Instituto de Ciências Matemáticas e de Computação
Departamento de Sistemas de Computação

SSC141 – Sistemas Operacionais II
1º Sem 2009

Trabalho 2 – Mini Shell

1. Objetivos
   Escrever um programa Shell para o sistema operacional Linux.

2. Motivação
   Um Shell é um interpretador de comandos de linha interativo que pode executar programas especificados pelo usuário. Um Shell repetidamente imprime um prompt e aceita uma linha de comando ser digitada em stdin.

   A linha de comando é uma sequência de palavras codificadas em ASCII e delimitadas por espaços em branco. A primeira palavra na linha de comando ou é o nome de um comando embutido do Shell ou é o caminho de um comando externo (programa executável). As palavras restantes são argumentos ou outros comandos.

   Se a primeira palavra é um comando embutido, o Shell executa este comando no processo atual. Se for o caminho de um programa executável, o Shell executa um comando fork criando um processo filho e executa o programa no contexto do processo filho. Os processos filho criados por uma única linha de comando são denominados coletivamente por job. Em geral, jobs são constituídos por múltiplos processos filho conectados por pipes.

   Se a linha de comando termina com um ´&´, então o job é executado em background, o que significa que o Shell não espera o job terminar para imprimir o prompt e esperar pela próxima linha de comando. Caso contrário, o Shell executa o job em foreground, esperando o término da sua execução para voltar a imprimir o prompt. No máximo um job pode estar em execução em foreground, embora muitos possam estar em execução em background.

3. Parte 1 – Programa Parser e comandos externos

   O parser deve aceitar uma linha de comando com palavras separadas por um ou mais espaços em branco. A primeira destas palavras deve ser um comando embutido ou um comando externo. O Shell deve criar um processo filho para a execução de comandos externos.

   O prompt do Shell implementado deve conter o número do grupo que o implementou, por exemplo:

   B2>

Testar o shell com comandos externos básicos do Unix como ls, ps, kill, etc. O Shell deve garantir que o programa receba os parâmetros corretos, por exemplo:

B2> /bin/ls –l –d

Deve fazer com que a função main do programa ls, receba os parâmetros:

- argc == 3
- argv[0] == "/bin/ls"
- argv[1] == "-l"
- argv[2] == "-d"

4. Parte 2 – Controle de Jobs e programas internos

   Nesta segunda parte do trabalho, o Shell deve permitir a execução de programas em background, provendo números de job e seu controle.

   Shells Unix provêem o controle de jobs permitindo que sua execução possa ser movida de foreground para background e vice versa. Pode-se também alterar o estado do processo de um job entre em execução, parado e terminado.

   Cada job pode ser identificado por um PID ou JID (job ID). O JID é um número inteiro positivo definido pelo Shell. Na linha de comando, PIDs aparecem como números sem prefixos enquanto JIDs números com o prefixo '%`.

   Os seguintes comandos internos devem ser implementados:

   - pwd – mostra o diretório corrente
   - cd – troca de diretório
   - quit – termina o Shell
   - jobs – lista todos os jobs em background
   - bg <PID ou JID> – reinicia um job enviando a ele o sinal SIGCONT e passando sua execução para background
   - fg <PID ou JID> - reinicia um job enviando a ele o sinal SIGCONT e passando sua execução para foreground

5. Parte 3 – Sinais básicos

A digitação de CTRL-C deve enviar um sinal SIGINT para todos os processos do job em foreground, (processo principal e processos filhos criados pelo processo principal). O efeito do sinal SIGINT é o termino dos processos. Se não existe um job em foreground, então o SIGINT não deve ter qualquer efeito.

A digitação de CTRL-Z, deve enviar um sinal SIGTSTP para todos os processos do job em foreground. O efeito do sinal SIGTSTP é colocar todos os processos no estado "parado", no qual eles devem permanecer até receberem um sinal SIGCONT.

6. Parte 4 – Redirecionamento de E/S e Pipes

Implementar as seguintes operações de redirecionamento de E/S

- > - stdin para arquivo (se o arquivo já existe, a informação original é perdida)
- >> - stdin para adição no final de arquivo (se o arquivo não existe, ele é criado)
- 2> - sterr para arquivo
- < - recebe a entrada de um arquivo, em vez de stdin

Implementar a comunicação entre processos por pipes, prevendo qualquer número de processos concatenados.

Referências:

- http://linuxgazette.net/111/ramankutty.html
- Haviland, K.; Gray, D., Salama, B.; Unix System Programming, Addison Wesley, 1998.
- Love, R.; Linux System Programming; O´Reilly Media Inc, 2008.
- Stevens, R.; Rago, S.A.; Advanced Programming in the Unix Environment; 2nd edition, Addison Wesley, 2005.
- http://w3.linux-magazine.com/issue/40/Writing_a_Shell.pdf

Todos os trabalhos devem ser enviados por e-mail para:
ssc141@icmc.usp.br
O subject da mensagem deve estar na forma T12B3, onde:
T1 = trabalho 1 parte 2
B3 = grupo B3