Project 3: Improved UNIX/Linux Command Line Interpreter

Nolan Tuttle & Mathew Hobson

College of Engineering and Technology, Grand Canyon University

CST-315: Operating Systems Lecture and Lab

Professor Citro

February 23, 2025

Updated Description

The previous project involved creating a basic Unix/Linux command-line interpreter capable of processing user input and executing commands. The interpreter supported batch file execution and single command execution. However, it lacked advanced features such as directory navigation (cd command), and making/deleting directories.

Methodology/Approach

1. Refactored Code Structure: Slightly improved initial user prompts and readability.
2. New Features:
   - Change Directory (cd): Implemented the ability to change directories.
   - Directory Creation (mkdir): Implemented to allow for folder creation using execv() so that they can be navigated using cd.

Algorithm for Parsing and Processing Shell Commands

1. Read user input.
2. Remove trailing newline characters.
3. Check for special characters (cd for directory changes, | for pipes).
4. Tokenize the input based on space and semicolon (;).
5. Handle cd command by invoking chdir().
6. Handle background commands by using fork().
7. Execute commands using execv() instead of system() for better security and efficiency.

8. Repeat until user exits the shell.

Code and Comments

Handling Batch Files

```
if (argc == 2) // Condition for batch file usage
{
    char *batch_file = argv[1];
    FILE *file = fopen(batch_file, "r");
    char str[100];


    while (fgets(str, sizeof(str), file))
    {
        str[strcspn(str, "\n")] = 0; // Remove newline character
        if (str[0] == '\0' || str[0] == '#') continue; // Skip empty lines/comments
        system(str); // Execute batch command
    }
    fclose(file);
}
```

Processing Input

```
while (isRunning)

        {

                printf("seaShell$ ");                  // Command line prompt

                fgets(str, sizeof(str), stdin); // Read input, including spaces

                str[strcspn(str, "\n")] = 0;     // Remove newline character



                // Condition for user inputting "CTRL+X" or the "quit" command

                if (str[0] == 24 || (!strcmp(str, "quit")))

                {

                        exit(1); // Ends execution

                }

                // Condition for user inputting "CTRL+B" or the "exit" command

                else if (str[0] == 2 || (!strcmp(str, "exit")))

                {

                        isRunning = 0;
                        break;

                }
```

Command Execution

```
        // splits up the input into semicolon-separated tokens

        char *command = strtok(str, ";");


        while (command != NULL)

        {

                if (command[0] == 'c' && command[1] == 'd')

                {

                        char *temp = strtok(command, " ");

                        temp = strtok(NULL, " ");

                        chdir(temp);

                        break;

                }


                char *args[] = {"/bin/bash", "-c", command, NULL};


                pid_t pid = fork(); // create a child process


                if (pid == 0)

                {
```

```
            if (execv("/bin/bash", args) == -1)

            {

                perror("exec failed");

                exit(1);

            }

        }

        else if (pid > 0)

        {

            int status;

            pid_t childPID = waitpid(pid, &status, 0);

        }

        command = strtok(NULL, ";");

    }

}
```

References

Stallings, W. (2019). *Operating Systems: Internals and Design Principles* (9th ed.). Pearson.

Tuttle, N., Hobson, M., (2025) *CST315* GitHub. https://github.com/nolantuttle/CST315.git