Lab: MyShell - Implement Your own shell in C

Shell commands are realized by C programs. In this lab, you will program as if you are the authors of the shell library, and write code to do file IO to implement the commands.

In this lab, you are asked to implement your own library to support the shell commands. We did an exercise in the class where you implemented basic `touch` **and** `cat` commands using File I/O functions in standard C library. In this lab, it is an extension to that exercise and you will implement shell commands for `printf`, `cat`, `redirection`, and `touch`. We call our shell commands with prefix `my_` (for example my_cat, my_echo) and the following is what the outcome of this lab will look like. Note that we use `@` instead of `>` to represent redirection in `myshell`.

./my_cat `file1` #cat file1

./my_echo `Alice` #echo Alice

./my_echo `Bob @ file1` #echo Bob > file1

./my_echo `Alice @ @ file1` #echo Bob >> file1

./my_touch `file1` #touch file1

This lab consists of the following tasks:

1. Setup the project framework using Makefile
2. Implement **my_touch** using file I/O
3. Implement `my_printf`
4. Implement `my_echo`
5. Implement `my_echo` with redirection
6. Implement `my_cat`
7. Implement `my_touch`

Programming references

- FileI/O functions

#include<fcntl.h>

open(char *pathname, int oflag)//open("file1",O_WRONLY|O_CREAT|O_APPEND);

#include<unistd.h>

ssize_t write(int fd, const void *buf, size_t nbytes);

ssize_t read(int fd, void *buf, size_t nbytes);

off_t lseek(int fd, off_t offset, int whence);

Lab_Tutorial

You need to have 8 files in this project:

1. Makefile
2. Header.h
3. myshell.c
4. main_printf.c (writes an arbitrary string on the screen)
5. main_touch.c (It creates a file)
6. main_cat.c (It shows the content of file)
7. main_echo.c (It writes the input string into the file)
8. main_echo0.c (writes the input string on the screen)

There are eleven steps/tasks in this lab and follow them below to finish the lab.

Step1: Create lab directory

Create a folder named "lab" and create 8 empty files with the following names (the names mentioned above):

Makefile, header.h, myshell.c, main_touch.c, main_printf.c, main_cat.c, main_echo.c, main_echo0.c

Step2: Makefile

Write the following content to file "Makefile" that you just created it:

```
printf: main_printf.o myshell.o

	gcc main_printf.o myshell.o

	./a.out


echo0: main_echo0.o myshell.o

	gcc main_echo0.o myshell.o -o my_echo0

	./my_echo0 Alice


echo: main_echo.o myshell.o

	gcc main_echo.o myshell.o -o my_echo

	-rm file1 file2

	./my_echo Alice @ file1
	./my_echo Bob @@ file1

	cat file1

	./my_echo Charlie @ file1

	cat file1

	./my_echo David @@ file2

	cat file2


cat: main_cat.o myshell.o

	gcc main_cat.o myshell.o -o my_cat

	./my_cat file1


touch: main_touch.o myshell.o

	gcc main_touch.o myshell.o -o my_touch

	./my_touch file1


compile:

	gcc -c main_touch.c main_printf.c main_cat.c main_echo.c main_echo0.c
myshell.c


clean:

	rm *.o *.out
```

You may need to understand this “Makefile” which is explained below. This is required for finishing step 7. But you can skip to the step 3 now.

Each line of “Makefile” is described here.

```
printf: main_printf.o myshell.o

	gcc main_printf.o myshell.o
	./a.out
```

First line: It introduces the target “printf” and puts “:” after it. Pay attention that after “:”, we see the list of files which target “printf” is dependent on. So here “printf” is dependent on main_printf.o and myshell.o files.

Second line: This line is the linking step which creates an executable file from two object files (main_printf.o and myshell.o).

Third line: The executable file “./a.out” is executed in this line. (Note that there are tabs in the beginning of lines two and three)

```
echo0: main_echo0.o myshell.o
	gcc main_echo0.o myshell.o -o my_echo0
	./my_echo0 Alice
```

First line: It introduces the target “echo0” and puts “:” after it. Pay attention that after “:”, we see the list of files which command “echo0” is dependent on them. So here “echo0” is dependent on main_echo0.o and myshell.o object files.

Second line: It generates an executable file named by the parameter that comes after “-o”, so it creates the executable file “my_echo0” by linking main_echo0.o and myshell.o.

Third line: Since the executable file “my_echo0” is created in the second line, we can run this file by command “./my_echo Alice”. “Alice” is the string we expect terminal to show us.

```
echo: main_echo.o myshell.o
	gcc main_echo.o myshell.o -o my_echo
	-rm file1 file2
	./my_echo Alice @ file1
	./my_echo Bob @@ file1
	cat file1
	./my_echo Charlie @ file1
	cat file1
	./my_echo David @@ file2
	cat file2
```

The first and second lines are described above.
Third line: It removes file1 and file2. The '-' symbol before "rm" means to run the following commands in the "Makefile" even when "rm" fails (for instance removing a non-exist file).
Forth line: As we know, the second line creates the executable file "my_echo". This command should overwrite "Alice" to file1. "@" is the new redirection symbol defined in our "./my_echo " command.
5th line: This command appends "Bob" to file1.
6th line: it shows the content of file1.
7th line: it should overwrite "Charlie" to file1.
9th line: it should append "David" to file2.

```
cat: main_cat.o myshell.o

	gcc main_cat.o myshell.o -o my_cat

	./my_cat file1
```

Second line: the executable file "my_cat" is created.
Third line: This line should show the content of file1.

```
compile:

	gcc -c main_touch.c main_printf.c main_cat.c main_echo.c main_echo0.c
myshell.c
```

This command compiles all the c source files and creates 5 object files. This target should be run before every other targets in the Makefile.

Step3: Implement header.h

Copy and paste this content to the file "header.h":

```
#if !defined(HEADER_H)
#define HEADER_H
#include<stdio.h>
#include<unistd.h> //lseek, STDIN_FILENO
#include<stdlib.h>
#include <fcntl.h>

int my_strlen(char* format_string);
void my_echo(int fd, char* str);
void my_printf(char* format_string);
void my_cat(char* pathname);
void my_touch(char* pathname);
#endif
```

The last four lines are the functions that are called in the five C source files, so they are declared in "header.h".

Step4: Implement myshell.c

Write 4 functions in the file "myshell.c". Copy and paste this content to the file "myshell.c":

```
#include "header.h"
int my_strlen(char* str){
}
void my_cat(char* pathname){
}
Void my_touch(char* pathname){
}
void my_echo(int fd, char* str){
}
void my_printf(char* str){
}
```

You should implement the body of these functions with the use of File I/O functions.

- Function "my_strlen" should be able to find the length of an input array (char* str).
- Function "my_cat" should be able to show the content of the input pathname (char* pathname).
- Function "my_echo" should write the input string (char * str) in to the specified file descriptor (int fd).
- Function "my_printf" should be able to output the input string (char *str) on the screen.

Step5: Implement main_printf.c

The goal of writing this file is to show an arbitrary string on the screen. So, you should relate the implementation of this file and the implementation of function "my_printf" in file "myshell.c" to show an arbitrary string on the screen.

Hint: As it is described in step 2, by running "Make printf" (run it in step9), it compiles the files "main_printf.c" and "myshell.c", and the executes "./a.out". So you should write a simple main function ("int main") now which only passes an arbitrary string into the function "my_printf" in "myshell.c" (this function is described in step 4 and since you are calling it here, it is included in file "header.h" (step3)).

Step6: Implement main_echo0.c

The goal of writing this file is to show the input string on the screen. So, you should relate the implementation of this file and the implementation of function "my_echo" in file "myshell.c" to show the input string on the screen.

Hint: As it is described in step 2, by running "Make echo0" (run it in step9), it compiles the files "main_echo0.c" and "myshell.c", and the executes "./my_echo0 Alice". So you should write a simple main function (int main (int argc, char *argv[])) now which only passes the input string into the function "my_echo" in "myshell.c" (this function is described in step 4 and since you are calling it here, it is included in file "header.h" (step3)).

Hint: int main (int argc, char *argv[])

The input variable "argc" is the number of parameters in the command. For example by running command "./my_echo Alice", we have two parameters (argc=2).

The input variable "argv" is a two dimensional array of parameters in the command. For example by running command "./my_echo Alice":

Argv[0] = myecho

Argv[1] = Alice

Argv[1][0] = A, Argv[1][1] = l, Argv[1][2] = I, Argv[1][3] = c, Argv[1][4] = e

Step7: Implement main_echo.c

The goal of writing this file is to write the input string into the file. So, you should relate the implementation of this file and the implementation of function "my_echo" in file "myshell.c" to write the input string into the file.

Hint: As it is described in step 2, by running "Make echo" (run it in step9), it compiles the files "main_echo.c" and "myshell.c", and then executes "./my_echo Alice @ file1" or "./my_echo Alice @@ file1". So you should write a simple main function (int main (int argc, char *argv[])) which passes the input string and the input file into the function "my_echo" in "myshell.c" (this function is described in step 4 and since you are calling it here, it is included in file "header.h" (step3)).

Hint: You should pay attention that the difference between "main_echo.c" and "main_echo0.c" is the number of input parameters. Here you have 4 input parameters while in "main_echo0.c" you had 2 input parameters.

Hint: command "./my_echo Alice @ file1" overwrites "Alice" in file "file1", while command "./my_echo Alice @@ file1" appends "Alice" to the end of file "file1". You should consider it while implementing here.

Step8: Implement main_cat.c

The goal of writing this file is to show the content of file. So, you should relate the implementation in this file and the implementation of function "my_cat" in file "myshell.c" to show the contents on the screen.

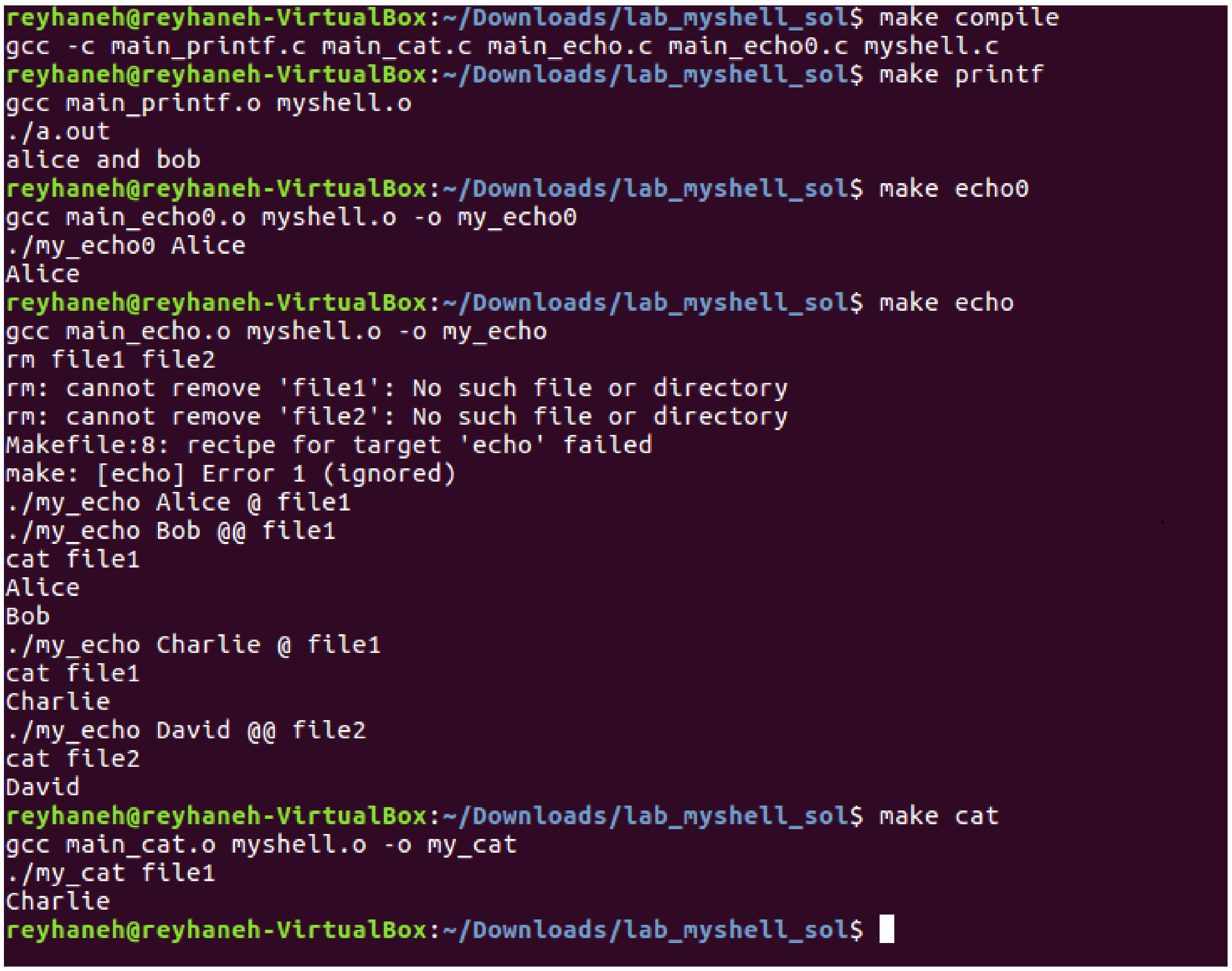
Hint: As it is described in step 2, by running "Make cat" (run it in step9), it compiles the files "main_cat.c" and "myshell.c", and then executes "./my_cat file1". So you should write a simple main function (int main (int argc, char *argv[])) which only passes the input filename into the function “my_cat” in “myshell.c” (this function is described in step 4 and since you are calling it here, it is included in file “header.h” (step3)).

Step9: Run Makefile

Enter the “lab” directory and run the following commands in this step:

- make compile
- make printf
- make echo
- make echo0
- make cat

If you run these commands and write the codes correctly, you should be able to see the output the same as the following image (Note that my arbitrary string in function “my_printf” was “alice and bob”):



```
reyhaneh@reyhaneh-VirtualBox:~/Downloads/lab_myshell_sol$ make compile
gcc -c main_printf.c main_cat.c main_echo.c main_echo0.c myshell.c
reyhaneh@reyhaneh-VirtualBox:~/Downloads/lab_myshell_sol$ make printf
gcc main_printf.o myshell.o
./a.out
alice and bob
reyhaneh@reyhaneh-VirtualBox:~/Downloads/lab_myshell_sol$ make echo0
gcc main_echo0.o myshell.o -o my_echo0
./my_echo0 Alice
Alice
reyhaneh@reyhaneh-VirtualBox:~/Downloads/lab_myshell_sol$ make echo
gcc main_echo.o myshell.o -o my_echo
rm file1 file2
rm: cannot remove 'file1': No such file or directory
rm: cannot remove 'file2': No such file or directory
Makefile:8: recipe for target 'echo' failed
make: [echo] Error 1 (ignored)
./my_echo Alice @ file1
./my_echo Bob @@ file1
cat file1
Alice
Bob
./my_echo Charlie @ file1
cat file1
Charlie
./my_echo David @@ file2
cat file2
David
reyhaneh@reyhaneh-VirtualBox:~/Downloads/lab_myshell_sol$ make cat
gcc main_cat.o myshell.o -o my_cat
./my_cat file1
Charlie
reyhaneh@reyhaneh-VirtualBox:~/Downloads/lab_myshell_sol$ 
```

As you see, the output of each command is based on the commands in "Makefile".

Step10: Execute your program and submit deliverables

Exercise: Submit to Blackboard all the files (including Makefile, header file and 2 c files main_printf.c and myshell.c) and screen shots of running the following commands.

Run these commands in terminal:

- make compile
- make printf

Homework 4: Submit to Blackboard all the files (including Makefile, header file and 4 c files main_cat.c, main_echo0.c, main_echo.c and myshell.c) and screen shots of running the following commands.

Run these commands in terminal:

- make compile
- make echo0
- make echo
- make cat

Or

- make compile
- make echo0; ./my_echo0 Alice
- make echo; ./my_echo Bob @ file1; cat file1
- ./my_echo Alice @@ file1; cat file1
- make cat; ./my_cat file1