CIS 415 Operating Systems

Assignment 1 Report Collection

Submitted to:
Prof. Allen Malony

Author:
Xuehai Zhou

Report

Introduction

The pseudo-shell is a command-line interpreter based on the functionality of GNU Bash. All commands it will respond include ls, pwd, mkdir, cd, cp, mv, rm and cat, followed by 0 to 2 arguments. Incorrect usage like missing or extra arguments will cause error report. In addition, this pseudo-shell can identify several commands in one command line separate by ";". The program has an interactive mode and a file mode, which read commands from stdin or a file respectively. In interactive mode, the output will go to the stdout, while in the file mode the output goes to an output file. Files include a main.c, a command.c, a command.h and a Makefile, where command.h is provided. Library calls are allowed in main.c, but all commands are only built on system calls in command.c.

Background

I'll list built-in functions we need to know below before we start the project. (I'll skip simple functions like fprint())

1. *char *strtok_r(char *str, const char *delim, char **saveptr);*
2. *ssize_t getline(char **lineptr, size_t *n, FILE *stream);*
3. *DIR *opendir(const char *name);*
4. *struct dirent *readdir(DIR *dirp);*
5. *int mkdir(const char *pathname, mode_t mode);*
6. *int remove(const char *pathname);*
7. *int chdir(const char *path);*
8. *char *strrchr(const char *s, int c);*
9. *int closedir(DIR *dirp);*
10. *ssize_t read(int fd, void *buf, size_t count);*
11. *ssize_t write(int fd, const void *buf, size_t count);*
12. *int open(const char *pathname, int flags, mode_t mode);*
13. *int close(int fd);*

Implementation

Since main.c reads commands from the stdin or a file, we determine which mode is the program working on. If argc is equal to 3 and we see the f flag specified, we want to redirect the stdout to an output file using freopen(). If argc is equal to 1, we will go to the interactive mode. Else it's neither interactive mode nor file mode, then an error report. While loop uses getline() to get each command line and ends when encountered a "exit" or reached the end of the file in the file mode. Then, tokenize each line by ";" using strtok_r(). Since strtok_r tokenize 'string' from front to tail, we can do a "first token first serve". We pass each token to another strtok_r() and split by empty space. Words (here the 'word' I mean command and arguments) we got in second token we store them in a word_array. Then, word_array[0] will be the command and word_array[1], word_array[2] will be arguments. Thus, we can pass in each command to check if it's one of the valid commands. If so, we pass its argument to its command function which is implemented in command.c. If not, we throw an error and break out the loop.

There are many ways we can implement the main.c. For example, we can also implement it without a word_array or using additional array to store each token separate by ";". For implementation without an array, we can check each first second token to see whether it's a command or not. If so, we can pass its second *and third words as arguments to the command functions straight away. Then looping to the next token so far and so forth.*

The implementation of the command.c we should be aware of the consistency of the system calls. I'll demonstrate each function as follow. (for all following prints I mean calling write())

1. *ls: we go to the current directory and print out all the file name or directory name.*
2. *pwd: we call getcwd() to get the current directory and print out the current directory.*
3. *mkdir: we open the dirname first to check the directory's already existence. If the directory already exists, then report an error. Else call mkdir() to create the directory.*
4. *cd: I customized if we don't have an argument specified, then we go to the home directory by calling getenv(). Else we call chdir() to go to that directory, if chdir() == -1, then we report an error.*
5. *cp: we have two parameters passed in. The source path is a file, while the destination path could be a directory or a file. First of all, we determine if destination path is a directory by calling opendir() and check the existence of the directory. If exists, check if the source file is in current directory. If not, we tokenize the file name by calling strrchr() separate by "/". And then we add the file name to the end of the destination path. Then we are able to open the file in destination path with create and write flag, so it will create the file name if the file doesn't exist in the destination path or it's already to be overwritten if it has been existed. Then we just read the source file and write the file character by character to the destination path by calling a while loop and read(), write() system calls.*
   To be aware of: when we do strrchr(), it will change the destination pointer permanently, so we want to allocate space for a copy of the destination path and manipulate on it. Otherwise, it will cause massive bugs.
6. *mv: since the mv functionality is similar to cp from cp two parameters and delete the source file, I can simply call the cp function by passing same two parameters then call the rm function to remove the source file.*
7. *rm: check the existence of the file then call the remove() to delete the file.*
8. *cat: open the file and read it character by character then write to the stdout by calling read() and write().*

Performance Results and Discussion

In both modes, all functions work properly, and all memory leaks are fixed. Type in ./pseudo-shell with or without -f flag and an input file after make file. In interactive mode, it asks user to type in commands in the correct manner. Otherwise, it reports error and prompt user the right usage. In file mode, it will redirect the stdout to an output file named "output.txt".

Conclusion

This program is only a simple shell simulator. All commands cannot play any role with flags, so there are a lot of usage we can add on in the future.

Some videos I found would be helpful: (No copyright infringements intended. All credits go to the owner of these videos.)

1. *strtok():* https://www.youtube.com/watch?v=HCqqSJYE00M
2. *open(), read(), write(), close():* https://www.youtube.com/watch?v=dP3N8g7h8gY&t=533s

credits: cp function inspiration from Missy Shi.