**CS4375: Theory of Operating Systems - Assignment 2A Report**

**Building Your Own Shell**

**By Adalberto Vazquez Villalobos**

**Instructor: Dr. Deepak Tosh**

**Spring 2025**

**Due Date: 02/15/2025, Midnight**

**1. Introduction**

The objective of this assignment was to develop a simple shell called minershell, which reads user commands, executes them using system calls such as fork, exec, and wait, and handles basic command-line interactions.

The shell operates in an interactive while loop, displaying a prompt (minersh$ ) and waiting for user input. It supports standard Linux commands, handles errors gracefully, and includes built-in functionality for changing directories using the cd command.

**2. Implementation Details**

2.1 Command Processing

The shell continuously prompts the user for input. The entered command is tokenized and processed accordingly:

* If the command is exit, the shell terminates.
* If the command is cd, the chdir system call is used to change directories.
* For all other commands, a child process is created using fork(), and execvp() is used to execute the command.

2.2 Key Functionalities Implemented

1. **Basic Command Execution:**

* Commands such as ls, pwd, echo, and cat execute correctly by invoking their respective executables in Linux.
* Example usage:

A screenshot of a computer

AI-generated content may be incorrect.

1. **Graceful Exit:**

* The shell exits cleanly when the exit command is entered.
* Example usage:

A screen shot of a computer

AI-generated content may be incorrect.

1. **Handling the cd Command:**

* The chdir() system call is used to change the working directory.
* Incorrect usage results in an error message.
* Example usage:

A screenshot of a computer

AI-generated content may be incorrect.

1. **Error Handling:**

* Invalid commands display appropriate error messages.
* Fork failures are handled to prevent shell crashes.
* Example usage:

A screenshot of a computer

AI-generated content may be incorrect.

1. **Memory Management:**

* Memory allocated for tokenized input is freed after each command to prevent memory leaks.

**3. Test Scenarios and Screenshots**

To validate the functionality of the shell, the following tests were conducted:

**3.1 Basic Command Execution**

* **Test:** Running ls in the shell
* **Expected Output:** List of files in the current directory

**A screenshot of a computer

AI-generated content may be incorrect.**

**3.2 Change Directory (cd)**

* **Test:** Running cd to navigate directories
* **Expected Output:** No output, but pwd should reflect the change

A screenshot of a computer

AI-generated content may be incorrect.

**3.3 Error Handling for Invalid Commands**

* **Test:** Running an invalid command
* **Expected Output:** Error: The command could not be executed.

**A screenshot of a computer

AI-generated content may be incorrect.**

**3.4 Handling exit Command**

* **Test:** Running exit
* **Expected Output:** Shell terminates

**A screenshot of a computer

AI-generated content may be incorrect.3.5 Ensuring No Zombie Processes**

* **Test:** Running ps before and after executing commands
* **Expected Output:** No leftover child processes

A screenshot of a computer

AI-generated content may be incorrect.

**4. Challenges Faced**

* **Handling cd without forking:** Since cd changes the working directory of the shell itself, it had to be executed in the parent process using chdir(). Forking a child would not persist the directory change.
* **Avoiding memory leaks:** The shell repeatedly processes user input, so dynamically allocated memory for tokens had to be freed after each command to prevent leaks.
* **Error handling for unknown commands:** If a user entered an invalid command, execvp() could fail. The shell needed to handle these cases gracefully by displaying an error message without crashing.

**5. Conclusion**

* The minershell successfully implements a simple shell capable of executing basic Linux commands. The shell correctly handles cd, prevents zombie processes, manages memory efficiently, and gracefully handles errors. Further enhancements could include support for piping (|), redirections (>, <), and background processes (&).

**6. References**

* *An A-Z index of the Linux Command Line: Bash + Utilities.* An A-Z Index of the Linux command line - SS64.com. (n.d.). https://ss64.com/bash/
* Köhntopp, K. (2021, January 4). *Fork, exec, wait and exit*. Percona. https://percona.community/blog/2021/01/04/fork-exec-wait-and-exit/
* (N.d.). https://pubs.opengroup.org/onlinepubs/009695299/basedefs/sys/wait.h.html