Documentation Report for Quash Shell

Overview

The Quash Shell is a simple command-line shell implementation in C that mimics basic functionalities of Unix-like shells. It supports executing built-in commands, handling input/output redirection, piping between commands, and managing background processes. The design emphasizes modularity, readability, and maintainability, making it an excellent educational tool for understanding shell operations and process management.

Design Choices

1. Structure and Modularity

The code is organized into distinct functions, each responsible for specific tasks, enhancing readability and maintainability. Key functions include:

- print_prompt(): Displays the shell prompt, including the current user, hostname, and working directory.
- tokenize_command(): Parses the input command line into individual arguments, handling quoted strings and environment variable expansion.
- builtin_commands(): Checks if the entered command is a built-in command (like cd, pwd, echo, etc.) and executes it if so.
- create_child_process(): Manages the creation of child processes for executing commands, including handling background processes and redirection.
- execute_pipe(): Implements piping between two commands, allowing the output of one command to be used as the input for another.
- execute_redirection(): Handles input and output redirection using > and < operators.

2. Signal Handling

The shell uses signal handling to manage user interruptions (SIGINT) and process timeouts (SIGALRM). This allows the shell to gracefully handle user requests to terminate processes and to automatically kill processes that exceed a specified execution time (10 seconds in this case).

- handle_sigint(int sig): Responds to the SIGINT signal by printing a new prompt and allowing the user to continue interacting with the shell.
- handle_timer_alarm(int sig): Responds to the SIGALRM signal by terminating the foreground process if it exceeds the time limit.

3. Environment Variable Expansion

The shell supports environment variable expansion, allowing users to reference environment variables in their commands. This is achieved by checking if an argument starts with $ and retrieving its value using getenv().

4. Input/Output Redirection and Piping

The shell implements input/output redirection and piping, which are essential features for command-line interfaces. The execute_redirection() function handles redirection, while execute_pipe() manages the piping of commands.

5. Background Process Management

The shell allows users to run processes in the background by appending & to the command. The shell keeps track of the foreground process using the foreground_pid variable, enabling it to manage timeouts and interruptions effectively.

Code Documentation

Main Function

The main() function serves as the entry point for the shell. It initializes necessary signal handlers, enters a loop to read user input, and processes commands.

Command Tokenization

The tokenize_command() function splits the input command line into individual arguments, handling spaces, quotes, and environment variables.

Built-in Commands

The builtin_commands() function checks if the command is a built-in command and executes it accordingly.

Process Creation

The create_child_process() function forks a new process to execute the command, handling redirection and piping as needed.

Redirection and Piping

The execute_redirection() and execute_pipe() functions manage input/output redirection and piping between commands.

Testing

The shell can be tested using various commands, including built-in commands, external commands, and combinations of redirection and piping. The provided testfile.c and background.c can be compiled and executed to verify the shell's handling of background processes and output redirection.

Error Handling

The shell includes error handling for various scenarios, such as:

- Invalid Commands: If a command cannot be executed, an error message is printed using perror().
- File Handling Errors: When attempting to open files for redirection, the shell checks for errors and provides appropriate feedback.
- Environment Variable Expansion: If an environment variable does not exist, the shell substitutes it with an empty string.
-

Future Improvements

While the Quash Shell provides a solid foundation for a command-line interface, there are several areas for potential enhancement:

1. Job Control: Implementing job control features to manage foreground and background jobs more effectively, including the ability to bring background jobs to the foreground.
2. Command History: Adding a command history feature to allow users to navigate through previously executed commands using arrow keys.
3. Tab Completion: Implementing tab completion for commands and file paths to enhance user experience.
4. Scripting Support: Allowing users to execute shell scripts directly from the shell.
5. Improved Error Messages: Providing more descriptive error messages for various failure scenarios to aid debugging.