Trabalho Prático 1, Sistemas Operacionais

Vítor Rezende Silva

Maio 2023

1 Introduction

O objetivo desse trabalho é implementar um interpretador de comandos, chamado shell, utilizando chamadas de sistema na linguagem C.

2 Executando comandos e arquivos

A execução de comandos do shell ou de arquivos em C é feita pela função executa_arquivo().

Essa função faz o uso das chamadas de sistema fork() e execvp(). fork() é a função responsável por criar um processo filho, esse processo executa o comando pela função execvp() e em seguida encerra esse processo.

A função execvp() foi escolhida uma vez que nela não é necessário saber o local dos arquivos que serão executados pela função.

Além disso, é possivel executar um proccesso em background, nesse caso, o processo pai não aguarda o processo filho encerrar para continuar.

3 Redirecionando de entrada e saída

3.1 Redirecionamento de entrada

O redirecionameto de entrada da entrada padrão para um arquivo é uma operação representada pur um comando na forma: "processo " <= " arquivo", e indica que o processo deve aceitar como entrada o conteúdo de um arquivo no lugar da entrada padrão. Essa função exige que o arquivo exista previamente, exibindo uma mensagem de erro caso ele não exista.

Uma vez identificado o comando, o redirecionamento é realizado pelas funçoes redirect_stdin() e return_stdin(). redirect_stdin() recebe como parâmetro o nome do arquivo de entrada, a função então abre esse arquivo como um inteiro, duplica a entrada padrão anterior através da chamada de sistema dup(), e através da chamada dup2() o shell troca a entrada padrão para o arquivo. A função ao fim da execução retorna o valor inteiro saved_stdin, que representa a entrada padrão anterior

A função return_stdin() retorna a entrada para a entrada padrão, ela recebe como parâmetro o valor saved_stdin, que é gerado pela redirect_stdin(), e através da chamada dup() ela retorna a entrada para a entrada padrão. Essa função é chamada após o processo filho ser executado.

3.2 Redirecionamento de saída

O redirecionamento de saída padrão para um arquivo é representada por um comando na forma: "processo => arquivo", e indica que o processo deve aceitar como saída padrão a escrita nesse arquivo. Ao contrário do redirecionameto de entradas, o arquivo pode não existir antes da execução do comando, e nesses casos ele deve ser criado.

Analogamente ao redirecionamento de entradas, o redirecionamento de saída é feito pelas funções redirect_stdout() e return_stdout(), que operam de forma muito similar às funções mencionadas acima, primeiro a saída padrão é trocado pela saída de um arquivo, em seguida o processo é executado e por fim a escrita no arquivo é trocado pela saída padrão.

3.3 Redirecionamento silmultâneo de entrada e saída

O redirecionamento de entrada e saída ao mesmo tempo é possível no shell implementado. Essa operação é identificada pelo comando "processo <= arquivo_in => arquivo_out".

4 Pipes

Pipes são estruturas muito úteis para a comunicação entre processos. Com ele o shell é capaz de pegar a saída de um processo e usar ela como entrada de um outro processo.

Uma operação de pipe é sinalizada para o shell usando o caractere "|", na forma "processo1 argv | processo2 argv", onde a saída do processo à esquerda será usada como entrada para o processo à direita.

Essa operação será realizada pelas funções Pipeline() e Pipe(), A primeira tem como argumento a linha de comandos, a posição do primeiro comando e do final do ultimo comando, o tamanho desse array e o inteiro background, essa função separa os processos da linha de comando e os repassa para a função Pipe(), que tem como parâmetro um array contendo o nome de todos os processos escritos na linha de comando, junto com seus argumentos de linha de comando, o tamanho desse array e o inteiro background, que vai sinalizar se o processo deve ou não executar em background.

Vários processos podem ser encadeados com o uso do sinal "—", ou seja, o pipe não é limitado a 2 processos.

5 Executando processos em background

A execução de processos em background consiste no processo pai não esperar o processo filho terminar sua execução para continuar. Isso é executado pelo sinal & no final da linha de comando.

Uma vez identificado que um processo deve ocorrer em background, o shell irá passar o valor do inteiro background como 1 para as funções responśaveis por executar um processo, nesse caso as chamadas de sistema waitpid e wait não irão executar.

6 Interpretando a linha de comando do shell

Interpretar a linha de comandos do shell não é trivial, visto que podem existir uma quantidade indefinida de processos, argumentos e operadores. A forma como isso é feita no shell implementado é através da identificação dos sinais.

O primeiro passo para isso é dividir a linha de comando usando um espaço em branco como separador. Nisso é gerado um array de strings chamado command_parsed. Em seguida deve-se encontrar o sinal "|" para descobrir se ocorrerá um pipe, depois os sinais " <= " e " => " para identificar o redirecionamento de entrada e sáida, respectivamente.

Caso haja o redirecionamento de entrada e saída, o processo ja altera a entrada e saída padrão para os arquivos selecionados, e depois da execução de um comando ele retorna para a entrada e saída padrão.

Caso não ocorra pipe, o programa vai gerar outro array de strings chamado argv, de estrutura similar aos argumentos de linha de comando, em seguida execcuta o programa indicado,

Caso ocorra pipe, o programa vai chamar a função Pipeline() com a linha de comando como parâmetro.

Além disso, caso o ultimo item da linha de comando seja o sinal "", os processos indicados devem executar em background.

Outro comando importante, é o "fim", ou Ctrl + D, que quando identificado deve encerrar a operação do shell.

O shell identifica quando uma linha está vazia, caso seja o caso ele vai pra próxima iteração do loop principal, imprimindo o prompt novamente.

Caso o shell seja executado com argumentos de linha de comando, argv e argc, ele passa seu conteúdo para a função caminhar_nos_comandos(), que executa o processo indicado e encerra a execução do shell.

7 Funções Auxiliares

Para executar algumas funções essenciais para o funcionamento foram criados outros dois arquivos em C, argv.c e auxiliar.c

7.1 argv.c

As funções desse arquivo são responsáveis por criar o subconjunto argv da linha de comando, que contem todos os comandos que não são sinais ou nomes de arquivos. O array argv é essencial para a execução de processos.

7.2 auxiliar.c

O arquivo auxiliar.c possui funções essenciais para o funcionamento, mas que não se encaixam tão bem no código dos outros arquivos, como contar o numero de comandos presentes na linha de comandos, separar ela com ajuda da função strtok() e procurar arquivos.