Assignment 4 – C Shell

Operating Systems (CS416)

Rutgers University

Name: **Jishan Desai** (jpd222), **Malav Doshi** (md1378)

The project implements the working of a shell in Linux.

Knowing the following variables may help to understand the documentation faster:

- **char cwd[100]:** stores the current working directory.
- **int pid:** stores id of child process.
- **int killp:** 1 if other commands need to be killed. 0 otherwise
- **int mult:** 1 if the input command contains multiple commands
- **int last_read:** read end of the pipe where a command wrote last. (to be used by next command)
- **jump_buf env:** Used to store the environment before going to signal handler.
- **unsigned char isChild:** 1 if the shell has created a child. 0 otherwise.

The detail of each function and how it works is given below:

----------Functions ----------

1**. void run_command(char* command):** This function is called to run a particular command. ***command*** stores the command that would be passed to this function. It then parses ***command*** using *strtok()* and runs the command.

2. **check_space(char* string):** This is a helper function which truncates all the leading and trailing spaces from the given string. This is helpful while extracting the command to get rid of unwanted spaces or new lines.

3. **void handle_sigint(int signum):** This is a signal handler which is invoked when an interrupt occurs while running the shell. This does not exit from shell but from the child process which was running the given command. If the input command has multiple commands and SIGINT occurs then the signal handler wont allow the commands after the running command to execute.

4. **void run_redir(char* redirCommand, unsigned char isPiped):**

- **redirCommand** : command with redirection (ls > ls.txt)
- **IsPiped:** 1 if the function is called by run_pipe(). Otherwise 0.

This function is used to run redirection commands. The function first gets the string till ">". So, In case of ls > ls.txt. The function first stores "ls" and then checks if the next token is ">" if so, the command is asking for ">>" (an indicator is set accordingly). Then the using dup() system calls the stdout is manipulated to redirect the output to a file. However, if isPiped is on, even stdin is changed to file descriptor pointed by last_read. So, run command reads from that file descriptor rather than stdin. Once the changes are made, stdout and stdin are restored.

5. **int countPipes(char* pipeCmd):**

- **char* pipeCmd:** command from which to extract the number of pipes

This is used to count the number of pipes in the given command.

Returns the number of pipes in the command *pipeCmd*.

6. **void run_pipe(char* cmd):**

- **char* cmd:** command with pipes to run.

The function first tokenizes the cmd to get the first command till the " | ". So, if the input provided is "ls | grep <something>". First command will be ls and the last command will be grep <something>. When the function is invoked, it will grab the first command and pass it to temp_run_command function with updated read write values. **Read** and **Write** can either be 1 or –1. 1 meaning the current command will need that end of the pipe. So, following values can be passed:

Read	Write	Interpretation
-1	1	First Command
1	1	Some middle Command
1	-1	Last Command

Table 1

When temp_run_command is done, run_pipe will grab another command grep <something> (from the example provided) and updated read and write values and call temp_run_command.

If the command grabbed is the last command, the function will check if it has any sort of redirection, if it has redirection then instead of temp_run_command, run_redir called with an indication variable saying that it is called from pipe.

Note: We are assuming that redirection command will always be at the end of pipes. As it was stated on piazza.

7. **void temp_run_command(char* command, int read, int write, unsigned char isPiped):**

- **char* command =** command to run
- **int read** = tells if the command will read from pipe or not
- **int write** = tells if the command will write to pope or not
- **isPiped =** will be 1 if it is called from run_pipe, otherwise 0.

This function is used to run one command from the set of commands with piping. For example, if command given is "ls | grep fil.txt "this function is called first from *run_pipe()* to execute "ls". Before executing, based on **Table 1,** the function will change the writing and reading descriptors using dup2 system call. It will follow the same steps for other set of commands too. At the end the the read end of the currentThis way even multiple pipes are supported.

8. **void run_cd(char* cmd):**

- **char* cmd:**

It runs the "cd" command with the help of *chdir()* system call.

9. **int check_semicolon(char* string):**

- **string:** string in which to check for semicolon

This function is used to check for semicolons in the given command. This is to check if the command has other sub commands or not.

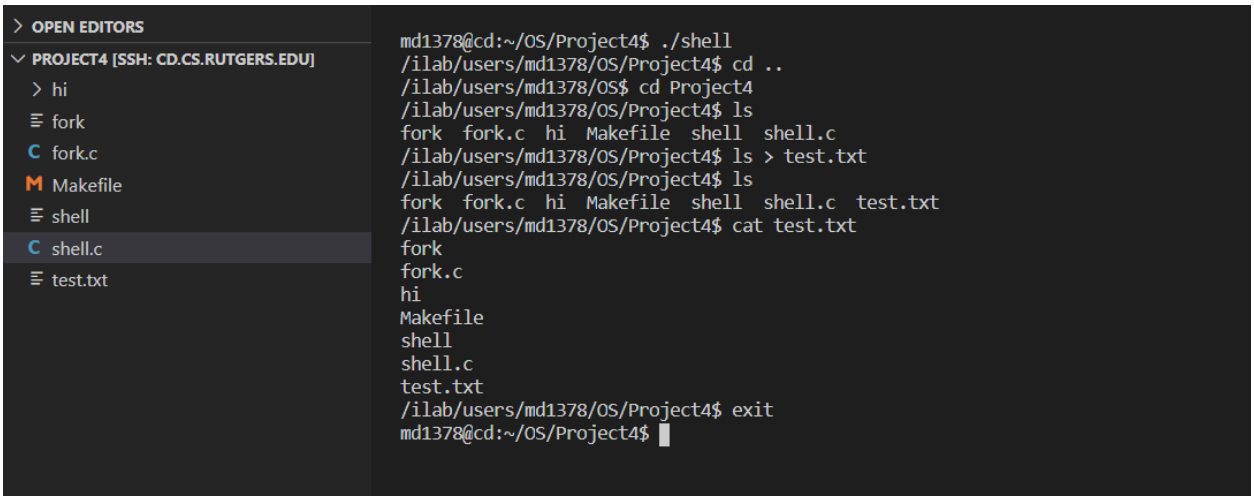
10. **char** string_tokenize(char* com, char* args[10]):**

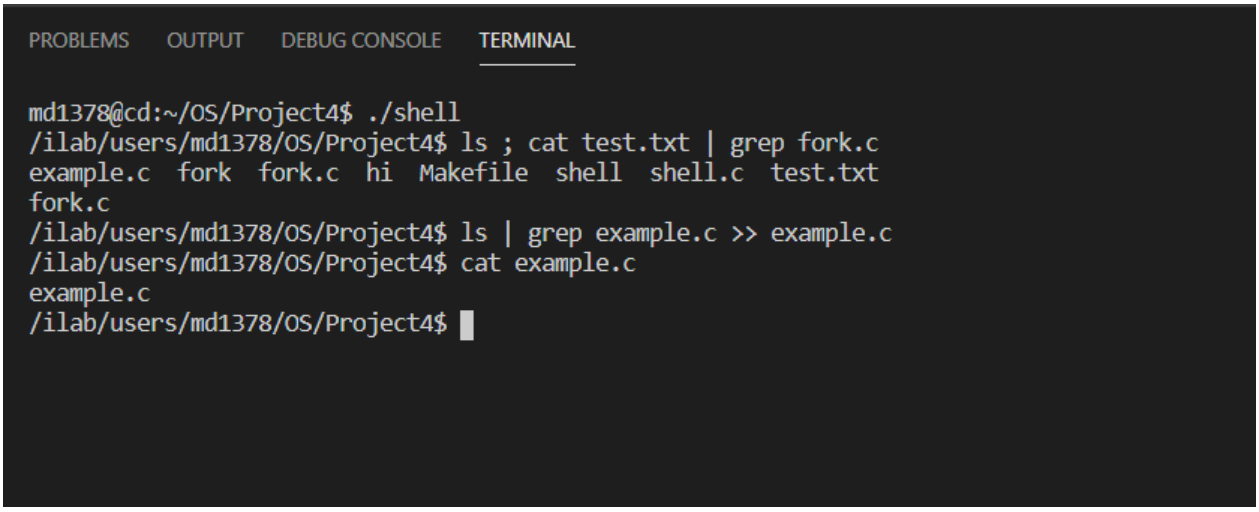
- **com:** command to tokenize
- **args:** array of strings to store the tokens

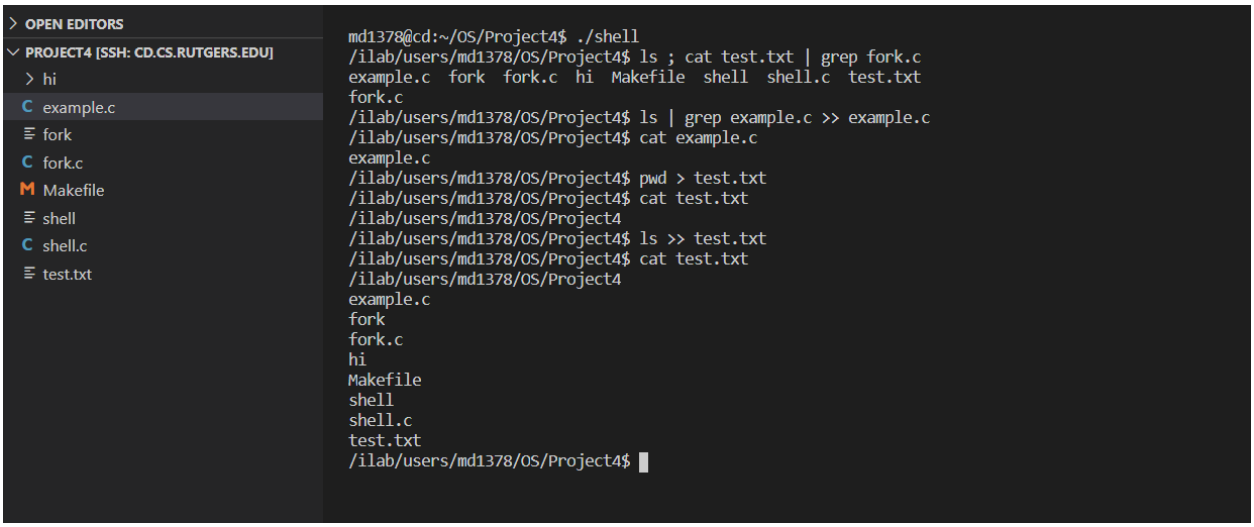
This function is used to tokenize commands which have other sub commands separated by ';'.

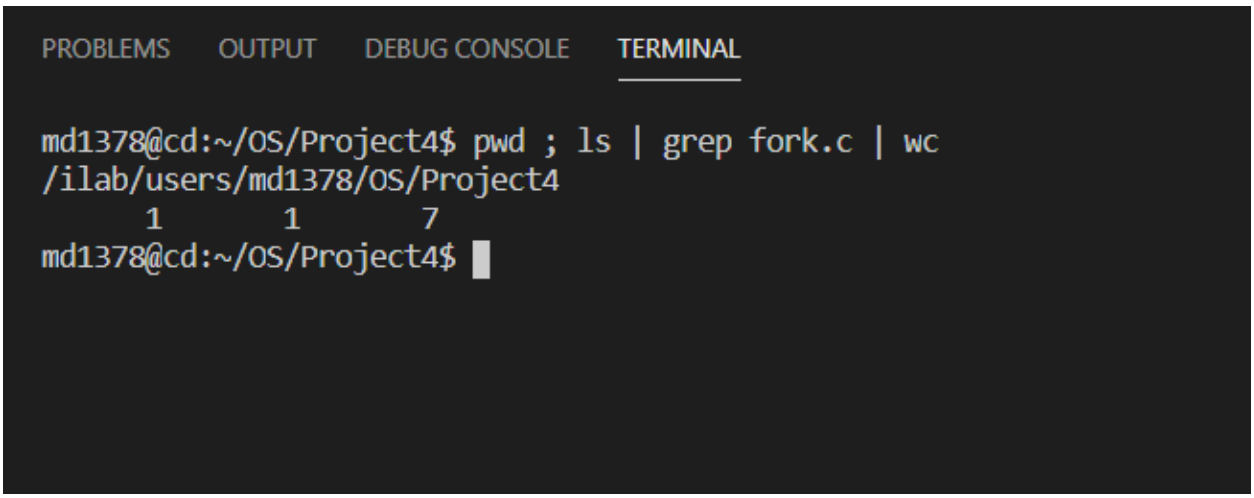
Each command is then stored in args which is then returned.

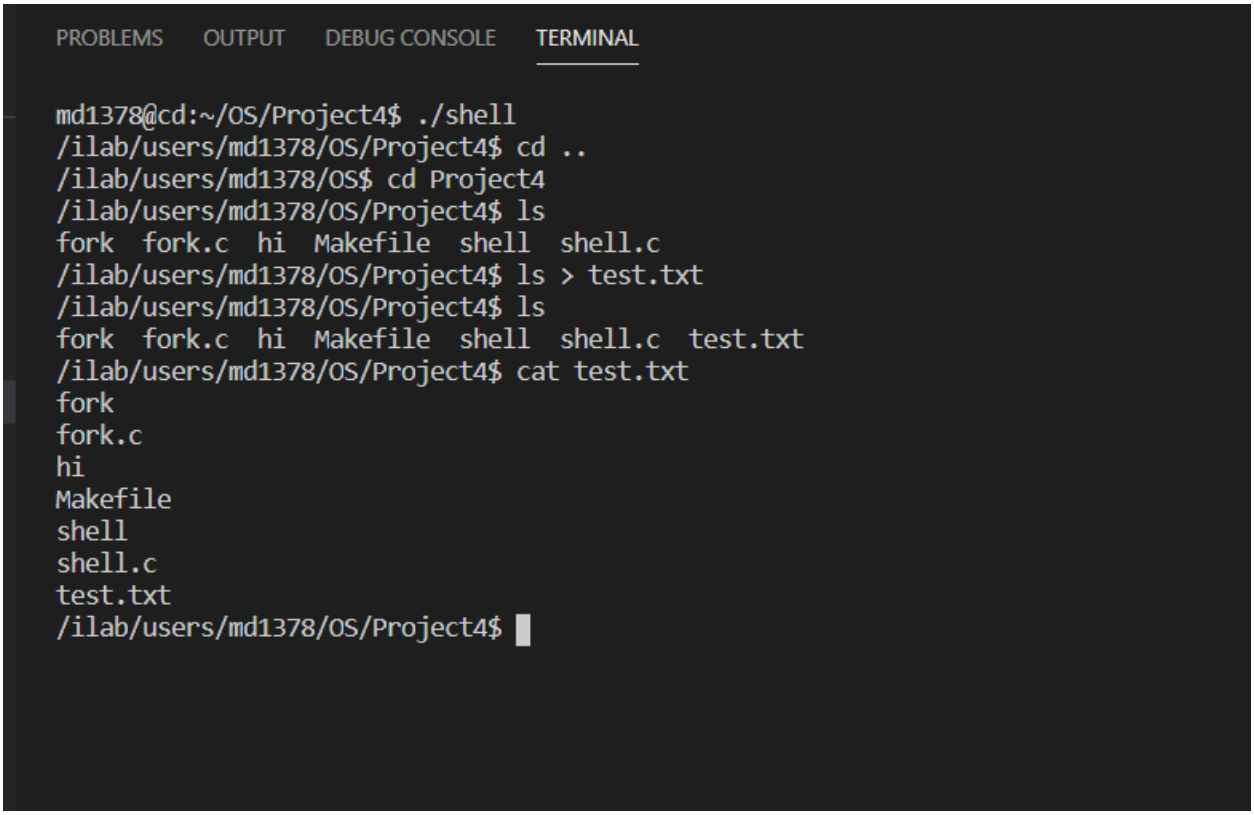
----------RESULTS----------


OPEN EDITORS
PROJECT4 [SSH: CD.CS.RUTGERS.EDU]
hi
fork
fork.c
Makefile
shell
shell.c
test.txt
md1378@cd:~/OS/Project4$ ./shell
/ilab/users/md1378/OS/Project4$ cd ..
/ilab/users/md1378/OS$ cd Project4
/ilab/users/md1378/OS/Project4$ ls
fork fork.c hi Makefile shell shell.c
/ilab/users/md1378/OS/Project4$ ls > test.txt
/ilab/users/md1378/OS/Project4$ ls
fork fork.c hi Makefile shell shell.c test.txt
/ilab/users/md1378/OS/Project4$ cat test.txt
fork
fork.c
hi
Makefile
shell
shell.c
test.txt
/ilab/users/md1378/OS/Project4$ exit
md1378@cd:~/OS/Project4$


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
md1378@cd:~/OS/Project4$ ./shell
/ilab/users/md1378/OS/Project4$ ls ; cat test.txt | grep fork.c
example.c fork fork.c hi Makefile shell shell.c test.txt
fork.c
/ilab/users/md1378/OS/Project4$ ls | grep example.c >> example.c
/ilab/users/md1378/OS/Project4$ cat example.c
example.c
/ilab/users/md1378/OS/Project4$


OPEN EDITORS
PROJECT4 [SSH: CD.CS.RUTGERS.EDU]
hi
example.c
fork
fork.c
Makefile
shell
shell.c
test.txt
md1378@cd:~/OS/Project4$ ./shell
/ilab/users/md1378/OS/Project4$ ls ; cat test.txt | grep fork.c
example.c fork fork.c hi Makefile shell shell.c test.txt
fork.c
/ilab/users/md1378/OS/Project4$ ls | grep example.c >> example.c
/ilab/users/md1378/OS/Project4$ cat example.c
example.c
/ilab/users/md1378/OS/Project4$ pwd > test.txt
/ilab/users/md1378/OS/Project4$ cat test.txt
/ilab/users/md1378/OS/Project4
/ilab/users/md1378/OS/Project4$ ls >> test.txt
/ilab/users/md1378/OS/Project4$ cat test.txt
/ilab/users/md1378/OS/Project4
example.c
fork
fork.c
hi
Makefile
shell
shell.c
test.txt
/ilab/users/md1378/OS/Project4$


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
md1378@cd:~/OS/Project4$ pwd ; ls | grep fork.c | wc
/ilab/users/md1378/OS/Project4
1 1 7
md1378@cd:~/OS/Project4$


PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
md1378@cd:~/OS/Project4$ ./shell
/ilab/users/md1378/OS/Project4$ cd ..
/ilab/users/md1378/OS$ cd Project4
/ilab/users/md1378/OS/Project4$ ls
fork fork.c hi Makefile shell shell.c
/ilab/users/md1378/OS/Project4$ ls > test.txt
/ilab/users/md1378/OS/Project4$ ls
fork fork.c hi Makefile shell shell.c test.txt
/ilab/users/md1378/OS/Project4$ cat test.txt
fork
fork.c
hi
Makefile
shell
shell.c
test.txt
/ilab/users/md1378/OS/Project4$