

ECE 40800 / CSCI 40300 Project 1

Introduction to Operating Systems

Caleb Kirby | John Baker | Dax Patel | Parth Patel

Introduction

The purpose of this project is to implement a command line interpreter or shell. The shell should be able to operate on both interactive and batch mode. Ubuntu virtual machine will be used to implement the shell. Subsequent projects will build further upon this for the remainder of the semester.

Requirements

Following are the brief requirements of the project:

- The shell should operate in both batch and interactive mode.
- In interactive mode, the shell prompts the user for command, executes it, and opens up prompt for the next command.
- In batch mode, the shell will read a text file where Each line may contain multiple commands separated with semicolons. Each command separated by a semicolon should be run simultaneously or concurrently.
- Text followed by a pound-sign(#) shall be ignored.
- Users should be able to exit the shell by “quit” or “ctrl + d” command.
- The shell must behave in a reasonable manner if it is not able to perform any operations or encounters a syntax error etc.

Methodology

First of all, a logic was developed to distinguish what mode the shell should run on. If the command line argument count (argc) is greater than 1, and at least one of the options is not preceded with a hyphen, then the shell will attempt to operate in batch mode with input file as command line argument. If this file cannot be found, the shell will exit. After this part, interactive mode was developed.

A. Interactive Mode

In this mode, shell iteratively does the following:

1. The shell prompts the user for command.
2. The input is processed. This step includes validating input, trimming white spaces, and other error-check mechanisms.
3. Then the shell creates a child process that executes the command the user entered.
 - a. The **execv** system call was used for execution of the input command in the child process.
4. Once the child process terminates, the shell prompts for the next user input.
5. Repeat.

Concept of processes learned in the class lecture was used to implement the core functionality of executing the command.

B. Batch Mode

In this mode, shell simultaneously does the following:

1. The shell program is invoked as “shell script-file”
2. The shell reads a text file where each line is executed as a command without a prompt.

When the shell determines it has to operate in batch mode, there is some extra stuff done. First of all, it will check whether the command line argument provided is a file and it exists. If any of these checks fail, the shell will print out appropriate message to the stderr. After, the first line of the file is processed to see if it is a shebang line so that the shell can execute it in the specified way. After, a buffer of file lines are created and executed one after another.

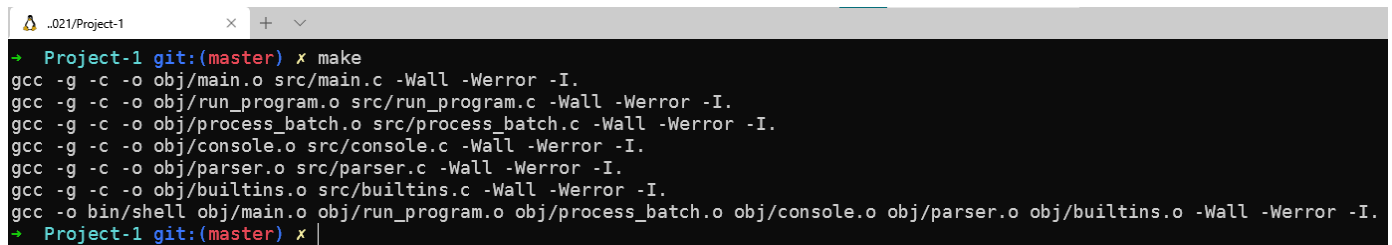
While designing the shell, it became clear that both interactive mode and batch mode will end up executing a line of command at time. The only difference seen from the high level is the batch mode makes it easier to do a set of lines in tandem instead of having to write out one by one very time. As a result, the shell program contains a common function called **process_line**, whose task is to execute a line, regardless of where it comes from. Other functions related to interactive mode and batch mode worked together to process the user input or a file input into a line which can be handled by **process_line**. The shell uses the provided PATH environment variable in the linux based OS to find and execute the commands entered by the user. **Refer to the appendix section for code.**

Results

This section provides working proof of the methodology.

A. Compilation

To compile and build the shell project, make sure you are in the root directory of the project folder and use the *make* command to compile the project. A Virtual machine or a device with linux based OS is required for this to work as expected.



```

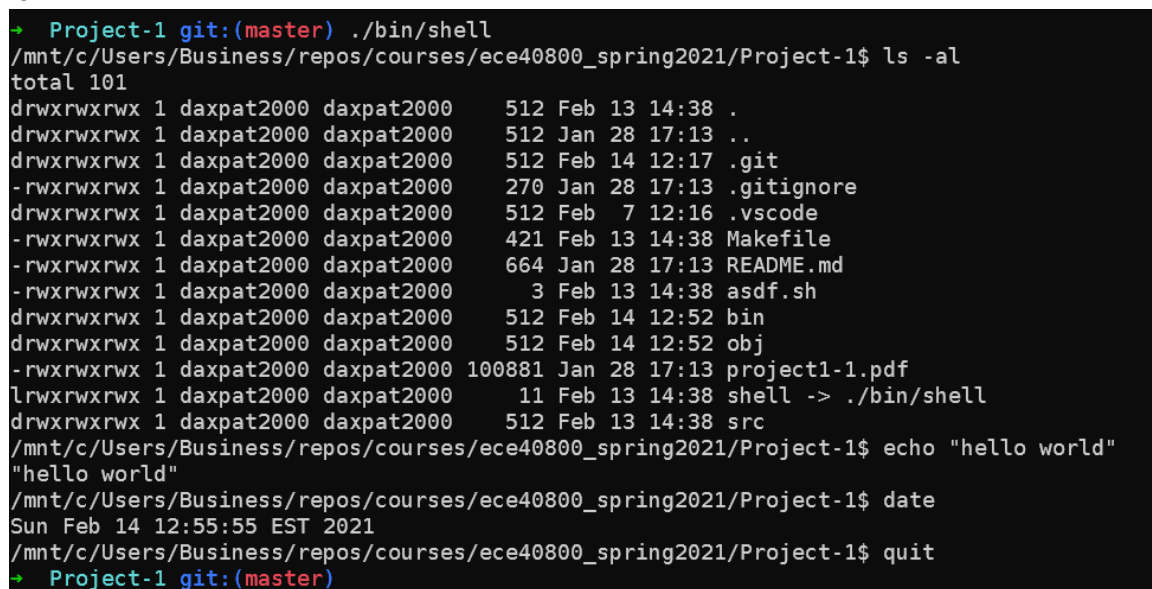
Project-1 git:(master) x make
gcc -g -c -o obj/main.o src/main.c -Wall -Werror -I.
gcc -g -c -o obj/run_program.o src/run_program.c -Wall -Werror -I.
gcc -g -c -o obj/process_batch.o src/process_batch.c -Wall -Werror -I.
gcc -g -c -o obj/console.o src/console.c -Wall -Werror -I.
gcc -g -c -o obj/parser.o src/parser.c -Wall -Werror -I.
gcc -g -c -o obj/builtins.o src/builtins.c -Wall -Werror -I.
gcc -o bin/shell obj/main.o obj/run_program.o obj/process_batch.o obj/console.o obj/parser.o obj/builtins.o -Wall -Werror -I.
Project-1 git:(master) x |

```

Figure 1 Build Shell Using Make

B. Interactive Mode

Following the output for interactive mode



```

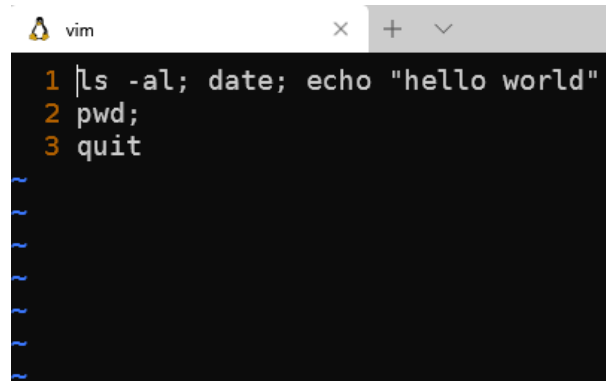
Project-1 git:(master) ./bin/shell
/mnt/c/Users/Business/repos/courses/ece40800_spring2021/Project-1$ ls -al
total 101
drwxrwxrwx 1 daxpat2000 daxpat2000 512 Feb 13 14:38 .
drwxrwxrwx 1 daxpat2000 daxpat2000 512 Jan 28 17:13 ..
drwxrwxrwx 1 daxpat2000 daxpat2000 512 Feb 14 12:17 .git
-rwxrwxrwx 1 daxpat2000 daxpat2000 270 Jan 28 17:13 .gitignore
drwxrwxrwx 1 daxpat2000 daxpat2000 512 Feb 7 12:16 .vscode
-rwxrwxrwx 1 daxpat2000 daxpat2000 421 Feb 13 14:38 Makefile
-rwxrwxrwx 1 daxpat2000 daxpat2000 664 Jan 28 17:13 README.md
-rwxrwxrwx 1 daxpat2000 daxpat2000 3 Feb 13 14:38 asdf.sh
drwxrwxrwx 1 daxpat2000 daxpat2000 512 Feb 14 12:52 bin
drwxrwxrwx 1 daxpat2000 daxpat2000 512 Feb 14 12:52 obj
-rwxrwxrwx 1 daxpat2000 daxpat2000 100881 Jan 28 17:13 project1-1.pdf
lrwxrwxrwx 1 daxpat2000 daxpat2000 11 Feb 13 14:38 shell -> ./bin/shell
drwxrwxrwx 1 daxpat2000 daxpat2000 512 Feb 13 14:38 src
/mnt/c/Users/Business/repos/courses/ece40800_spring2021/Project-1$ echo "hello world"
"hello world"
/mnt/c/Users/Business/repos/courses/ece40800_spring2021/Project-1$ date
Sun Feb 14 12:55:55 EST 2021
/mnt/c/Users/Business/repos/courses/ece40800_spring2021/Project-1$ quit
Project-1 git:(master)

```

Figure 2 Interact with the shell

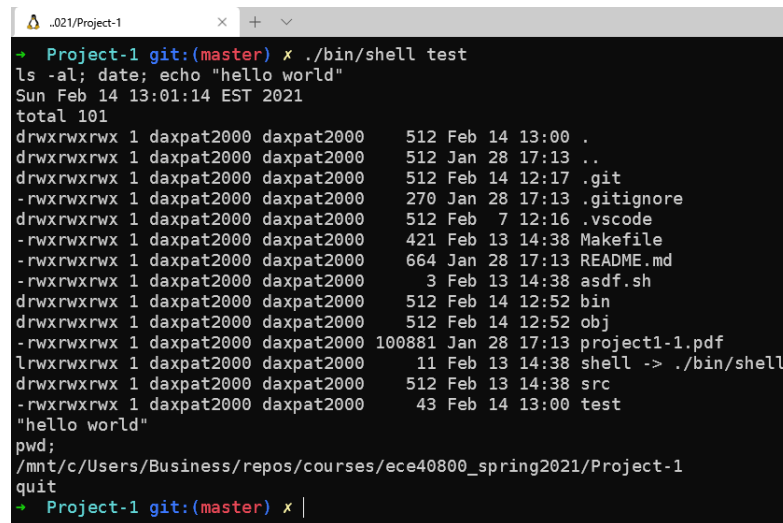
C. Batch Mode

Following is the file input which will be fed to our shell.



```
vim
1 |ls -al; date; echo "hello world"
2 pwd;
3 quit
```

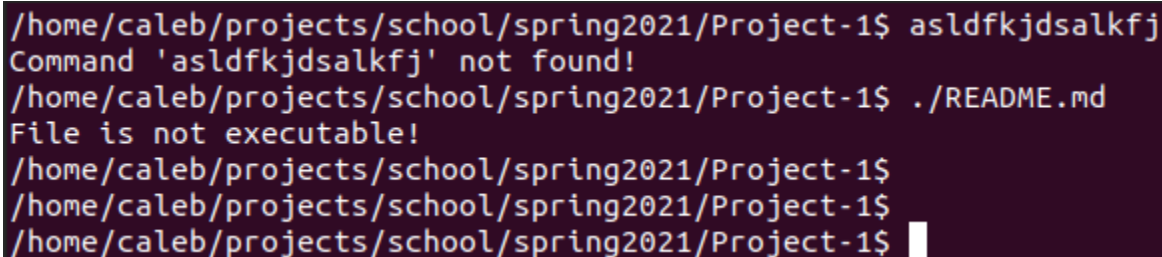
Figure 3 Example of a script file for the shell



```
.021/Project-1
→ Project-1 git:(master) x ./bin/shell test
ls -al; date; echo "hello world"
Sun Feb 14 13:01:14 EST 2021
total 101
drwxrwxrwx 1 daxpat2000 daxpat2000 512 Feb 14 13:00 .
drwxrwxrwx 1 daxpat2000 daxpat2000 512 Jan 28 17:13 ..
drwxrwxrwx 1 daxpat2000 daxpat2000 512 Feb 14 12:17 .git
-rwxrwxrwx 1 daxpat2000 daxpat2000 270 Jan 28 17:13 .gitignore
drwxrwxrwx 1 daxpat2000 daxpat2000 512 Feb 7 12:16 .vscode
-rwxrwxrwx 1 daxpat2000 daxpat2000 421 Feb 13 14:38 Makefile
-rwxrwxrwx 1 daxpat2000 daxpat2000 664 Jan 28 17:13 README.md
-rwxrwxrwx 1 daxpat2000 daxpat2000 3 Feb 13 14:38 asdf.sh
drwxrwxrwx 1 daxpat2000 daxpat2000 512 Feb 14 12:52 bin
drwxrwxrwx 1 daxpat2000 daxpat2000 512 Feb 14 12:52 obj
-rwxrwxrwx 1 daxpat2000 daxpat2000 100881 Jan 28 17:13 project1-1.pdf
lrwxrwxrwx 1 daxpat2000 daxpat2000 11 Feb 13 14:38 shell -> ./bin/shell
drwxrwxrwx 1 daxpat2000 daxpat2000 512 Feb 13 14:38 src
-rwxrwxrwx 1 daxpat2000 daxpat2000 43 Feb 14 13:00 test
"hello world"
pwd;
/mnt/c/Users/Business/repos/courses/ece40800_spring2021/Project-1
quit
→ Project-1 git:(master) x |
```

Figure 4 Shell's execution of script file in Figure 3.

D. Error Handling



```
/home/caleb/projects/school/spring2021/Project-1$ asldfkjdsalkfj
Command 'asldfkjdsalkfj' not found!
/home/caleb/projects/school/spring2021/Project-1$ ./README.md
File is not executable!
/home/caleb/projects/school/spring2021/Project-1$
/home/caleb/projects/school/spring2021/Project-1$
/home/caleb/projects/school/spring2021/Project-1$
```

Figure 5 Error handling done by the shell

Conclusion

In this project, our group created a shell program which operated in two modes: Interactive, where the user types in a command and shell immediately executes it and shows the result, and Batch mode where user can feed the shell with lines of commands written in a file, and shell executes each command. The shell was developed such that both modes depend on a single function for executing the commands, while other parts of the program build up to that function.

Appendix