Patel Harshank
Dr. Ahmed
UIN: 527009145

Assignment Report

CSCE-313-512

PA2 Implementing a Linux Shell

Objective: In this assignment, we implemented our very own Linux shell. My shell has the ability to function almost as much as the Linux/ubuntu shell in Linux OS, which lets a user navigate through the file system and perform a wide variety of tasks using a series of easy to remember and simple commands.

VIDEO PRESENTATION: ALREADY DEMOED TO TA (PIN LYU) IN LAB CSCE-512 ON THURSDAY MARCH 4, 2021 (He said I got a 100 on it)

Extra Credit That Works:

The only extra credit that works with my code is mkdir or make directory:

mkdir a

The code outputs a directory with the user inputted.

I believe I should get at least 1 point for it. Thanks!

Procedure: The assignment was divided into two steps, the first step in the assignment was to get the string manipulation to work. I started with by the help of professor, and I learned that the pipe function has the top priority in splitting the given input string. So, I began to write a split function that takes in an argument as a delimiter and then splits the string. Second step was to implement a trim function which trims the extra white space in a given input string. So, how my code works is by taking a for loop and splitting the given input string by the pipes and then I split the string by the white space and then I convert the vector of strings to a char array. The code is shown as follows:

Regular Split Function:

```
vector<string> split(string str, char delim) {
    vector<string> out;
    size_t start;
    size_t end = 0;
    while ((start = str.find_first_not_of(delim, end)) != string::npos) {
        end = str.find(delim, start);
```

```
            out.push_back(str.substr(start, end - start));
        }
        return out;
}
```

Regular Trim Function:

```
std::string trim(std::string str) {
        while (!str.empty() && std::isspace(str.back()))
            str.pop_back();
        std::size_t pos = 0;
        while (pos < str.size() && std::isspace(str[pos]))
            ++pos;
        return str.substr(pos);
}
```

Regular Vector to Char Array:

```
char **vec_to_char_array(vector<string> &position) {
        char **result = new char *[position.size() + 1];
        for (int i = 0; i < position.size(); i++) {
            result[i] = (char *)position[i].c_str();
        }
        result[position.size()] = NULL;
        return result;
}
```

Second function that I am proud about is the implementation of the SPECIAL AWK function that needs to be split by without touching the string in between the { } brackets. So therefore, the regular split couldn't have worked in this case because the split function would have split the split the contents in between the brackets as well. Therefore, the implementation for this function is a bit different.

Special AWK Split:

```
vector<string> specialAWKsplit(string str){
        vector<string> ans;
        std::size_t found = str.find("{");
        if (found != std::string::npos){
            int pos1 = str.find('{');
            vector<string> temp = split(str, '{');
            vector <string> temp2 = split(temp[0],'{');
            vector <string> bothPartsLeftRight = split(temp2[0],'');
            string leftSide = bothPartsLeftRight[0];
            string rightSide = bothPartsLeftRight[1] + "{" + temp[1];
            ans.push_back(leftSide);
            ans.push_back(rightSide);
        }
        else{
            vector<string> temp = split(str, '');
```

```
        for(int i = 0; i < temp.size(); i++){
            ans.push_back(temp.at(i));
        }
    }
    return ans;
}
```

Another issue was to remove the quote from the input string because the input string was given something like this:

```
ps aux | awk '/init/{print $1}' | sort -r
```

Therefore, there was a need to remove those single quotes from the given input string. My function as shown below removes those single quotes and returns the same input string un-harmed except those single quotes.

Special Quote Removal:

```
std::string awkQuotesRemoval(string input){
    string ans = "";
    for(int i = 0; i < input.size(); i++){
        if(input.at(i) == '     \'     '){
            //skip because its okay!
        }
        else{
            ans += input.at(i);
        }
    }
    return ans;
}
```

Another function I am happy to share is the function to implement the “cd -”, which keeps toggling between the previous directory and the current directory you visited.

Part of my code for “cd -”:

```
if (trim(inputline).find("cd ") == 0) {
    //So Dash is fouond!
    if (inputline.find("-") == inputline.size()-1) {
        char currWorkingDirectory[10000];
        getcwd(currWorkingDirectory, sizeof(currWorkingDirectory));

        //Prev in the dash is same
        if(strcmp(currWorkingDirectory,prevWorkingDirectory) == 0 ){
            cerr <<"No Change Made because there is no previous directory in the history\n";
        }
        //Prev in dash is not same! Interchhange them
        else{
            char currWorkingDirectory[10000];
            getcwd(currWorkingDirectory, sizeof(currWorkingDirectory));
            chdir(prevWorkingDirectory);
            strcpy ( prevWorkingDirectory,currWorkingDirectory);
```

```
            }
        }
        else{
            string dirName = trim(split(inputline,' ')[1]);
            char currWorkingDirectory[10000];
            getcwd(currWorkingDirectory, sizeof(currWorkingDirectory));
            chdir(dirName.c_str());
            if(strcmp(currWorkingDirectory,prevWorkingDirectory) == 0 ){
            }else{
                strcpy(prevWorkingDirectory,currWorkingDirectory);
            }
        }
        continue;
    }
```

Conclusion: The project works as per the instruction and I believe the TA said I will get a 100 after a demo. I particularly enjoyed this assignment and I hope the assignments gets easier after this one.