PA2 Report: Implementing a Linux Shell

Gregory Petri
CSCE 313-501
Dr. Tanzir Ahmed
5 March 2021

Introduction

Programming Assignment 2 required the creation of a custom Linux OS shell. Directory navigation, file redirection, and piping were a few challenges of creating a functional shell. This report will detail the methods (designs and algorithms) for smooth shell operations.

Designs and Algorithms

I. Quotes

The shell user can enter commands, such as echo and awk, which do not recognize parameters in quotes with "execvp()." Additionally, the parser should only separate a user input into pipes if the pipe character ("|") is not within quotations. The algorithm's steps are as follows:

1. Store quote matches of the whole input string in a vector:
    a. The function "find_quotes()" will find the first quote (") or tick (‘), set the flag "in_quote" to true, and set either the quote or tick starting index variable to the current iteration index of the string. If a matching quote or tick character is found while "in_quote" is true, then the quote or tick starting index and current iteration index are pushed to the vector.
2. Split the user input by pipes and push to vector:
    a. The pipe splitter function separates the user input into pipe string segments, and pushes the segments into a commands vector. Iterating through the input string, if the current character is a pipe symbol, then "pipe_split()" checks "inquotes()". The function "inquotes()" that iterates by two through the "quotes" vector, and checks that the current index is within any pair of quote vector entries. If "inquotes()" returns true, then the user input is not split at that index.
3. Store quotes for a command, split the command by spaces, and remove leading and trailing quotes:
    a. After the "commands" vector is fully populated, each entry of "commands" is sent to the "find_quotes()" function again. This command is split at an empty space if "inquotes()" returns false. Additionally, every split string is checked for leading and trailing quotes or ticks with the function "remove_endquotes()". If leading and trailing quotes or ticks are found, then they are stripped.
4. Push command splits to the vector "parts":
    a. Finally, the split strings from the command are pushed to the "parts" vector.

II. Vector of Strings to Character Array

Arguments to "execvp()" are required to of const char pointers or a char array data types. This can be achieved via the function "vector_to_char_array()":

1. Create a char array pointer on the heap of size "parts.size()" vector + 1

2. Loop through the “parts” vector of strings and in each iteration:
    a. Create a char pointer in the char array pointer on the heap.
    b. String copy the current string from “parts” to the char pointer.
3. Set the last index of the char array to “NULL”, then return the char array pointer.

III. History

A history of commands passed to the shell is often desired by users. This functionality is included in this shell. History is implemented as follows:

1. Put all past commands into a “past_commands” vector:
    a. When “getline()” receives a user input string, the trimmed input is pushed into a vector for storage.
2. Print the “past_commands” vector:
    a. If “history” is passed at the first position of the user input (without leading whitespace), then the last vector entry (history) is popped. A loop through the “past_commands” vector ensues, which prints each index and string entry.

IV. Clear

Clearing the screen is necessary for a kernel so users can minimize clutter. Clear is implemented as follows:

1. Check for the “clear” command in the same manner that the “history” command was checked.
2. Clear the screen by printing the escape sequence “\033[H\033[J” to standard output and exiting the for loop.

Video

https://www.youtube.com/watch?v=pomX3aKq70U

Conclusion

Millions of people use the Linux shell daily, but many of these users do not fully understand the fundamental behind-the-scenes operations of the shell. This programming assignment helped me furnish a deeper understanding of the Linux shell functionality. This knowledge will truly help me write code and execute processes with more foresight of the output.

References

Get User Login Name: https://linux.die.net/man/3/getpwuid
Get Current Time:
https://stackoverflow.com/questions/16357999/current-date-and-time-as-string/16358264
Parsing Tokens: https://www.geeksforgeeks.org/tokenizing-a-string-cpp/
Clear Screen: https://www.geeksforgeeks.org/making-linux-shell-c/

Appendix

I. Quotes – find_quotes()

```
void find_quotes(vector<int> &quotes, string input)
{
    int quote_st = -1;
    int tick_st = -1;
    bool in_quote = false;
    //char quote_type;
    unsigned char p;
    // check for quotes
    for (int i = 0; i < input.size(); i++)
    {
        p = (unsigned char)input[i];
        if (p == '\"')
        {
            if (!in_quote)
            {
                //quote_type = '\"';
                in_quote = true;
                quote_st = i;
            } else if (in_quote && quote_st != -1)
            {
                quotes.push_back(quote_st);
                quotes.push_back(i);
                quote_st = -1;
                in_quote = false;
            }
        } else if (p == '\'')
        {
            if (!in_quote)
            {
                //quote_type = '\'';
                in_quote = true;
                tick_st = i;
            } else if (in_quote && tick_st != -1)
            {
                quotes.push_back(tick_st);
                quotes.push_back(i);
                tick_st = -1;
                in_quote = false;
            }
        }
    }
}
```

I. Quotes - split_pipes()

```
// PARSE COMMAND
vector<string> split_pipes(string input)
{
    vector<string> commands;
    vector<int> quotes;
    find_quotes(quotes, input);
    int quote_idx = 0;
    //for (const auto i : quotes)
    //{
    //  cout << i << ',';
    //}
    // split by pipes
    string curr_string = "";
    for (int i = 0; i < input.size(); i++)
    {
        if (input[i] == '|' && inquotes(quotes, i))
            curr_string += input[i];
        else if (input[i] != '|')
            curr_string += input[i];
        if (i == input.size() - 1)
        {
            commands.push_back(trim_string(curr_string));
            curr_string.clear();
        }
        else if (input[i] == '|' && !inquotes(quotes, i))
        {
            commands.push_back(trim_string(curr_string));
            curr_string.clear();
        }
    }
    return commands;
}
```

I. Quotes – split()

```
// SPLIT
vector<string> split(string line, vector<int> quotes, char delimiter=' ')
{
    vector<string> vec;
    string curr_string = "";
    bool in_brackets = false;
    for (int i = 0; i < line.size(); i++)
    {
        curr_string += line[i];
        if (i == line.size() - 1)
        {
            vec.push_back(remove_endquotes(trim_string(curr_string)));
        }
        else if (line[i] == delimiter && !inquotes(quotes, i) && !in_brackets)
        {
            vec.push_back(remove_endquotes(trim_string(curr_string)));
            curr_string.clear();
        }
    }
    return vec;
}
```

I. Quotes – inquotes()

```
// CHECK IF INPUT INDEX IS IN QUOTES
bool inquotes(vector<int> quotes, int idx)
{
    for (int i = 0; i < quotes.size(); i += 2)
    {
        if (idx >= quotes[i] && idx <= quotes[i+1])
            return true;
    }
}
```

I. Quotes – remove_endquotes()

```
// REMOVE ENDQUOTES
string remove_endquotes(string str){
    if (str[0]  == '\'' && str[str.size()-1] == '\'')
        return str.substr(1,str.size()-2);
    else if (str[0] == '\"' && str[str.size()-1] == '\"')
        return str.substr(1,str.size()-2);
    return str;
}
```

II. Vector of Strings to Character Array – vec_to_char_array()

```
// VECTOR ARRAY TO CHAR ARRAY
char** vec_to_char_array(vector<string> vec)
{
    char** arr = new char*[vec.size()+1];
    for (int i = 0; i < vec.size(); i++){
        if (trim_string(vec[i]).find("$PATH") == 0){
            arr[i] = new char[PATH.size()+1];
            strcpy(arr[i], PATH.c_str());
        } else{
            arr[i] = new char[vec[i].size()+1];
            strcpy(arr[i], vec[i].c_str());
        }
    }
    arr[vec.size()] = NULL;
    return arr;
}
```

III. History

```
} else if (trim_string(inputline).find("history") == 0) // see command history
{
    past_commands.pop_back();
    for (int i = 0; i < past_commands.size(); i++)
    {
        printf(" %x %s\n", i, past_commands[i].c_str());
    }
    exit(1);
} else if (trim_string(inputline).find("clear") == 0) // clear screen
```

IV. Clear

```
} else if (trim_string(inputline).find("clear") == 0) // clear screen
{
    printf("\033[H\033[J");
    exit(1);
} else{
```