CIS 415 Operating Systems

Project 1 Report Collection

Submitted to:

Prof. Allen Malony

Author:

*Nico Sicat*

*nsicat*

*951941219*

**Report**

**Introduction**

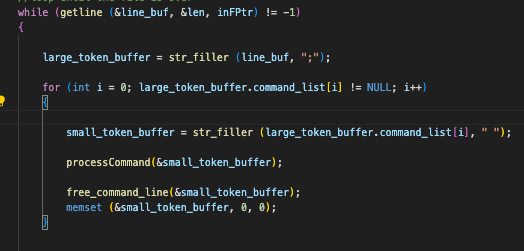
This report documents the development and implementation of Project 1, a simplified Unix shell designed to process basic Unix commands and interact with the operating system. The shell's primary functionality encompasses file navigation and input/output operations, supporting a range of fundamental Unix commands, including ls, pwd, mkdir, rm, cd, and cat. The shell operates in two distinct modes. Interactive mode mimics the behavior of a traditional shell, allowing users to input and execute commands in real-time. Conversely, File mode enables batch processing, where the shell executes a sequence of commands listed in a file, which is similar to the functionality of the Bash shell.

**Background**

The project mainly depends on having a solid understanding of Unix system calls. System calls facilitate communication by acting as a link between an application and the operating system kernel. For this project, we did not use high-level functions like printf statements and execvp; instead, we implemented low-level commands using system calls like readdir, chdir, mkdir, and others. The project also uses malloc() to allocate memory in order to handle user input. Malloc() was chosen because it prevents memory leaks by enabling memory to be allocated and released in an appropriate manner. Valgrind was used to confirm that there were no memory leaks.

**Implementation**

The implementation involved creating the shell in C, with two key modes: interactive and file mode. The project was structured with a main.c file for managing the command line interface and execution logic, and a command.c file for defining the actual functions like ls, pwd, cd, and file handling commands. There is also string\_parser.c which handles the tokenization of the arguments.



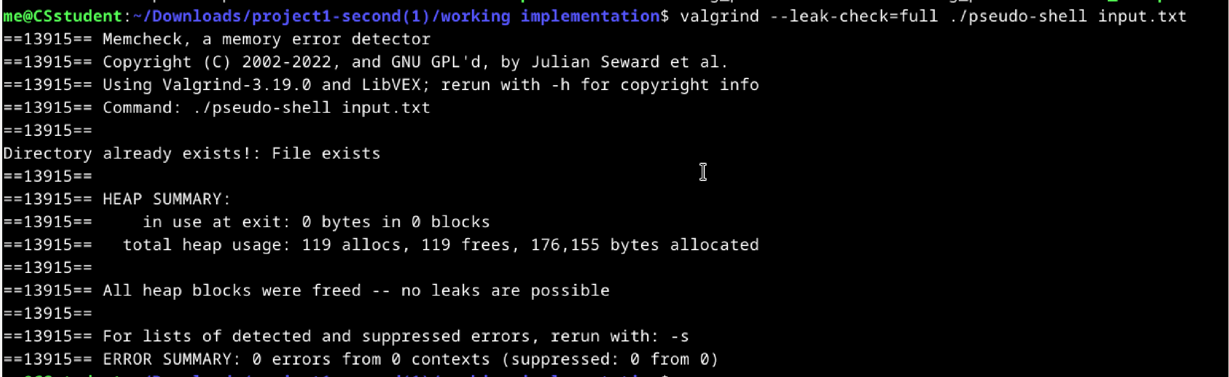
Getline() was used to read user input for input processing, and strtok() helped with tokenization and command parsing. This made it possible for the shell to efficiently decompose complicated commands into executable parts. This snippet calls upon str\_filler in string\_parser.c



In file mode, all outputs were redirected to an output.txt file rather than the console. This was achieved by manually opening the file for writing and redirecting standard output (STDOUT) to this file. Specifically, stdout was reassigned to the file pointer. After execution is over, stdout is reverted to its original state.



Valgrind was ran in multiple occasions to see if there are memory leaks in the program.



**Performance Results and Discussion**

The pseudo-shell performed effectively in both file and interactive modes, successfully executing implemented commands (ls, pwd, mkdir, cp, and mv) using solely system calls. Valgrind verification confirmed efficient memory management, detecting no memory leaks. Robust error handling enabled the shell to gracefully manage edge cases, such as incorrect file paths and spaces in command input, while processing multiple commands quickly and efficiently.

Although the pseudo-shell successfully executed essential functions, it encountered issues with the supplied bash test script, specifically failing the file mode test. Despite efforts to address this, the issue persisted. While manual testing revealed no memory leaks, potential problems with other functions may be contributing to the test failure. Due to time constraints, further investigation was not possible. Nonetheless, the pseudo-shell passes all tests except the file mode test, demonstrating its overall effectiveness.

**Conclusion**

This project offered insightful information about how to visualize working with system-level programming, particularly regarding the calls made by our version of Unix commands to carry out the desired actions. Not only did it help me see what to look for, but it also helped me get better at handling errors, parsing strings, and managing my memory. All in all, this project was a lot of fun and gave you some basic C operating system knowledge.