Angelica Stewart
Teammate: Mary Nkata
Operating Systems Project 1
Creating a Shell Program
10/17/2023

Documentation for Shell Program with I/O Redirection and Piping Functionality

Introduction

This document provides comprehensive documentation for the shell program, highlighting its features, design, and usage. The shell program is a command-line interpreter that supports various Unix-like commands, incorporating advanced functionalities such as input/output redirection and piping. The program allows users to execute commands, navigate the file system, and manage environment variables efficiently.

Table of Contents

1. Overview

The shell program is designed to provide a user-friendly interface for interacting with the operating system. It includes support for essential commands, input/output redirection, and piping, making it a powerful tool for both beginners and experienced users.

2. Features

a. Prompt and User Interface

- ★ The shell displays a prompt indicating the current working directory, providing context to the user.

b. Built-in Commands

- ★ cd: Change directory.
- ★ exit: Terminate the shell.
- ★ pwd: Print current working directory.
- ★ echo: Print messages and environment variables.
- ★ env: Display current environment variables.
- ★ setenv: Set environment variables.

c. Signal Handling

- ★ The shell handles SIGINT (Ctrl-C) and SIGALRM (process timeout) signals gracefully.

d. Input/Output Redirection

- ★ Support for output redirection using `>` symbol.
- ★ Support for input redirection using `<` symbol.

e. Piping

- ★ Support for piping using `|` symbol to chain multiple commands together.

f. Error Handling

- ★ Informative error messages are displayed for incorrect commands or file not found errors.

3. Usage

- ★ Prompt: The shell prompt displays the current working directory. Users can enter commands after the prompt.
- ★ Command Execution: Enter Unix-like commands to execute them.

★ Built-in Commands: Use built-in commands (`cd`, `exit`, `pwd`, `echo`, `env`, `setenv`) for specific functionalities.
★ Redirection: Use `>` for output redirection and `<` for input redirection.
★ Piping: Use `|` to chain multiple commands together, where the output of one command serves as input for the next.

4. Built-in Commands

a. `cd [directory]`
★ Change the current directory to the specified directory. If no directory is provided, go to the home directory.

b. `exit`
★ Terminate the shell program.

c. `pwd`
★ Print the current working directory.

d. `echo [message or environment variable]`
★ Print the specified message or value of the environment variable(s).

e. `env`
★ Display the current environment variables.

f. `setenv [variable] [value]`
★ Set the specified environment variable to the given value.

5. Signal Handling

★ SIGINT (Ctrl-C): The shell gracefully handles SIGINT, allowing the user to interrupt the current command.

★ SIGALRM (Process Timeout): The shell sets an alarm for 10 seconds for each command. If a command runs longer than 10 seconds, it times out and displays an error message.

6. Input/Output Redirection

- Output Redirection (`>`): Redirects the standard output of a command to a file. If the file does not exist, it is created. If it exists, its contents are overwritten.

- Input Redirection (`<`): Redirects the standard input of a command from a file.

7. Piping

- Pipe Operator (`|`): Allows chaining multiple commands together. The output of one command becomes the input for the next.

8. Error Handling

- The shell provides clear and informative error messages for various scenarios, enhancing the user experience.

9. Conclusion

The shell program offers a robust and user-friendly interface for executing Unix-like commands. With its support for input/output redirection and piping, it provides advanced functionalities, making it a valuable tool for both novice and experienced users. The shell's intuitive interface, built-in commands, and error handling capabilities make it an efficient and accessible tool for interacting with the operating system.