جامعـة نيويورك أبـوظـبي

NYU | ABU DHABI

CS-UH 3010
Operating Systems, Spring 2025

Project Phase 1

Hariharan Janardhanan and Moujahid Moussa

Architecture and Design

The system is structured as a simple shell implementation with the following key components:

1. Command Parser (command_parser.c and command_parser.h)
2. Executor (executor.c and executor.h)
3. Main program (main.c)

Key design decisions:

- Separation of parsing and execution logic for better modularity
- Support for input/output redirection and piping
- Use of fork() and execvp() for command execution
- Dynamic memory allocation for input handling

File structure and code organization:

- command_parser.h: Defines constants and function prototype for parsing
- command_parser.c: Implements the command parsing logic
- executor.h: Declares functions for command execution
- executor.c: Implements single and piped command execution
- main.c: Contains the main loop for the shell

Implementation Highlights

1. Command Parsing:
    - The parse_command function in command_parser.c tokenizes the input and handles redirections.
    - It supports input (<), output (>), and error (2>) redirections.

    ```
    int parse_command(char *cmd, char **args, int *input_fd, int *output_fd, int
    *error_fd) {

        // Tokenization and redirection handling logic

    }
    ```

2. Command Execution:
    - Single commands are executed using execute_command in executor.c.
    - Piped commands are handled by execute_piped_commands.
    - Both functions use fork() to create child processes and execvp() for command execution.
3. Error Handling:
    - The code checks for various error conditions, such as empty commands, missing files for redirection, and pipe creation failures.
    - Appropriate error messages are displayed using perror() or fprintf() to stderr.

    ```
    if (input_fd == -1) {

        perror("Error opening input file");

        return -1;

    }
    ```

4. Main Loop:
    - The main program in main.c continuously prompts for user input.
    - It splits the input into commands based on the pipe symbol (|).
    - Based on the number of commands, it calls either execute_command or execute_piped_commands

Execution Instructions

To execute the code for the first time. Using the Linux Shell.
Type:

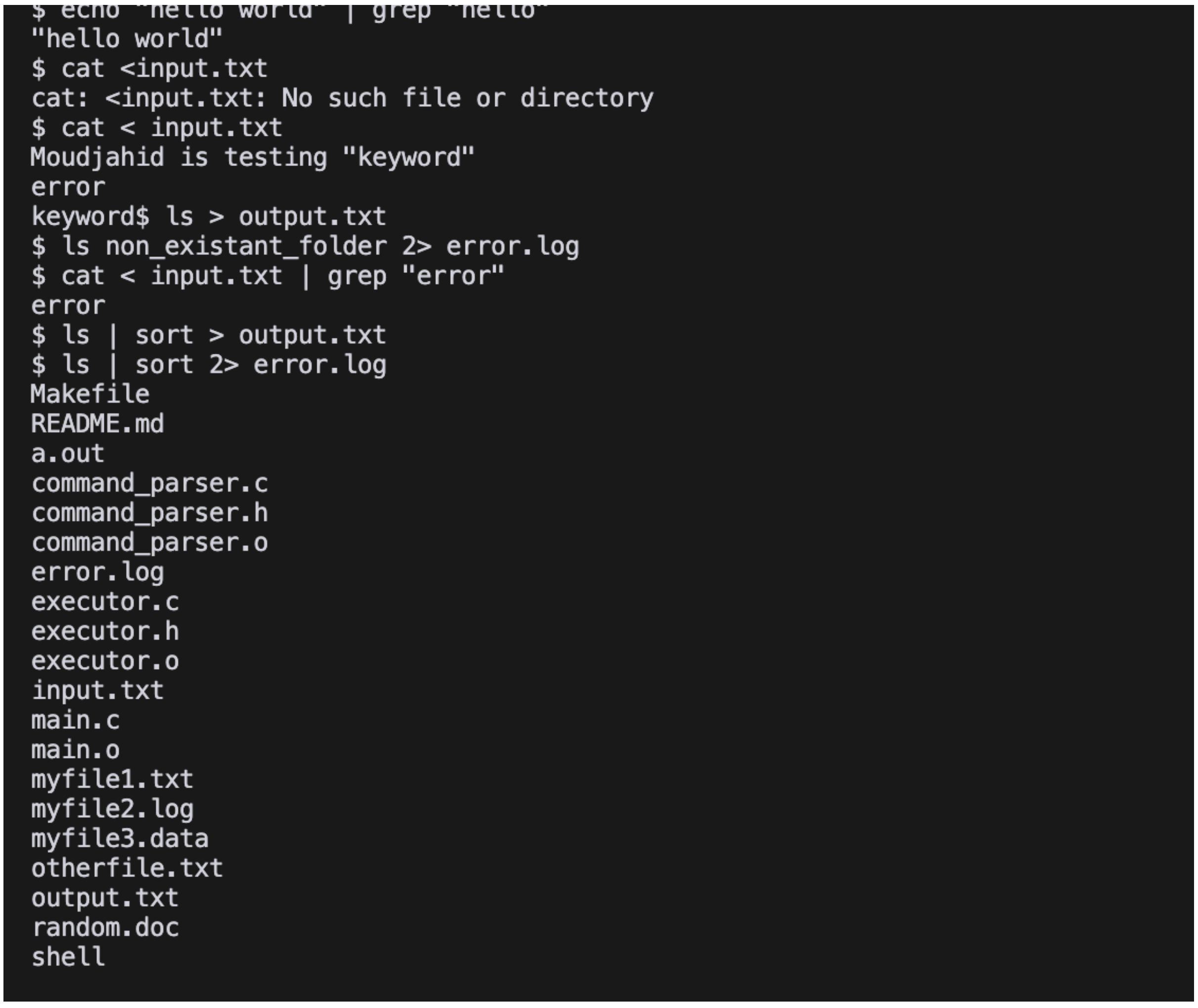
make run

To exit the shell:
exit

To clear and clean:
make clean

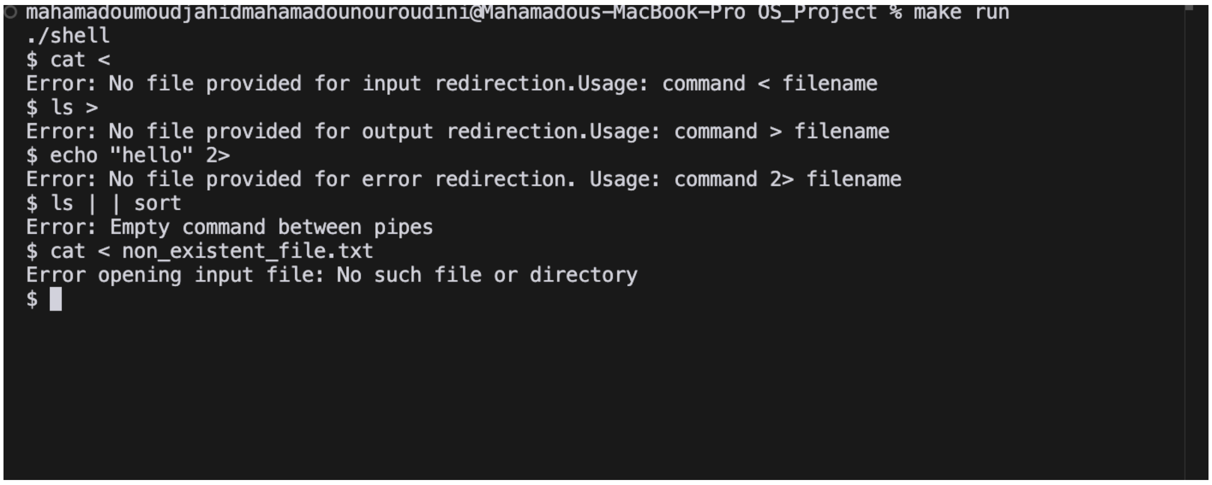
Testing

Below are the tests performed. We covered two testing phases, the first was to try all possible commands without errors and see output. It was successful and the screenshot can be found below. For each of the commands, the intended action was performed in the files. Some of the files modified are also included in the submitted Zip file.



```
$ echo "hello world" | grep "hello"
"hello world"
$ cat <input.txt
cat: <input.txt: No such file or directory
$ cat < input.txt
Moudjahid is testing "keyword"
error
keyword$ ls > output.txt
$ ls non_existant_folder 2> error.log
$ cat < input.txt | grep "error"
error
$ ls | sort > output.txt
$ ls | sort 2> error.log
Makefile
README.md
a.out
command_parser.c
command_parser.h
command_parser.o
error.log
executor.c
executor.h
executor.o
input.txt
main.c
main.o
myfile1.txt
myfile2.log
myfile3.data
otherfile.txt
output.txt
random.doc
shell
```

The second phase of our testing was to check for errors and how they are handled. Below is a screenshot of the tests. We checked for no files provided, no existing file between pipes etc.



```
mahamadoumoudjahidmahamadounouroudini@Mahamadous-MacBook-Pro OS_Project % make run
./shell
$ cat <
Error: No file provided for input redirection.Usage: command < filename
$ ls >
Error: No file provided for output redirection.Usage: command > filename
$ echo "hello" 2>
Error: No file provided for error redirection. Usage: command 2> filename
$ ls | | sort
Error: Empty command between pipes
$ cat < non_existent_file.txt
Error opening input file: No such file or directory
$
```

Challenges

During the development of myshell, our team encountered several significant challenges that required careful consideration and innovative problem-solving:

1. Edge Case Handling
   - Challenge: Parsing complex command structures with multiple redirections and pipes required extensive error checking and edge case handling to ensure reliability.
   - Solution: We developed a comprehensive parsing algorithm that tokenizes input while accounting for quoted arguments and special characters. We also implemented thorough input validation to catch and report syntax errors before command execution.
2. Process Management
   - Challenge: Coordinating multiple child processes for piped commands while avoiding deadlocks and resource leaks was particularly challenging, especially when dealing with long pipelines.
   - Solution: We implemented a careful process creation and monitoring system using fork() and waitpid(). We also used signal handling to manage process termination and prevent zombie processes.
3. Error Propagation
   - Challenge: Ensuring that errors in one part of a pipeline were properly propagated and reported to the user without breaking the entire shell session.
   - Solution: We implemented a hierarchical error handling system that captures and reports errors at various levels of command execution, from parsing to process creation and execution.
4. File Descriptor Management
   - Challenge: Ensuring proper handling of file descriptors for redirections and pipes proved to be complex, especially when dealing with multiple redirections in a single command.
   - Solution: We implemented a robust file descriptor tracking system, using a custom data structure to manage open file descriptors throughout the command execution process. This allowed us to properly close unused descriptors and avoid resource leaks.

These challenges pushed us to deepen our understanding of Unix system programming and forced us to think critically about robust software design. Overcoming them has resulted in a more reliable and efficient shell implementation

Division of Tasks

The project tasks were divided between the two team members as follows:
Moudjahid:

- Successful compilation with a Makefile on remote Linux Server
- Single Commands (without & with arguments)
- Modularization and code refactoring
- Error Handling
- Comments

Hariharan:

- Input, output and error redirection
- Pipes implementation
- Composed Compound Commands
- Report preparation

References

Using dup2: https://stackoverflow.com/questions/1720535/practical-examples-use-dup-or-dup2
File for output redirect :
https://stackoverflow.com/questions/15798450/open-with-o-creat-was-it-opened-or-created

2
–
–
.