Operating System Project Report

Full Unit – Final Report

Shello

Maria Afara & Mohammad Abboud

Supervisor: **Dr Hussein Wehbi**

Department of Computer Science

Section 5, Lebanese University

January 19, 2018

Getting Started

Objectives:

1. Understand the working of command line interface in Unix-like environment.
2. Understand the process forking mechanism.

You need to take the input from the command line in an infinite loop till an "exit" is entered and the corresponding output should be printed to stdout.

The working of a basic shell can be explained as follows

1. Wait for user input (command)
2. Parse the input to get the command and the arguments
3. Create a new process to execute the command (Use fork () system call)
4. In the new process, execute the command (Use execvp () system call) and exit with the status of the command executed
5. In the parent process, wait for the new process to terminate (Use wait () system call). Get the exit status of the child process
6. Go to step 1 to continue waiting for the user input.

Implemented Functionalities are as under:

1. Execute all the External commands (ls, clear, etc.)
2. Implement Internal commands: cd
3. Redirection operators: STDIN, STDOUT, STDERR (>>,>,<)
4. Support for some command-chaining operators
5. Pipes "|" (multiple)

Used command-chaining operators:

1. Ampersand Operator (&)

The function of '&' is to make the command run in background. Just type the command followed with a white space and '&'. You can execute more than one command in the background, in a single go.

2. AND Operator (&&)

The AND Operator (&&) would execute the second command only, if the execution of first command SUCCEEDS, i.e., the exit status of the first command is 0. This command is very useful in checking the execution status of last command.

3. OR Operator (||)

The OR Operator (||) is much like an 'else' statement in programming. The above operator allow you to execute second command only if the execution of first command fails, i.e., the exit status of first command is '1'.

4. PIPE Operator (|)

This PIPE operator is very useful where the output of first command acts as an input to the second command. For example, pipeline the output of 'ls -l' to 'less' and see the output of the command.

Commonly Used functions

Waitpid is used when you have more than one child for the process and you want to wait for particular child to get its execution done before parent resumes.

The **strtok** function is used to tokenize a string and thus separates it into multiple strings divided by a delimiter.

Commonly used commands

Table 1. Some commands and functionality

commands	functions
ls	List directory contents
wc	Print newline , word , and byte count for each file
who	Shows who is logged on
cd	Change the working directory
clear	Clear the terminal screen
cat	Concatenate the files and print on the standard output
grep	It searches files for lines that match a pattern and returns the results
Ls -a	lists all contents in the working directory, including hidden files and directories
Ls -l	lists all contents of a directory in long format
Ls -t	Orders files and directories by the time they were last modified.
Pwd	Prints the name of the working directory
Sort	Takes a filename or standard input and orders each line alphabetically, printing it to standard output.

Ourshell2

Basic Function used:

Handler ()

Function that handles input lines that include pipe families that we included in our project
Such as redirections of files…

PrintDir ()

That shows us in which directory we are.

SignalInt ()

Function for **Control** + **C** which is used to kill a process with the signal SIGINT.

Redirection () /fpipe1

Function that handles redirections of files.

FExec ()

Function that executes single commands.

Some Functions description

Shell runs in an infinite loop
In this loop
Read input line
Check if the user enters enter key directly
Then the loop continue
Else each word of this input is saved in an array
Then sent to handler function
The array is handled now
First checks if the is special character where we used in our project the redirections character's (<, >, < >) and pipes (|) and &.
And also checks if the user enters the words clear, cd, or exit
Where clear is to clear the shell screen, cd to change current directory, and exit to exit from the system created.
If one of the redirection characters is found
According to the character it is sent to the function that execute it with its option.
For example if (>)
I am writing from a file fPipe1 (args1, NULL, args[i+1],0); 0 for writing
This is done in the following steps
Open file
Close the screen
Dup for the file in order to write in the file and not on the screen
Then close file Descriptor

Ourshell1

Basic Function used:

Char *inputReading()

Reading input from the shell we allocate a buffer to store input from stdin if we hit end of file or \n we replace by \0 to terminate and return this input

First allocate the buffer then checks if its allocation was successful or not

If it's successful; we read character by character if we read EOF or enter add to the buffer '\0' and return it else we add to the buffer by each index its corresponding character read. And jump to next position, Then if needed we reallocate it.

Int launch(char ** args)

it used if there are no pipes we should give the work for a child in order to keep the program running otherwise the program will execute the args and terminate

Char **splittinginput(char * line)

Splitting from delimiters the saved line from the stdin since the execvp executes the arguments that are split from delimiters.
First we allocate the buffer
If buffer allocated successfully strtok function is executed on the line
(Strtok generates pointers of the arg that are split from the given delimiters)
While we still have arguments we store each argument in the array of arguments
If needed we realloc the buffer. And after we finish splitting we put null at the last of args since when executing execvp we should give the last argument which is null or (char *)null. And returns the args .

char **splitingPipe(char *line)

this is used when we have a pipe in order to make a first split on '|' and then we will use the above method to split delimiters of each argument
//for splitting the pipe same procedure is followed as in function splittingInput but here we replace the delimiters with the pipe character '|'

int executing(char **args)

Function that executes building functions .

void execArgsPiped1(char** parsed, char** parsedpipe)

Function for one pipe.

int launchMany(char *args)

its used if their are no pipes we should give the work for a child in order to keep the program running otherwise the program will execute the args and terminate

problems

While redirection inputs to a file if the file is already created manually we faced no problem ,whereas it's not created , the file is created but could not access it or retrieve data from it .same problem for all redirections implemented .

Problem was forgot to close

Modes of files explanation:

fopen() mode	open() flags
r	O_RDONLY
w	O_WRONLY | O_CREAT | O_TRUNC
a	O_WRONLY | O_CREAT | O_APPEND
r+	O_RDWR
w+	O_RDWR | O_CREAT | O_TRUNC
a+	O_RDWR | O_CREAT | O_APPEND