Sarah's Shell

And Commands

Sarah Higgins
Department of Computer Science and Networking
Wentworth Institute of Technology
Boston, MA 02115, USA
{higginss1}@wit.edu

***Abstract* – The purpose of this project is to build a Shell and commands that the Shell will accept in the programming language C. A Shell is a terminal that accepts commands from the user keyboard and sends those commands to the operating system to perform. Commands are what the user types into the terminal command line to allow the Shell to navigate through the operating system's directory components.**

***Keywords* –** *Shell, command*

I. INTRODUCTION

Have you ever wondered if you could be the mastermind behind your computer and its capabilities? What if that was entirely possible through the Linux Operating System, and you just simply never knew that such a thing was possible? Well not only is it possible and already exists, but building such a tool is also possible!

With programming, the user is able to build from scratch this exact tool with the same functionality – a Linux-like Shell. The Shell is a program for the user and the Operating System to perform all the things that the user wants it to do. The Shell is an ordinary program that reads commands from the user and executes them, and is the primary user interface to traditional Unix-like systems [1].

With the right commands, the Shell allows the user to get a broader, more detailed version of what the GUI, or graphical user interface, provides on the surface of the underlying details. The Shell is a helpful program that allows the user to navigate the Operating System's directories, their components and details, and what they contain. Details of a directory can be as precise as telling the user the names of all of the files within a directory to the dates that those programs were created.

I. PROBLEM

A. *Definition/ Problem Formulation*

Building a functional replica of the Linux Shell requires knowledge in a few areas. First, knowing the Linux Shell and how to navigate it with the proper commands is a fundamental requirement. Second, knowing a programming language that will allow the user to implement this functionality is a must. In the case of this project, the programming language used is C. Fig. 1 is a screenshot of the completed, implemented Shell written in C.

C is a great programming language for this because it easily allows for the use of opening, reading from, writing to, and closing files and directories with the proper include and define statements at the top of the program, and easily allows the programmer to work with file descriptors that access different areas of the files being worked with, depending on which file descriptor the user chooses. In short, the file descriptor is the gateway into the kernel's abstractions of underlying hardware [2].

B. *Sub-Problems*

In addition to the Shell, implemented commands for the user to execute and access are essential to the program functionality. If the Shell does not execute commands the user types in, then it is not a proper Shell. It wouldn't have any functionality, and the user would get frustrated with the lack of performance.

In order to have the shell work appropriately, it executes basic commands. These commands are ls, pwd, head, tail, cat, and cp. These commands allow the user to list the contents of the current working directory, display the current working directory name, display the beginning part of a file, display the ending part of a file, concatenate files, and copy files, respectively.

Fig. 2 is a screenshot of the completed, implemented ls command as a command example written in C, and Fig. 3 shows the successful output upon the completion of the command being executed.

III. SOLUTION

Provide your findings or solutions of your problem/ projects.

A. *Abbreviations and Acronyms*

ls: Linux command to list the files of the current working directory. Arguments can be added, or can be typed into command line by itself.

pwd: Linux command to display the current working directory. Typed into command line by itself.

head: Linux command to display the beginning of a text file or piped data. To display correctly, syntax is typed in as:

head [-number of lines] [file name]

tail: Linux command to display the ending of a text file or piped data. To display correctly, syntax is typed in as:

tail [-number of lines] [file name]

cat: reads a file from beginning to end, and can read files sequentially if given more than one file name as its arguments.

cp: copy one file to a new destination. To display correctly, syntax is typed in as:

cp [file name] [copied file destination]

B. *Figures and Tables*

```c
/*
 * Sarah Higgins
 * Operating Systems Final Project
 * SarahShell.c is a mini shell that closes with CTRL"C".
 * Performs ls, ls -l, pwd, cat, cp, head, tail, vi, ping,
 *          fdisk, whoami, echo, and more!  It does not,
 *          however, perform cd because cd is built into the
 *          Linux shell already and is not a binary.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/wait.h>
#define CMD_LENGTH 100
#define TOKENS 10

/*
 * Main function runs the program
 * @param int argc
 * @param char* argv[]
 *
 */
int main(int argc, char* argv[]) {
  long line_size = CMD_LENGTH;
  char* input_line;
  input_line = (char*) malloc(CMD_LENGTH + 1);


  char* command[TOKENS];
  int cmd_pid;
/*
 * Void function command_tokenize
 * ------------------------------
 *   char** command: pointer of the String command from the user
 *   char* command_text: String of command_text from the user
 *
 *   Breaks string into a series of tokens
 */
void command_tokenize(char** command, char* command_text) {
  char* token = strtok(command_text, " ");
  int token_index = 0;
  while(token != NULL) {          /* while SarahShell has a command */
    command[token_index] = token;
    token = strtok(NULL, " ");
    token_index++;
  }

  char * last_token = command[token_index -1];  /* the last token of command */
  int length = strlen(last_token);
  last_token[length - 1] = 0;
  command[token_index] = 0;
}
```

Fig. 1 SARAH'S SHELL SCREENSHOT

```c
/*
 * Sarah Higgins
 * ls.c
 * This program recreates what the ls command does within the Linux terminal.  The
 * ls command lists the files within the current working directory.
 */

#include <dirent.h>
#include <stdio.h>

int main(int argc, char* argv[]) {   /* finds the working directory */
  struct dirent *entry;
  DIR *d = opendir(".");

  while ((entry = readdir(d)) != NULL) {    /* while entry is not null */
    if (entry->d_name[0] != '.') {          /* print the programs within working direcory */
      printf("%s ", entry->d_name);
    }
  }
}
```

Fig. 2 LS COMMAND IMPLEMENTATION

```
sarah@sarah-Satellite-M645 ~ $ ./SarahShell
|SarahShell> ls
add.c     copy.c              Downloads  lab2     Pictures    SarahShell.c
Arduino   cp.c                file1.txt  lab2.c   projectsSH  tail.c
cat.c     Desktop             FlashNe    ls       pwd.c       wall.c
cd.c      discord-canary.deb  head.c     ls.c     pyramid.c   watcher.c
config    Documents           lab1.c     msh.c    SarahShell  write.c
|SarahShell>
```

Fig. 3 LS COMMAND OUTPUT

IV. CONCLUSION

Building a Linux Shell and seeing it work is not only satisfying, it's fun! Testing the Shell to ensure that it gives the proper feedback not only gives the user a chance to confirm their creation of the Shell, but to be even more aware of the structure of their computers' directory tree, its programs and contents, and the Shell's capabilities in showing the user what it contains.

Although C may not be the most widely used programming language in the working world, it's a language that has great capability to help the user become much more familiar with their computer and operating system. Building a Linux Shell may seem complex, but it's not as difficult as one may seem. With the right tools and adequate level of knowledge, building a Shell or other programs in C is entirely possible!

REFERENCES

[1] R. Cox, F. Kaashock, R. Morris, A Simple, Unix-like Teaching Operating System: xv6-book@pdos.csail.mit.edu

[2] Ian Wienand, Computer Science From The Bottom Up: https://www.bottomupcs.com/file_descriptors.xhtml