CS6456 (F2015) Operating Systems

Project 2: Writing Your Own Shell

Write a simple Linux shell that can (i) run programs and (ii) support commands for file redirection and pipes.

1. How to orgonize and parse the input.

At first, I use char* cmd[][] as the structure of input, every cmd[] is a token group, and each cmd[][] is a command or argument. It works, however it's hard to copy value and when it comes to loop, it becomes too complicated to use. So I change to use struct to save and parse the input. I use struct Command to represent the users ' whole input, and use struct Subcomand to represent every token group devided by '|' character.

Command has two elements, including the Subcommend, which is also struct type, and number of Subcommands.

The struct SubCommand has two elements .Line is the content of the SubCommand. Stdin_redirect is the name of the file after '<', and the stdout_ redirect is the name of file after '>'. Both value can be NULL in the case of a<b or a>b; argv[] is a array saving all the arguments in this token group. Considering allocate space, I set a MAX_ARGS == 10 for the argv[]. Pid is an intiger used to return the status of very child process. here is what the two struct look like:

```
struct Command
{
    struct SubCommand sub_commands[MAX_SUB_COMMANDS];
    int num_sub_commands;
};

struct SubCommand
{
    char *line;
    char *argv[MAX_ARGS];
    char *stdin_redirect;
```

```
    char *stdout_redirect;
    int pid;
};
```

2. Functions

I use several function to realize parse input, checking input, execute command, realize file redirection and so on. The following is the detail explanation of each functions.

ReadArgs(char* input, char** argv, int size)

1. Devide every token into single command and argument and save as an array argv[].
2. Check whether each argument has invalid characters which is not A-Z, a-z, 0-9, dash, dot, forward slash, or underscore.

ReadRedirects(truct Command *command)

1. check whether the input has two same file redirection, make sure that the “in” redirection is followed by an filename(not command)and “out” redirection follows a filename(not command).
2. Check whether the filename has invalid characters which is not A-Z, a-z, 0-9, dash, dot, forward slash, or underscore.

ReadCommand(char *line, struct Command *command)

1. Divide input into Subcommands(i.e. token groups), separated by “|”.
2. Call readArgs() to divide command(token groups) into arguments and to check whether the arguments has invalid characters.
3. Delete “\n” at the end of the input. Fgets(char * str, int num, FILE * stream) function return a string with ‘\n’, so ‘\n’ must be set as ‘\0 ’, otherwise linux can’t find the file with a name including’\n’ or excute a order with a argument including’\n’.
4. The use of loop with the function of strtok() is very helpful, however there may comes out some tricky bugs, for example the return values of strtok point to the same address, so we must use strdup to copy the string,otherwise all the arguments will be same when the loop finish.

Print_args(char** argv)

Just to print arguments in standard format.

PrintCommand(struct Command *command)

Call print_args() to print the whole command(token groups) in standard fomat.

ExecuteCommand(struct SubCommand *sub_command)

1. open the the files that need to be redirected and report error if the shell cannot open them.
2. Execute the each subcommand and report error if the execution failed.

ChangeDirectory(struct SubCommand* sub_command)

1. Since I didn't get what the instructor mean about the path part, I add a function that supports change directory, you can try it just for fun(enter: cd "your desired pathname") .

CheckValidInput(char*command)

1. Check whther the use of "|" is correct(there should be " " before and after "|")
2. Check invalid characters roughly, this checking allows "<", ">", "|" , which is not in arguments checking function.

runWholeCommand(struct Command* command)

1. If the command only has one subcommand, it may use file redirection in it. Fork a child process for it and call excuteCommand function to execute it. At last, shell waits the return value of child precess and print status of it.

2. If the command has several subcommands, it may use pipe, fork a pipe for each subcommand except for the last one. Another problem that need to be notice is the first subcommand only need to use pipe as output. What really important is closing one side of pipe before the other side is used. At last, the shell wait the child process and print the status of it.

3. The process of running a command.

1. Get the input using fgets().
2. Retrun 0 if users enter end-of-file.
3. Check whether the input is over 100 characters. Report error if input over 100 characters
4. Check the whether the input has invalid characters roughly and check whether the use of "|" is correct.
5. Devide input into subcommands and then divide subcommands into arguments, check whether each argument has valid charaters strictly.
6. Check whther the file redirection "<" is followed by a filename not command. check whether file name has invalid characters and "out" redirection follows a filename(not command).
7. When shell encounters "exit", terminate itself.
8. When shell encounters "cd pathname", go to that path.
9. Run the command with file redirection and pipe.