Sistemas Operacionais

Atividade - Terminal de linha de comandos (*shell*)

1 Informações práticas

Disciplina: Sistemas Operacionais.

Data de entrega: 05/08/2024.

Forma de Entrega: *Link github*, com `README.md` e códigos fontes. Também haverá uma apresentação individual, com explicação do código.

OBS: ***Atividade individual.***

2 Objetivo

Proporcionar aplicação prática dos conceitos de gerenciamento de processos em sistemas operacionais *Unix* e derivados.

3 Intrudução

Uma **shell** é um programa que facilita o uso do computador pelo usuário. A `bash`, uma **shell** usada no linux, por exemplo, é uma programa executável que pode ser encontrada no diretório `/bin`. O nome completo da `bash` no sistema de arquivos é `/bin/bash`.

Tente executar `/bin/bash` e voce pode perceber que ele se executa normalmente como qualquer outro programa. Ao digitar `exit` da `bash` você termina a sessão atual e o controle volta para o programa que chamou a `bash`, nesse caso a `bash` inicial onde você estava conectado.

Quando você se *loga* em um sistema, o programa de login, que pede usuário e senha, após autenticar o usuário executa o programa `/bin/bash`, ou outra **shell** configurada na conta do usuário. No caso do linux a **shell** pode ser configurada pelo administrador do sistema (usuário **root**) no arquivo `/etc/passwd`.

4 Descrição

Você deve desenvolver uma **shell** simples, em linguagem C/C++, de acordo com os seguintes requisitos:

4.1 Parte 1: Básico

1. A **shell** deve executar um laço infinito, mostrando como prompt EXATAMENTE o símbolo `$` (cifrão), sem espaços ou qualquer outro caractere. No caso da `bash` o prompt pode ser configurada através da variável **PS1**. Não é obrigatório, mas você pode fazer o prompt configurável também. Caso seja feito coloque na documentação

2. A **shell** deve esperar o usuário digitar uma linha por vez da entrada padrão (`stdin`). A linha é composta de um comando e um conjunto de argumentos. A **shell** deve processar a linha para separar o comando e os argumentos.

   - Pode ser considerado como separador de palavras apenas o espaço (ASCII 32). Fique a vontadade para usar também o `[TAB]`.
   - Pode não considerar caracteres especiais, tipo `[TAB]`, Aspas (simples ou duplas), *backspace*, etc. Isso implica que a sua **shell** não poderá tratar argumentos com espaços no nome. Por exemplo, arquivos e/ou diretórios com espaços no nome não poderão ser tratados pela sua **shell** nesse caso.

3. Depois de processar a linha de comando a **shell** deve executá-lo. O comando pode ser o nome de um arquivo executável ou um comando interno (veja a parte 2: comandos internos). Para a parte 1 considere apenas comandos externos.

4. Para executar um comando externo a **shell** deve criar um processo filho com a chamada de sistema `fork()` e depois substituir o código executável do processo filho com a chamada `exec()`, ou uma de suas variantes. O processo pai deve esperar que o processo filho termine antes de continuar. Caso deseje implementar processos em segundo plano (*background*) haverá um aumento na nota do trabalho.

   Para implementar processos executando em *background* você pode usar o padrão da **bash**, que é adicionar o & (e comercial) no final do comando ou definir seu próprio padrão. Caso haja implementação de processos em *background* você deve implementar um comando para listar todos os processos em execução, como o comando `jobs` da **bash**. Fique a vontade para incrementar as funcionalidades de controle de processos: voltar um dos processos para *foreground*, finalizar (*kill*) um processo, etc.

   Aqui tem um tutorial sobre manipulação de processos no terminal de linha de comandos.

5. [**Opcional**] Implemente o gerenciamento de **variáveis de ambiente**. A *shell* possui variáveis que armazenam valores. Estes valores podem ser usados por comandos internos ou para configurar o ambiente da *shell*. Alguns exemplos de variáveis de ambiente mais usadas para configuração são a `HOME`, `PATH` e a `PS1`, usadas para definir o diretório *home* do usuário, configurar os diretórios a serem usados para execução de comandos externos e a aparência do *prompt* de comando da *shell*, respectivamente. Aqui, aqui e aqui há informações sobre o funcionamento de variáveis de ambiente.

4.2 Parte 2: Comandos internos

O terminal de linha de comandos é responsável por executar comandos de programação instalados no sistema. Além dos programas instalado, a **shell** possui um conjunto de comandos que ela reconhece e executa. Estes comandos são chamados de **comandos internos da shell** (*shell builtin commands*).

A **shell** deve considerar os seguintes comandos internos.

`exit [n]` O comando exit simplesmente sai da *shell* e retorna 0 (zero), realizando as devidas liberações (memória, arquivos, etc), caso necessário.

`pwd` Mostra na tela o diretório atual do usuário.

`cd dir` Muda o diretório atual do usuário para `dir`, caso exista. Se não existir o diretório `dir` a *shell* mostra uma mensagem de erro.

`history [-c] [offset]` O comando history (https://www.gnu.org/software/bash/manual/html_node/Bash-History-Builtins.html) deve mostrar os último comandos executados, similar o history da **bash**, porém mais simples.

1. `history` sem argumentos deve mostrar os últimos 10 comandos digitados, um por linha. Cada linha deve conter um número a esquerda, chamado *offset*, do comando e um espaço entre o número e o comando. Os comandos são numerados a partir de 0 (zero) até 9 (nove), sendo 0 (zero) o comando mais recente.
2. `history -c` deve apagar todos os comandos do histórico.
3. `history [offset]` deve executar o comando de número [offset]. A shell deve mostrar um erro caso o número não seja válido.
4. Exemplo de uma saída do history:

```
$ history
9 /bin/ls
8 cd /tmp
7 /bin/ls
```

```
6 cat teste.txt
5 cd /home/jorgiano
4 cd
3 ls
2 clear
1 cat lista.txt
0 clear
$
```

Caso sua *shell* use variáveis de ambiente, defina comandos para manipular as variáveis.

4.3 Parte 3: Comandos externos

Para executar comandos externos, ou seja, programas executáveis armazenados em algum diretório, a *shell* deve procura o nome do comando digitado, que deve ser o nome de um arquivo em um diretório. O sua *shell* deve armazenar uma lista com os caminhos completos dos diretório onde deve procurar o comando a ser executado. No caso da **bash** há uma variável de ambiente, chamada `PATH` que armazena uma *string* contendo todos os diretórios separados por `':'` (dois pontos). Ao se modificar essa variável o usuário modificar a lista de diretórios onde a *shell* procura os diretórios. Aqui e aqui há material que explica a variável de ambiente `PATH`. Fique a vontade para definir seu próprio modelo de definição de lista de diretórios onde a *shell* procura os programas executáveis.

Caso você não implemente variáveis de ambiente, defina um conjunto de diretórios pré-definidos onde a *shell* procura os comandos externos executáveis.

5 Dicas

Use as *man pages* do linux para acessar a documentação das funções. As *man pages 2* possuem documentação das chamadas de sistema e as *man pages 3* as das bibliotecas de **C**. Por exemplo, para ler a documentação da chamada de sistema *fork* digite `man 2 fork`. Para a documentação da função *printf* digite `man 3 printf`.

- Para dúvidas de utilização do linux acesse a página da disciplina do Google Sala de Aula.
- A maioria dos problemas que vocês encontrarão possivelmente outras pessoas também encontraram `:)` Use o google para procurar respostas. O site stackoverflow possui perguntas e respostas sobre programação e pode ser útil para suas dúvidas. A versão em inglês é mais completa.

Para facilitar o entendimento da implementação segue um início de código de *shell*:

Código 1 - shell.cpp

```
#include <iostream>
#include <unistd.h>
#include <sys/wait.h>

void process_command(std::string command) {
    // Se for comando interno...
    if (command == "exit")
        exit(0);

    // Se for comando externo

    // * necessário verificar se é para ser executado em background
    /*  Se for caminho relativo, procurar o comando na lista de diretórios
        Se for absoluto verifica se comando existe
    */
    std::string absolute_path = "/bin/" + command;
    if (access(absolute_path.c_str(), F_OK) == 0) { // Arquivo existe no diretório
        if (access(absolute_path.c_str(), X_OK) == 0) { // Arquivo é executável
            pid_t pid = fork();
            if (pid < 0){ // Erro
                std::cout << "Erro de execução!" << std::endl;
                return;
            } else if (pid == 0){ //processo filho
                char * argv[2] = {(char *)command.c_str(), nullptr};
                execve(absolute_path.c_str(), argv, NULL);
            } else { // Processo pai
                /* Deve adicionar processo filho na lista (std::vector)
                   de processos em execução para gerenciar background. */
                /* Processo pai espera processo filho terminar. */
                waitpid(pid, nullptr, 0);
            }
        } else { // Arquivo não é executável
            std::cout << "permission denied: " << command << std::endl;
        }
    } else { // Arquivo não existe
        std::cout << "Command not found: " << command << std::endl;
    }
}

int main() {
    while (true) {
        std::cout << "$> ";
        std::string command;
        getline(std::cin, command);
        process_command(command);
    }
    return 0;
}
```