Project Description: Simple Shell Program

Overview

This project involves creating a **simple Linux shell** using C.
The shell will demonstrate core operating system concepts such as:

- process creation using `fork()`
- program loading with the `exec` family of functions
- waiting for process completion using `wait()` or `waitpid()`
- file descriptor manipulation for input/output redirection

The goal is to simulate how a Unix shell interprets commands, launches processes, and redirects standard input/output.

Supported Features Summary

Your shell implementation must support the following features:

- **Shell prompt and input** – Display `myshell>` prompt and read user commands
- **Command parsing** – Tokenize input, handle multiple spaces
- **External command execution** – Run programs using `fork()` + `execvp()`
- **Built-in commands** – `cd` and `exit`
- **Output redirection** – > to write to file (overwrite)
- **Input redirection** – < to read from file
- **Append redirection** – >> to append to file
- **Quoted strings** – `"hello world"` preserves spaces as single argument

Project Objectives

- To understand how a command-line shell interacts with the operating system.
- To implement basic command parsing and argument handling.
- To execute external programs using `fork()` and `execvp()`.
- To implement I/O redirection using `dup2()` and `open()`.
- To handle built-in commands like `cd` and `exit` within the shell.
- To compile and run the program using a Makefile.
- To practice software development, GitHub usage, and submission workflow.

Task Description

1. Shell Command Parsing

Your shell must read a line of input from the user and interpret it as a command. The shell should support:

- Multiple spaces or tabs between tokens
- Commands with arguments (`ls -l /home`)
- Quoted strings (`echo "hello world"`)
- Input redirection (<)
- Output redirection (>)
- Append redirection (>>)

The shell will parse these into a structured format that child processes can execute.

2. Executing Commands

Your shell must:

- Use `fork()` to create a new child process.
- In the child process, use `execvp()` to run external commands.
- Support commands such as:
    - `ls`, `echo`, `pwd`, `cat`, `env`, etc.
- Display an error message if a command does not exist.

3. Implementing I/O Redirection

Your shell must support redirecting standard input and output:

Input Redirection (<)

Example:

```
myshell> sort < names.txt
```

Child process behavior:

- Open the file
- Replace `STDIN_FILENO` with the file using `dup2()`
- Execute the command normally

Output Redirection (>)

Example:

```
myshell> echo hello > out.txt
```

Child process behavior:

- Overwrite or create the file
- Redirect standard output to the file via `dup2()`

Append Redirection (>>)

Example:

```
myshell> echo line >> log.txt
```

Child process:

- Open file in append mode
- Redirect output using `dup2()`

4. Built-in Command Handling

The shell must implement two built-in commands:

`cd`

- Change the shell's current working directory using `chdir()`.
- Must run in the **parent**, not in the child process.

`exit`

- Immediately terminates the shell.

5. Process Synchronization

After launching an external command:

- The parent shell must wait for the child process to finish.
- Use `wait()` or `waitpid()` to collect the child's exit status.
- The shell should not terminate until the user explicitly types `exit`.

6. Makefile

You must provide a Makefile that:

- Compiles your code into an executable named `myshell`.
- Allows compilation with a single command:

7. Test Cases

Use the following test cases to verify your shell implementation:

Basic Commands

- `ls` – List directory contents
- `echo hello world` – Print multiple arguments
- `echo    a       b` – Multiple spaces (should print “a b”)

Quoted Strings

- `echo "hello world"` – Preserves spaces inside quotes
    - expect to print `hello world`, without `""`.
- `touch "my file.txt"` – Creates file with space in name
    - expect to see `my file.txt` in the list.

Built-in Commands

- `cd /tmp` then `pwd` – Should show /tmp
- `cd` – Goes to HOME directory
- `exit` – Terminates the shell

Output Redirection (>)

- `echo hello > out.txt` then `cat out.txt`
- `ls -la > listing.txt`

Append Redirection (>>)

- `echo line1 > log.txt`
- `echo line2 >> log.txt`
- `cat log.txt` – Should show both lines

Input Redirection (<)

- `cat < log.txt` – Reads from file
- `wc -l < log.txt` – Counts lines from file

Error Handling

- `nonexistent_command` – Should show error message
- `cd /nonexistent_path` – Should show cd error
- `cat < nonexistent.txt` – Should show file open error

Submission Requirements

- A single C file (myshell.c) with your full implementation.
- A Makefile that compiles your program into an executable called myshell.
- At 3 Screenshots for your results of Tasks example.
- Upload everything to GitHub and submit GitHub link to Canvas.