EC3215: Coding Assignment

Your own ls command and simple shell

Due 2023. 12. 16. Midnight

1 Write your own `ls` program [100 pts]

Implement your own `ls` program. The goal of this assignment is to create a simplified version of the `ls` command using C programming. This command lists directory contents and displays various details about files and directories.

- Objective:
  - File Listing: Learn how to use system calls like opendir(), readdir(), closedir() to list directory contents.
  - Formatting Output: Implement options to format the output (e.g., -l for long listing, -a to show hidden files) utilizing system calls like stat().
- Requirements:
  - (10 pts) Your program should list the contents of a directory specified by the user. If no directory is specified, it should list the contents of the current directory.
  - (20 pts) Implement the -a or –all option to include hidden files (those starting with .) in the listing.
  - (50 pts) Implement the -l option to display detailed information about each file, including:File type and permissions. Number of hard links, File owner,File group,File size, Timestamp (last modification time), Linked file for a soft link and etc. File name.
  - (5 pts) When not using the -l option, display the file names in a columnar format, similar to the default ls output.
  - (10 pts) Your program should gracefully handle errors, such as an invalid directory, and display appropriate error messages.
  - Submit one source code file (`yourID_ls.c`) on the LMS site. You can include a README.md file that explaines how to compile the program if there is dependencies. Also you can use Makefile for compilation.
  - (5 pts) *** Your code should contain comments explaining what the code blocks do.**

Test case

```
$ ls
$ ls .
$ ls nodir     # (There is no 'nodir' directory in the path)
$ ls -l dir
$ ls -a
```

2 Write Your Own Shell [100pts]

For this assignment, you will implement your own simple shell program that supports I/O redirection and pipelines. The goal of this homework assignment is to allow you to practice writing basic system call functions, such as `fork, exec, wait`, as well as I/O rediction and pipeline. In this assignment, you will need to utilize `fork, exec, wait, exit, open, pipe, dup2, close`, etc. system call functions. You can start with the example codes such as "`simple-shell.c`" from Lecture 2 and "`shellex.c`" from Lecture 15.

- Objective:
  - Better understanding of shell in the Unix operating systems.
  - Learn about how to implement shell in your Linux system using system call functions and standard C library functions.

Submission guideline

You should submit your homework on the LMS site. Submit only one code named `yourID_shell.c` .

(5 pts) * Your code should contain comments explaining what the code blocks do.**

2.1 Handling Signals [5 pts]

Implement your own signal handler that traps SIGINT signal. When you press ctrl+c upon your terminal, print "You are in my custom shell. See you again. Bye!" and then terminate the shell program.

- Requirements:
  - (5 pts) When you press ctrl+c, rather than terminating the program instantly, intercept the signal and execute a user-defined signal handler.

Test case

```
$ ^c    #(ctrl+c)
```

2.2 I/O Redirection [50 pts]

Implement your own command line program that handles input/output redirection .

- Objective:
  - Better understanding of the `dub2` system call function.
  - Learn about how to implement I/O redirection in your Linux system.
- Requirements:
  - (10 pts) Redirect standard output to a file (`ls > file.txt`)
  - (10 pts) Redirect standard output to a file, but the results are appended (`ls » file.txt`)
  - (10 pts) Redirect standard input from a file (`sort < file.txt`)
  - (20 pts) Redirect standard input from a file and then output to a file (`sort < file.txt > sorted.txt`)

Test case

```
$ wc -l < test.c
$ ls > file_list.txt
$ ls >> file_list.txt
$ sort < file_list.txt > sorted_file.txt
```

2.3 Pipeline [40 pts]

Write your simple shell program that handles pipelines.

- Objective:
  - Better understanding of the `pipe` system call function, and uni-directional interprocess communications.
  - Learn about how to implement pipilines (|) in your Linux system.
- Requirements:
  - (10 pts) Handle two commands using single pipe.
  - (20 pts) Handle mutiple commands chained using multiple pipes.
  - (10 pts) Handle both I/O redirection and pipe in one program.
- Sample Execution:

Test case

```
$ ls -l | head
$ cat test.c | sort | uniq
$ grep int test.c | wc
$ ls | wc -l > out.txt
```