Trabalho Prático - Shell Básico

Lucas M. Barreto Rezende[1], Luiza Sodré Salgado[1]

[1]Departamento de Ciência de Computação – Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte – MG – Brazil

{lucasmbr,luiza-salgado}@ufmg.br

Abstract. *This project aims to implement some concepts from the first part of the course "DCC605 - Operating Systems." In this regard, concepts of pipes and kernel process structures were applied, aiming to implement the functionalities of a simple shell.*

Resumo. *Este projeto visa implementar alguns conceitos da primeira parte da disciplina 'DCC605 - Sistemas Operacionais'. Nesse sentido, foram aplicados conceitos de pipes e estruturas de processos de kernel, visando implementar funcionalidades de um shell simples.*

1. Implementações

Descrições sucintas das soluções implementadas neste projeto e explicações sobre os funcionamentos do código.

1.1. Fork

A implementação do fork1() foi feita da seguinte maneira:

1. Criação de novo processo usando pid_t (a signed integer type which is capable of representing a process ID).

2. Se o fork falhar, imprime uma mensagem e retorna -1. Caso contrário retorna o ID do processo filho.

1.2. handle_simple_cmd

A função handle_simple_cmd executa comandos simples (sem pipes ou redirecionamentos). Ela é chamada pelo runcmd quando detecta esse tipo de comando. O processo pai cria um filho com fork1(). No processo filho, execvp substitui a imagem do processo pelo comando indicado (ex.: ls, cat, etc.). Se execvp falhar, imprime uma mensagem de erro e encerra com exit(1). O processo pai aguarda a finalização do filho usando wait(NULL). Como o runcmd previamente verifica se o comando é nulo (if (ecmd->argv[0] == 0)), basta usar execvp (function in C that replaces the current process with a new one, taking a file name and argument list).

1.3. handle_redirection

Fecha o descritor de arquivo padrão que será substituído. Abre o arquivo especificado em rcmd->file com as permissões corretas (rcmd->mode).cSe o open falhar, imprime uma mensagem de erro e sai.

1.4. handle_pipe

Cria um pipe. Isso cria dois descritores de arquivo em p: p[0] para leitura e p[1] para escrita. Um fork() cria um novo processo.

No processo filho (pcmd->left):

- Redireciona saída padrão (stdout) para a ponta de escrita do pipe (p[1]). Para isso, fecha o stdout (close(1)) e duplica p[1] usando dup2(p[1], 1).
- Fecha as pontas do pipe que não serão usadas (close(p[0]) e close(p[1])).

No processo pai (pcmd->right):

- Redireciona sua entrada padrão (stdin) para a ponta de leitura do pipe (p[0]). Para isso, fecha o stdin (close(0)) e duplica p[0] usando dup2(p[0], 0)
- Fecha as pontas do pipe que não serão usadas (close(p[0]) e close(p[1])).
- O pai espera o filho terminar com wait().

1.3. Task 5 e problemática na mensagem de erro

O propósito do if, conforme a documentação dada por 'help cd', é implementar a troca de diretório usando o comando cd [-L|[-P [-e]] [-@]] [dir], i.e., "change the working directory of the current shell execution environment". Logo, o cd não pode ser executado num processo filho, pois se isso fosse feito o diretório do processo pai não seria alterado.

Nesse sentido, a mensagem de erro está incorreta pois afirma que o processo não existe, quando na realidade o problema é que o diretório não pode ser acessado/encontrado. No que tange a correção da mensagem, é adequado usar "No such file or directory" como mensagem de erro, pois é a mensagem de erro padrão usada no bash e explica de fato o problema que está ocorrendo.

3. References

References

The GNU Project (2001) “Process Identification”, The GNU C Library Reference Manual, https://ftp.gnu.org/old-gnu/Manuals/glibc-2.2.3/html_node/libc_554.html, January.

Leffler, J. (2018) “Where do I put perror("wait") with fork code”, Stack Overflow, https://stackoverflow.com/questions/49419299/where-do-i-put-perrorwait-with-fork-code, April.

The Linux man-pages project (2015) “execvp(3): execute file”, Linux Programmer's Manual, https://linux.die.net/man/3/execvp, August.