Unix Shell Implementation Project Report

Suramya Angdembay

April 18, 2025

Abstract

This report outlines the development of a Unix-like command-line shell named `osh`, written in the C programming language. The shell replicates essential features found in common Unix shells, including command execution, input/output redirection, piping, background processes, and command history. The implementation of these features provided an opportunity to gain hands-on experience with low-level system programming and Unix system calls.

Problem Description

The objective of this project was to implement a shell that can interpret and execute user commands in a Unix-like environment. The shell should support the execution of both built-in and external commands, manage foreground and background processes, and allow redirection of input and output streams. Additionally, features such as command history and inter-process communication via pipes were required. The ultimate goal was to deepen our understanding of how real-world shells function at the system level.

Algorithm (Pseudocode)

```
Initialize last_command as empty

while true:
    display "osh> " prompt
    read user input

    if input is empty:
        continue

    if input == "exit":
        break
```

```
if input == "!!":
    if last_command is empty:
        print "No commands in history"
        continue
    else:
        input = last_command
else:
    last_command = input

check for '|' in input:
    split input into cmd1 and cmd2
    parse cmd1 into args1
    parse cmd2 into args2
    create pipe
    fork child1:
        if child1:
            redirect stdout to pipe
            execvp(cmd1)
    fork child2:
        if child2:
            redirect stdin to pipe
            execvp(cmd2)
    parent:
        close pipe
        wait for both children
    continue loop

check for '<' or '>' in input:
    set redirect_input / redirect_output and filename

parse input into args

if args[0] == "cd":
    if args[1] exists:
        chdir(args[1])
    else:
        print error
    continue

check for '&' at end of args:
    set background = true and remove '&'

fork child:
    if child:
        if redirect_input:
```

```
            redirect stdin to filename
        if redirect_output:
            redirect stdout to filename
        execvp(args[0], args)
    else:
        if not background:
            wait for child
```

Implementation Details

The shell was implemented in C and compiled using GCC. The main source file is located in `src/osh.c`, and the compiled binary is generated in the `bin/` directory using a Makefile. Key system calls used include `fork()`, `execvp()`, `wait()`, `dup2()`, `pipe()`, and `chdir()`. The shell includes support for built-in commands such as `cd` and `exit`, as well as special features like command history recall using `!!`, input/output redirection with `<` and `>`, piping using `|`, and background process execution using `&`.

Running Results and Analysis

The shell was tested with various commands to verify its functionality. As shown in the figures below:

- Commands like `ls | grep .c` and `cat < file.txt` execute as expected, demonstrating correct piping and input redirection.
- The command `echo hello > file.txt` successfully writes to a file, which can then be read back using `cat`.
- The history feature is validated using `!!`, which re-executes the most recent command.
- Background execution is shown to work using `sleep 5 &`, where the shell prompt returns immediately without waiting.
- Commands like `cd src` correctly change directories, indicating proper handling of built-in commands.

Conclusions and Learning

This project was instrumental in helping me understand how shell environments operate under the hood. Implementing process control, command parsing, and inter-process communication gave me practical insight into how system-level programming works on Unix-based systems. It also strengthened my understanding of C programming, especially with regard to pointers, memory handling, and working with low-level system calls. This hands-on experience has deepened my appreciation for how sophisticated and efficient Unix shells are.

Figure 1: Demonstrating piping, redirection, history, built-in commands, and file navigation.

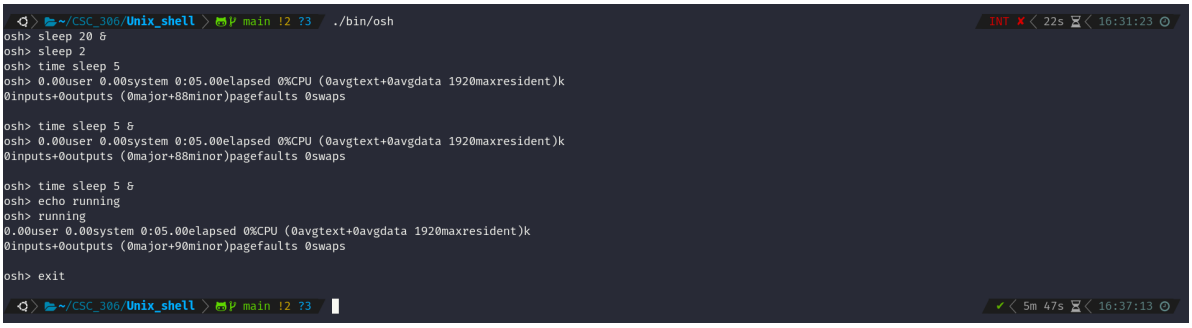

Figure 2: Demonstrating background execution using `sleep` with and without `&`.

References

- Prakash A. (2010). *How to Write, Compile and Run a C Program in Ubuntu and Other Linux Distributions*. https://itsfoss.com/run-c-program-linux/