Report: Custom Shell Implementation (`Quash Shell`)

Team Members:

- Kritish Pokharel
- Ankur Gyawali

---

1. Design Choices

a. Prompt Design

We designed the shell prompt to display user-friendly details, including the username, hostname, and the current working directory. This design aligns with the structure of modern shells (e.g., Bash) and improves usability by giving context for commands. The prompt also features colored output to make the shell visually appealing.

b. Command Parsing

The `tokenize_command` function handles command parsing. It supports:

- Standard delimiters (`space`, `tab`, `newline`).
- Quoted strings to allow spaces in arguments.
- Environment variable substitution using the `$` symbol.

The buffer design ensures safe handling of command-line input and prevents overflows.

c. Built-in Commands

The shell implements several built-in commands without relying on external binaries:

1. `cd`: Changes the current working directory.
2. `pwd`: Prints the current directory.
3. `echo`: Displays arguments or resolves environment variables.
4. `exit`: Terminates the shell.
5. `env`: Lists all environment variables.
6. `setenv`: Allows users to define or modify environment variables.

d. Process Management

- **Foreground and Background Processes**:
  The shell supports both foreground and background processes. A process is treated as a background process if the command ends with an `&`. The parent process does not wait for background processes.
- **Timeout Handling**:
  A timer (`SIGALRM`) terminates long-running foreground processes after 10 seconds to ensure responsive performance.

e. Redirection

The shell handles I/O redirection with `>` (output) and `<` (input). It uses the `dup2` system call to redirect file descriptors. This functionality is integrated into the child process created by `fork`.

f. Piping

The shell supports single-level piping (`|`). Piping is implemented by:

1. Splitting the input into two commands based on the `|` symbol.
2. Creating a pipe (`pipe()`), then using `dup2` to redirect `stdout` of the first command to the pipe and `stdin` of the second command from the pipe.
3. Managing both commands as separate child processes.

g. Signal Handling

Two key signals are handled to improve usability and robustness:

- **`SIGINT` (Ctrl+C)**: Prevents abrupt termination of the shell and allows graceful cleanup.
- **`SIGALRM`**: Automatically terminates unresponsive foreground processes.

---

2. Code Documentation

a. File Organization

All functionality is implemented in a single C file for simplicity. The functions are modularized to separate concerns.

b. Functionality Summary

Function Name	Purpose
`print_prompt`	Displays the shell prompt with user, hostname, and current directory.
`tokenize_command`	Parses and tokenizes the command-line input into arguments, supporting environment variables.
`builtin_commands`	Executes built-in commands (`cd`, `pwd`, `exit`, etc.). Returns `true` if a built-in command.
`create_child_process`	Forks a child process to execute non-built-in commands. Supports background processing.
`execute_redirection`	Handles input (`<`) and output (`>`) redirection by modifying file descriptors.
`execute_pipe`	Implements single-level piping by splitting commands and using a pipe for communication.
`handle_timer_alarm`	Kills long-running foreground processes after the timer expires.
`handle_sigint`	Handles Ctrl+C to ensure the shell does not terminate abruptly.

---

3. Key Features

1. **Interactive Prompt**: Provides contextual information (user, hostname, directory).
2. **Support for Environment Variables**: Dynamically fetches values from the environment.
3. **Error Handling**: Prints descriptive errors for invalid commands, redirections, and pipes.
4. **Process Control**: Manages foreground and background processes effectively.
5. **Advanced Features**: Includes piping and redirection, matching functionality in modern shells.

---

4. Future Enhancements

1. **Command History**: Add support for navigating and executing previous commands.
2. **Multi-level Piping**: Extend piping support to handle multiple commands.

3. **Job Control**: Implement features like `jobs`, `fg`, and `bg` for advanced process management.
4. **Dynamic Path Resolution**: Integrate a search for executables in `PATH`.

Repository Link

https://github.com/KritishPokharel/Quash_A_Simple_Shell