

OS Assignment 2

Ishaan Agrawal and Aniket Gupta

September 10, 2023

1 Individual Contributions

- Aniket Gupta (2022073): Sub-parts a, b, c, d, e and f
- Ishaan Agrawal (2022221): Sub-parts g, h, Bonus parts a and b

2 GitHub Repository

GitHub Repository Link.

3 How to Run

1. Go to the appropriate directory (/CSE231/Assignment 2/SimpleShell)

2. Compile the shell using

```
1 make shell
```

3. Run the shell using

```
1 ./shell
```

4. To terminate the shell, use Ctrl + C.

5. If you have multiple commands to run, write them in a .sh file commands.sh, and give the command

```
1 run commands.sh
```

while in the shell.

6. To delete the shell executable

```
1 make clean
```

4 Implementation

The Simple Shell is crafted in C, consisting of several modular components to ensure seamless operation. The implementation is divided into multiple C source files:

- simple-shell.c
- read_user_input.c
- create_process_and_run.c
- piped_commands.c
- background_commands.c
- history.c

In addition, two header files are utilized:

- `simple-shell.h`: It consolidates the essential libraries required for the entire implementation.
- `history.h`: This file features a global array dedicated to preserving user commands, complete with execution times and other pertinent parameters, throughout a single session.

4.1 Main Loop

Inside `simple-shell.c`, the core of the program resides in an infinite do-while loop, punctuated by an interactive prompt for the user. The `read_user_input()` function captures valid user input using `fgets`. The input command is then dispatched to the `launch` function, which in turn calls `create_process_and_run`.

4.2 Signal Handling

Implemented in the same file, signal handling adeptly captures a Ctrl + C interrupt. This event is promptly relayed to a custom handler through `sigaction`.

4.3 Running Shell Scripts

The `run_sh_file` function is designed to read and iterate through a Bash file, executing its commands.

4.4 Command Execution

In `create_process_and_run`, the method first checks if the command has any special characters like `'|'`, or `'&'`. If so, it deftly dispatches the execution to either `pipe_commands()` or `background_commands()`, neatly compartmentalizing and executing processes between those delimiters. If the command lacks any such special characteristics, the process is launched conventionally using `fork`, with the child process executing the command using `execv`.

4.5 Sample Files

Included are sample C files, `fib.c` and `helloworld.c`, as well as a bash script, for testing purposes.

5 Limitations

Following are some commands which our shell is unable to run.

1. `execl` cannot directly execute shell built-in commands, as they are interpreted by the shell itself. Examples of shell built-in commands include `alias`, `exit`, and `export`. System utilities like `'ifconfig'`, `'traceroute'`, or any custom executables not in the system's `PATH` won't be accessible through our shell. The `cd` command would also not work, but we have specially handled it using the `chdir` system call.
2. We have explicitly handled `&` and pipe commands with `execvp`. As a result, attempting to run in built shell commands in the background or as a piped process would lead to either a "command not recognised error, or some unintended output. For example, our shell does not support `history &`, `'history | grep clear'` etc.
3. Commands that involve shell-specific features like redirections, and conditional execution (e.g. `command > output.txt`, `command1 && command2`) may not work as expected because `execl` does not invoke a shell to interpret these features.
4. `execl` does not provide a way to set or modify environment variables directly. You might need to use `execle` or `execve` with custom environment variables if required.
5. "commands containing both `'&'` and `'|'` won't work since it would get really complicated to handle, and our current implementation is designed specifically to support commands where these operators don't appear together. In order to enable these kinds of commands, significant restructuring and additional logic would be required to manage the concurrent execution and piped communication between processes."