Lab 1: Mini UNIX Shell Technical Report

Introduction

This Mini shell is a custom shell program designed to replicate basic functionality found in Unix-like operating systems. The primary objective is to handle command execution, pipe operations, input/output redirection, and background jobs. This report discusses the design, implementation, and functionality of the shell.

1 System Design

The MiniShell program follows a modular design with a focus on simplicity and clarity. It follows:

Read line →tokenize →parse →execute.

The main components are:

- **Command Parsing**: A tokenizer splits the input string into individual tokens, identifying commands and their arguments.
- **Command Execution**: The shell executes commands through system calls such as fork(), waitpid().
- **Pipes and Redirection**: The program supports pipe operations (using the | symbol) and file redirection (using < and >).
- **Background Jobs**: Background execution is handled using the & symbol.

2 Implementation Details

2.1 Tokenization and Parsing

The input is read from the standard input, and a tokenizer splits the string into in-dividual tokens. These tokens are classified into:

- Single/double quotes
- Backslash escaping
- Recognition of special tokens: |, &, <, >, >>

After tokenization, the commands are analyzed and added to a pipeline for execution.

2.2 Executing Commands

Each command in the pipeline is executed by creating a new process using fork(). Input/output redirection is handled by duplicating file descriptors using dup2() before executing the command with execvp().

2.3 Pipes and Redirection

For pipes, a pipe is created between each pair of adjacent commands in the pipeline. The output of one command is connected to the input of the next using the pipe file descriptors. All file descriptors closed in both parent and child. Redirection applied after pipe wiring.

2.4 Background Jobs

The shell supports background job execution. When a command ends with &, the shell creates the command in the background by detaching the child process from the controlling terminal and avoiding waiting for its termination.

3 Error Handling

The program includes robust error handling for common issues such as:

- File-related errors (e.g., inability to open a file).
- System errors (e.g., Expected filename after '>' or '>>', No command before the pipe, Expected filename after '<').

Each error is reported with a helpful message, and the program handles them gracefully by returning to the prompt.

4 Additional features

- Support for $(...) in command
- Current directory prompt
- Background job notification with PGID

5 Testing and Evaluation

The program has been tested with various scenarios, including:

- Single commands with or without arguments.
- Pipe operations with multiple commands.

- Input and output redirection to files.
- Background job execution.
- Built-in commands such as cd, exit, and pwd.

The program performs as expected, and all features work correctly with no significant bugs.

Conclusion

This Mini Shell successfully implements the core functionalities of a command-line shell, including support for pipes, redirection, background jobs, and basic built-in com-mands. The design is modular and simple, making it easy to extend with additional features in the future.