Custom Alpine Shell

# Abstract

The Custom Alpine Shell is a shell written in C, which allows for easier access of the filesystem.

# Introduction

For the project, we have chosen to write a Custom Linux Shell. Our goal was to implement a shell that has many own features and thus stands out from other common shells. We have therefore decided to focus on file systems and their management. We wanted to optimise the use of file systems with our own features. Apart from that, our shell should still have the basic functionality of a conventional shell. This includes starting processes on the kernel and executing applications.  
  
We used C as programming language. We decided to do this because we already had experience with C. Furthermore, it is a standard Linux language.

Finally, we had to choose the Linux distribution where we wanted to implement our project. Our choice fell on Alpine Linux.  
Alpine Linux is a terminal-based Linux distribution. We have already worked with Alpine in another lecture. We found the management of files in Alpine very cumbersome and confusing. We wanted to make this easier with our shell.

# Implementation

## Command Parser

As Alpine does not use a Graphical User Interface, all user commands must be accessible by the command line. This means that the Command Line Input must be correctly interpreted and parsed.

-split-make commands

After a line is entered, the input is first split by the whitespaces, and the first argument is converted to the corresponding command, with not defined inputs being converted to the “error” command, which then displays an error to the user. The rest of the input is saved as nodes inside the Linked List.

-execute commands

After the input is converted into the Linked List, it is executed by finding the matching command case. These command cases first evaluate, whether the length of the Linked List is correct, win that case the execution of the command starts, otherwise an error is shown to the user and the execution stops.

-free memory

## Directory Management

For the directory management it was important to keep the ease of use in mind. Thus, the files and directories inside of the current working directory are saved inside Linked Lists. When using commands, instead of having to specify the filename or directory name, it is accessible by using the unique integer identifier of the corresponding Linked List.

These identifiers are displayed on the screen alongside their corresponding file/directory. Whenever a new directory is accessed, a new corresponding Linked List is generated, and the old one deleted.

## Command Implementation

### Showfiles

### Show

### Showdirectories

### Move

When the move command is called without further arguments, the files inside the current directory are displayed to the user, alongside the identifiers of these files. The following user input is then evaluated, and the complete file path is saved internally. Then all the subdirectories are displayed with their identifiers, with which the user can navigate through the directories. Once the final directory is chosen, the new path is saved internally, and the file is moved using these saved parameters.

### Copy

Conceptually it works the same as move, except that instead of being able to directly move a file using a system call, first a new file is created at the destination, into which the content of the first file is written into.

### Rename

### Go

By calling go, all the directories inside the current directory are displayed. When a number is typed, the current working directory is changed to the corresponding directory.

### Run

### Delete

Delete can be called by directly specifying a filename or without. If it is called without a argument, then the files inside the current working directory are displayed, and the file can be chosen by specifying the identifier. The specified file is then deleted, with an error being displayed if it is not possible.

### Help

## (Log)

# Results?

# Discussion

## Problems

-Windows-Linux-Alpine-Segmentation errors

There were several problems which had to be overcome for the project. The most common ones were cause by memory leaks as well as differences between the used operating systems, sometimes both.

An often-occurring problem was a segmentation error, caused by a difference in return values between Windows and Linux. Finding the actual bug was difficult, as the error itself didn’t occur at the actual spot, where the variable is accessed, but later when a print statement was made.

-C in general -memory leaks

The lack of already implemented dynamic datastructures in C such as an Array List meant that it was needed to implement things manually, most prominently the Linked List. As C does not have a garbage collector, wrong implementations could lead to memory leaks, which however could be resolved with the help of a debugger.

## Lessons Learned

## Division of Labour

At the beginning of our project, after research and design we worked separately on the directory management and parser. However, it quickly proved more efficient to work together more closely, as it was difficult to connect our different methods.

Research: Both

Design: Both

Directories: Matthew (+Elia)

Parser: Elia (+Matthew)

Commands: Both

Debugging: Both

Report: Both

# Conclusion

A fully functional Shell for Alpine was written in C, which allows for easier manipulation of the file system, including renaming, moving copying and deleting of files. Any functionality of the normal shell which are not implemented in the custom shell can be accessed with an integrated system call.