Documentation Assignment 1

Basic Linux/Unix Shell

Project Created By: Deepanshu Dabas

Semester 3, IIIT Delhi

***Internal Commands*:** 'cd', 'echo' and 'pwd'

Additional Command (for exiting shell): exit

External Commands: 'ls', 'cat', 'date', 'rm' and 'mkdir'

Cd:

Options:

1) cd .. (To return back to previous directory)
2) cd (To return to first directory of pc)
3) cd – (To go to /home/user folder)
4) cd ~ (To go to /home/user folder)

Functionality:

chdir() function for changing directory. If no arguments are given with cd, then our directory would be changed to default directory i.e "/".

If user enter cd .. then we will return to parent directory of current directory.

If correct path is specified it will change directory to specified directory. Also, our current path is also changed in shell output by updating pwd variable (an array to store current path).

Error Handling:

If incorrect path of directory is specified then error is produced (Error: No such file or directory)

Echo:

Options:

1) -n (To print without newline character)
2) --help (To print help menu of echo)

Functionality:

If nothing is entered with echo command then newline character is printed. Else all other thing specified with echo is printed to console.

If -n is specified then no newline character is printed

Pwd:

Options:

1) -L (print the value of $PWD if it names the current working directory)
2) -P (print the physical directory, without any symbolic links)

Functionality:

By default, `pwd' behaves as if `-L' were specified. getcwd() system call is used for getting current working directory and

realpath() for -P option. -L is implemented with getcwd() only.

Error Handling:

If getcwd() fails to fetch current working

directory null is printed

Exit: For exiting shell.

Cat:

Options:

1) -E (display $ at end of each line)
2) -n (number all output lines)

Functionality:

If filename is given as input, corresponding file content is printed. C file i/o used for implementing cat functionality. Using fgetc() file content is printed character by character until we reach End of file.

If nothing is given as input with cat. Then cat starts input, output mode Whatever content is given to it as input, is printed directly as output.

If -E flag is specified then at the end of each line '$' is printed. Implemented by just checking if char c== '\n' and printing $ in that case.

If -n flag is specified then line number is printed in each new line. Implemented by maintaining a variable and incrementing it after every "\n".

Error Handling:

If incorrect file is given as input, then error is printed (cat: Error, file does not exist). Even if flags are specified this error handling works.

If incorrect flag is given, error is printed (cat: Error, file does not exist)

Date:

Options:

1) -u (print Coordinated Universal Time (UTC))
2) -r (display the last modification time of FILE)

Functionality:

time.h library is used for getting current date (in case only date is entered as input) using time() function. For -u option gmtime() is used instead of ctime (which we used in case of user input== 'date' to fetch UTC time and by using asctime() function UTC time is printed as a string.

For -r flag, sys/stat.h is used (specifically struct stat) and by using ctime() again we got last modified time of specified file.

Error Handling:

In case of -r, if incorrect file is given as input, then error is printed (date: "filename": No such file or directory). If no filename is specified, error is printed(date: option requires an argument – "flag name")

If incorrect flag is given, error is printed (date: error invalid option)

If we are not able to fetch time in UTC, error is printed (date:error unable to handle process right now)

Rm:

Options:

1) -i (prompt before every removal)
2) -f (ignore non-existent files and arguments, never prompt)

Functionality:

Using unlink() function files are removed. Using stat function, we check if directory/file exist or not.

If -i flag is specified then we ask user for permission before removing file.

If -f flag is specified all warnings are suppressed and only error printed will be if no arguments are specified with -f flag.

Error Handling:

In caser user doesn't enter any command with flags, error is produced (rm: At least one argument required)

In case user doesn't enter anything except rm error is produced (rm: Error.Atleast one option required)

If user enter incorrect file/directory name, error is produced (only if -f flag isn't used).

If we are not able to remove file for any reason we print "rm: error can't remove file". (Also, in case of -i flag)

Mkdir:

Options:

1) -v (print a message for each created directory)
2) -m (set file mode)

Functionality:

Using mkdir() system call, directory is created and default mode of file is set.

In case of -m specified, we set specified mode instead of default (ex: 0755) with user input and directory is created in corresponding mode.

If -v flag is set then we will print a message on successful creation of directory (mkdir: created directory)

Error Handling:

If we are unable to create directory an error is printed (mkdir: Can't create directory)

If user select -m mode and doesn't specify any other argument then error is printed (mkdir: missing operand).

Also, if no arguments are given error is shown (mkdir: missing operand).

ls:

Options:

1) -a (do not ignore entries starting with .)
2) -i (print the index number of each file)

Functionality:

Using dirent.h functions such as readdir(), opendir(),etc, implementation of above functionalities is done. We first check which flags are specified and then set corresponding loop. If -a flag is specified all files are printed including hidden files. In case of -i all files are printed with their index using d_ino attribute of structure. Using d_name attribute, names of all the directories are printed. If no option is specified, then we need to list directories present in current directory. This is implemented by using opendir(“.”).As “.” Will point to current directory. And then by using stat and dirent.h structures, directories are printed.

Error Handling:

We first check the specified path with ls. if path is correct then only, we run other functionalities such -a ,handling incorrect path/folder specified by user.("ls: Error, 'file name' - No such file or directory)

If we are not able to open files in directory due to restricted file mode or any other error in opening, error is printed on screen ("ls: error process wasn't successful. Please try again").

MainShell:

Functionality:

Internal functions are implemented inside this only. Character by character input is taken from user and after user press enter, the command is split into array of string (or char**). Using strcmp(),command is checked whether its internal, if it is internal then corresponding function in called and this array is passed into it. If it's not internal and &t is not specified at the end of command then new process with fork is created.

Forking:

We then check which process it is using pid_t if its parent process then it needs to wait() till child process is completed, if it is child process then using execv() we called corresponding file (files created after compiling using our makefile). For calling file we need to specify path of the file, which we implemented by copying path into array and then strcat() it with filename. If execv() return -1 then it's an error, and then we print Error: Command Not found. If we are not able to do create process by fork() then an error is printed.

Threading:

If &t is specified at the end of command, then we need to call external file using system and by creating threading. Threading is implemented by creating a function for thread which contains system() in it. Since system requires a single string of command, we copy path

of directory and then copy command into a char *array and then send this array into system() function. We join this created thread with our main function by specifying NULL in join. Thread is created using POSIX pthread_create() function. In case of incorrect/not supported command system itself print an error.

Shell:

Loop is maintained by Boolean Flag=true. Some colour ascii are used for better display. In loop we first take input from user using our helper function. Then pass this function into our process handling function which compares commands with supported commands. We store return value for exiting from loop in case of exit, we can compare return value and exit.

Test Cases

pwd

cd

cd /home/Deepanshu/Desktop

cd ..

cd Desktop

pwd -P

pwd -L

cd --

pwd

echo -n Hello World

echo Hello System

echo –help

ls

ls -a

ls -i

ls -r

ls /home

ls -a /home

ls -i /home

ls incorrect_directory

ls -a incnc

ls -i oror

cat

cat Makefile

cat /home/deepanshu/Desktop/Makefile

```
cat -E Makefile
cat -n Makefile
cat -E
cat -n
date
date -u
date -r date.c
date -r mkdir.c
date -r mdjdjjd
date -r
rm test1
rm -i test2
rm -i test2
rm -f jejejej
rm -f test3
rm
rm -f
rm -i dnnnfd
mkdir abcd
mkdir -m 0755 b
mkdir -m 0777 c
mkdir -v newdire
ls
mkdir -m
mkdir -m 0777
mkdir -v
exit
```