**Trabalho 1**

**Shell Unix com Suporte à Histórico de Comandos**

Este trabalho consiste no projeto e implementação de um programa C para atuar como um shell que aceita comandos do usuário e então executa cada comando em um processo separado (processo filho). Este trabalho pode ser feito em qualquer sistema Linux, UNIX, ou Mac.

Um shell, após ser iniciado, aguarda a entrada de comandos por parte do usuário. O exemplo abaixo ilustra um caso onde um usuário entrou com um comando chamado cat na interface shell. Como resposta, o shell executa o programa cat no contexto de um processo filho. Como resultado, o conteúdo de arquivo prog.c é listado na saída padrão (stdout).

osh> cat prog.c

Uma técnica para implementar um shell consiste em ter um processo pai que lê os comandos que o usuário deseja executar (no exemplo anterior, cat prog.c), e então cria um processo filho separado onde o comando (executável) será efetivamente executado. A menos que seja especificado o contrário, o processo pai espera a finalização do processo filho para depois prosseguir. Porém, tipicamente, os shells também permitem que o processo filho executem em *backgound*, ou seja, concorrentemente. Para que isso aconteça, nós adicionamos um & no final do comando. Assim, se nós rescrevermos o comando acima como

osh> cat prog.c &

o processo pai (shell) e filho (comando) executarão concorrentemente.

Para cada comando, um processo filho é criado usando a chamada de sistema fork() e o executável associado ao comando é executado no contexto do processo filho usando uma chamada de sistema da família exec() (veja no manual todas as variações possíveis e escolha a mais apropriada para o seu caso particular). Um esqueleto para um shell é dado abaixo:

#include <stdio.h>

#include <unistd.h>

#define MAX\_LINE 80 /\* Tamanho máximo da linha de comando \*/

**int** main(void)

{

**char** \*args[MAX\_LINE/2 + 1]; /\* Número máximo de argumentos, incluindo o nome do executável \*/

**int** executar = 1; /\* flag para determinar quando deve sair do programa \*/

**while** (executar ) {

printf("osh>");

fflush(stdout); /\* força o esvaziamento do buffer de saída stdout (pesquise o manual) \*/

/\* Depois de ler a entrada, os passos são:

\* (1) criar um processo filho fork().

\* (2) o processo filho chama execvp() (ou outra chamada de sistema similar)

\* (3) se o commando não incluir &, pai deve chamar wait()

\*/

}

**return** 0;

}

A primeira tarefa é modificar a função main() de modo que o processo filho execute o comando solicitado pelo usuário. Isto requer que a string de entrada seja analisada para recuperar os *tokens* (argumentos) que compõe o comando especificado. Esses tokens serão usados com uma das chamadas de sistema da família exec(). Assegure-se de tratar o caso de & ser usado com o comando.

O próximo passo é implementar suporte a histórico de comandos digitados. O usuário deve ter a possibilidade de visitar diretamente os n últimos comandos digitados (por exemplo, os dez últimos), usando algum atalho do teclado (seta para cima e seta para baixo). Sugiro utilizar uma lista ligada para gerenciar essa funcionalidade.

Data de entrega: 06 de abril de 2015.