SimpleShell

Group ID: `110`

GitHub Repo: GitHub

Members: Yash Verma (2023610)

---

1. Overview

This project implements a simplified Unix-like shell called **SimpleShell** in C. The shell reads user input, parses the command string, and executes the given command in a new process. The shell repeats this process in a loop until terminated by the user using `Ctrl+C`.

2. Key Features

a. Command Execution

SimpleShell supports executing standard Unix commands by creating a new process using `fork()` and `execvp()`. Commands are passed as arguments to the `execvp()` function, and the shell waits for the child process to complete unless the command is run in the background.

b. Custom Commands

- **show_history:** This command displays a list of commands executed during the shell's session.

c. Piping

SimpleShell supports pipes (`|`) to allow the output of one command to be used as the input for another. The shell handles pipes by setting up appropriate file descriptors and chaining commands using multiple child processes.

d. Background Processes

Commands can be executed in the background by appending an `&` symbol. The shell parses the command and launches the process in the background, allowing the user to continue entering commands without waiting for the background process to complete.

e. Command History

The shell maintains a history of executed commands along with metadata such as the process ID, start time, end time, and duration of execution. The `show_history` command displays the stored history.

3. Detailed Implementation

a. Command Parsing

The shell reads input using the `readline()` library, which also provides history management. Commands are parsed using the `strtok()` function, which splits the command string into tokens based on whitespace and other delimiters like `|` and `&`.

b. Process Creation

For each command, the shell forks a new child process using `fork()`. The child process executes the command using one of the `exec` family functions, specifically `execvp()`, which searches for the executable in the system's path.

c. Pipes

Pipes are handled by creating multiple child processes connected via file descriptors. The shell redirects the standard output (`stdout`) of one process to the standard input (`stdin`) of the next process in the pipeline using `dup2()`.

d. Signal Handling

The shell uses a signal handler to catch `SIGINT` (triggered by `Ctrl+C`). When this signal is caught, the shell prints a summary of the command history and terminates gracefully.

e. Background Process Management

Background processes are identified by parsing commands that end with **`&`**. The shell launches these commands in a new process but does not wait for them to finish. The user can continue using the shell while background processes run concurrently.

4. Error Handling

The shell performs error handling at various stages, including:

- **Invalid commands:** The shell checks the return value of `execvp()` and displays an error message if the command fails to execute.
- **Pipe creation:** If `pipe()` fails, the shell displays an error message and terminates the command execution.

5. Limitations

- **Job Control:** SimpleShell does not support job control commands like `cd`, `exit`, `jobs`, `fg`, or `bg`, as these require more advanced management of process groups and signals.
- **Environment Variables:** Commands like `export`, `set`, and `unset` are not supported because they need to modify the shell's environment, which cannot be done in a child process.
- **Redirection:** Currently, the shell does not handle redirection (`>`, `<`, `>>`), though this could be added with additional parsing and file descriptor management.

6. Testing and Validation

The shell has been tested with various commands, including:

- Basic commands: `ls`, `echo`, `wc`, etc.
- Piped commands: `cat file.txt | grep 'pattern' | wc -l`, etc.
- Background processes: `sleep 5 &`
- Custom commands: `show_history`

All tests were performed on an Ubuntu machine using the GNU C Compiler (`gcc`).

7. Conclusion

This project implements a fully functional Unix-like shell with support for basic commands, pipes, background processes, and command history. Future improvements could include the addition of job control, redirection, and environment variable handling.