava em execução foi para o estado de parado.

Comando: após o comando Control+Z, $ bg 0
Resultado: o processo de número 0 (zero) foi colocado em segundo plano.

Comando: $ fg 0
Resultado: o processo de número 0 (zero) foi colocado em primeiro plano.

4) Conclusão

Em resumo, implementamos uma shell que contem a maioria das características pedidas, o que possibilitou fixar o conceito de fork, as chamadas de sistema apresentadas e a visão geral do agendamento de processos e outros detalhes interessantes do funcionamento de um S.O. baseado em Unix como o Linux.

Um dos maiores limitantes ao uso dessa shell é, de fato, a inexistência de pipelining.

A proposta do projeto é bastante válida para um primeiro interpretador de comados UNIX, pois é um meio-termo interessante entre “baixo nível” (manipulação de processos, chamadas do sistema, etc.) e a segurança e praticidade de uma aplicação rodando em user-mode. Desta forma, tanto alunos com mais experiência em programação quanto iniciantes conseguem se beneficiar.