SimpleShell_Project 2_Report

Group members:

- **1751064 Nguyen Hoang Gia**
- **1751063 Pham Bao Duy**

GitHub Repository link:

https://github.com/nhgia/SimpleShell

I. README.md

Prerequisite

- Ubuntu / Linux. (Recommended Ubuntu 18.04)
- Download or clone the project.
- Install essential components by running:

```
sudo apt-get install build-essential
```

- Change directory to the project folder.

Install: Compile the C file into output and runnable

- Compile through Makefile, run

```
make
```

- Run the shell

```
./simple-shell
```

SimpleShell commands and features

1. Single command

- List sub-folder & sub-item

```
ls
```

- Current directory

```
pwd
```

- Ping website

```
ping -i 5 -w 10 -W 10 google.com
```

- ...

2. Check history of run commands

- Using double exclamation mark "!!" for checking history

```
!!
```

3. Concurrence commands

- Add an ampersand mark after the command with whitespace required.

```
ping -i 5 -w 10 -W 10 google.com &
```

4. I/O redirection

- Redirect the output of a command to a file and input from a file to a command.

```
ls > myFile.txt
cat < myFile.txt
```

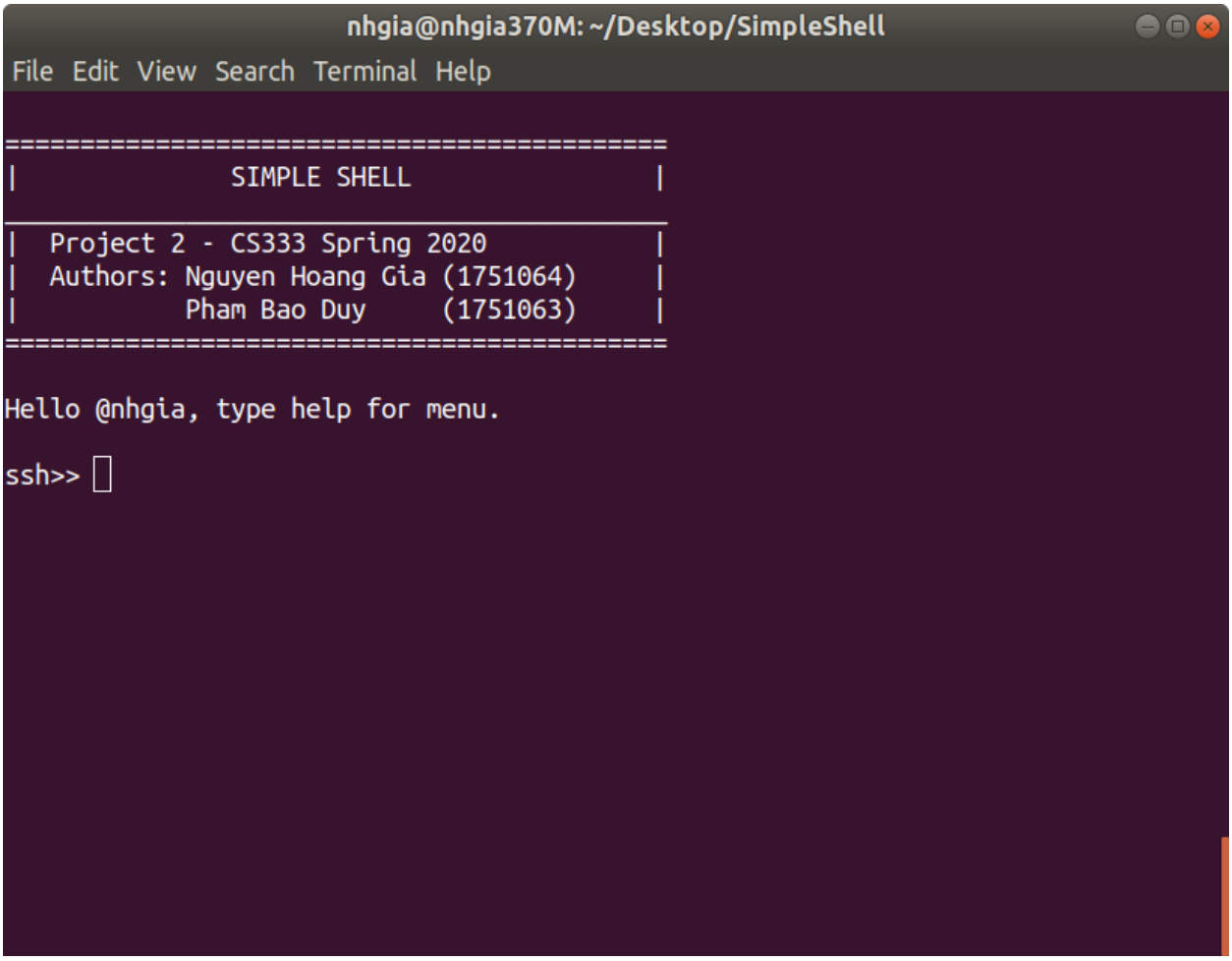
5. Communication via a Pipe

- Output of one command to serve as input to another using a pipe.
- For example: list sub-folder and sub-item in current directory, just display which has character "a" in the name.

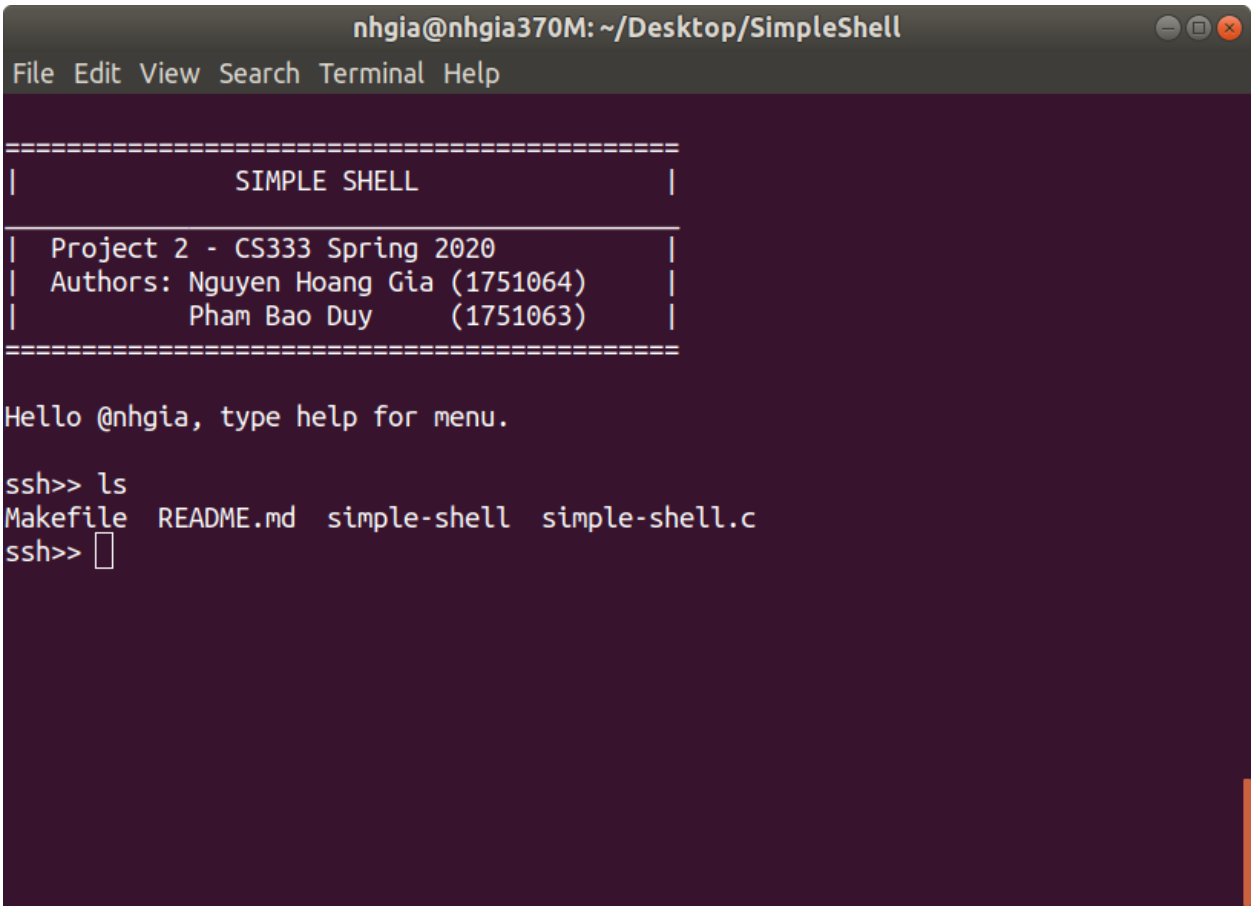
```
ls | grep a
```

II. Result report

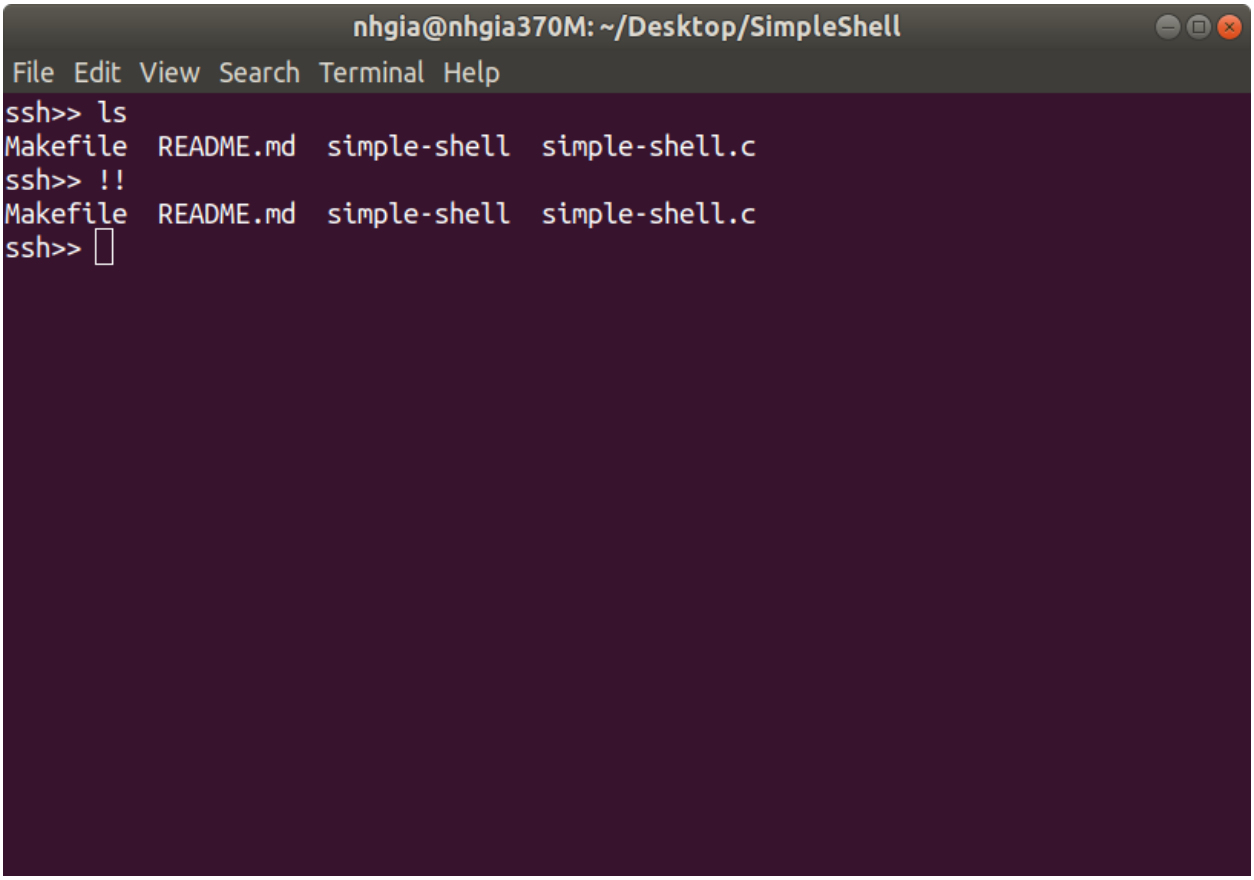
1. Main (after running ./simple-shell)



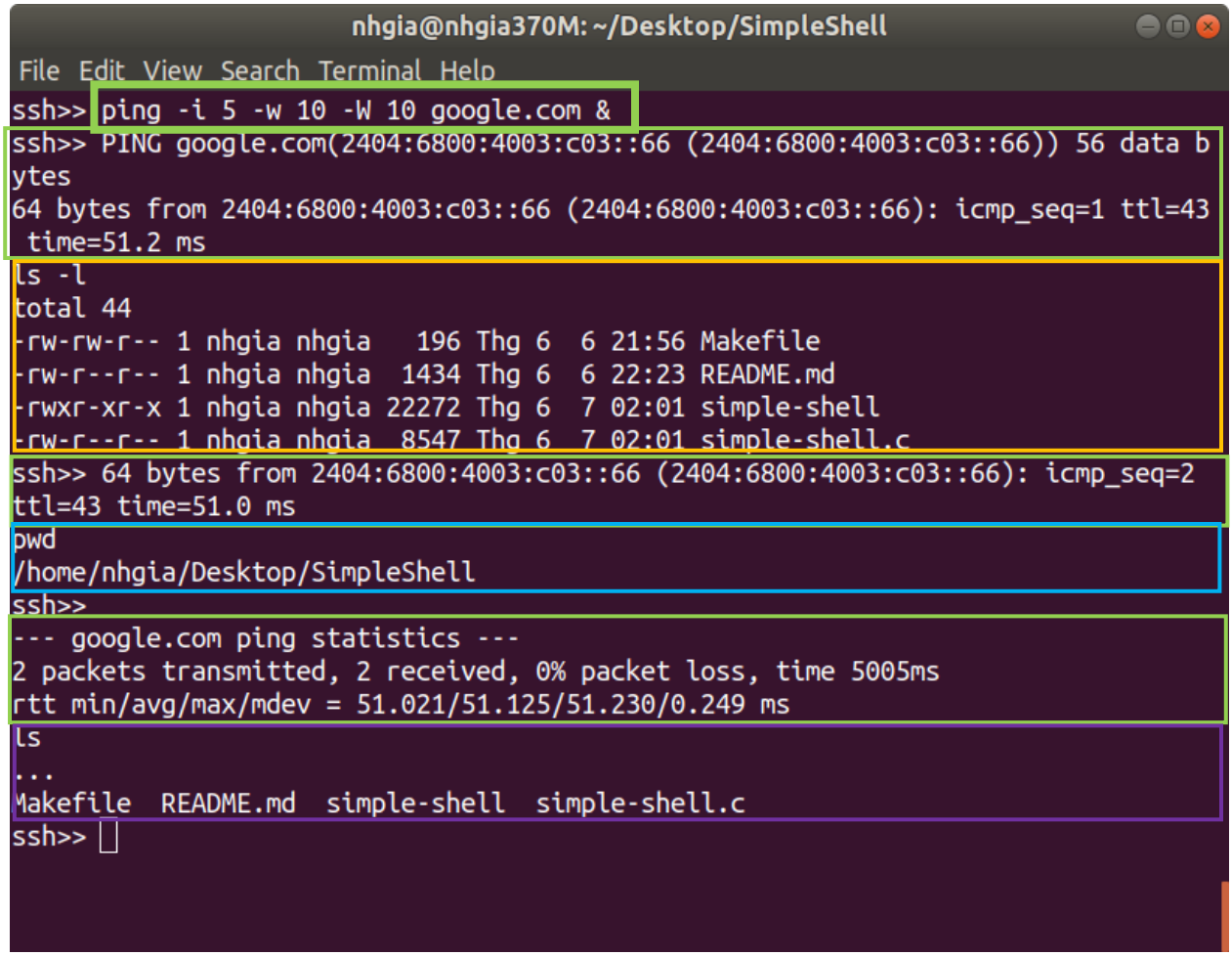
2. Single command with folk() child process



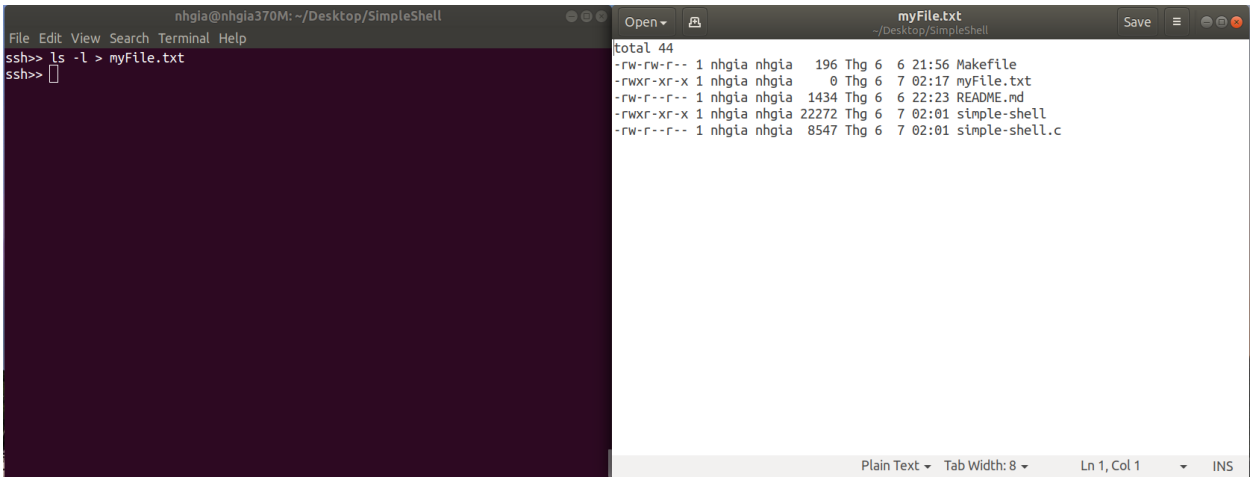
3. History by “!!”



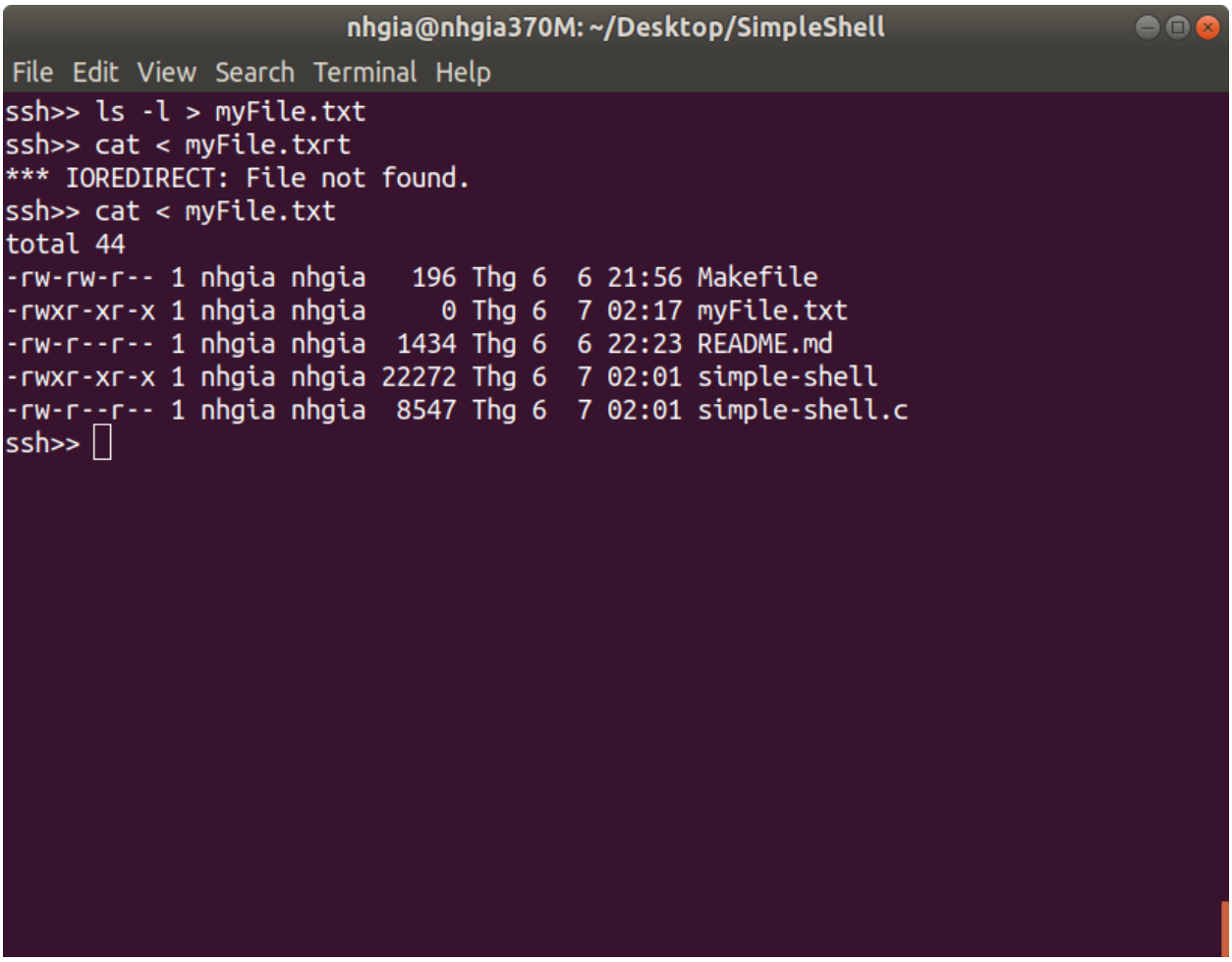
4. Concurrent conmmands



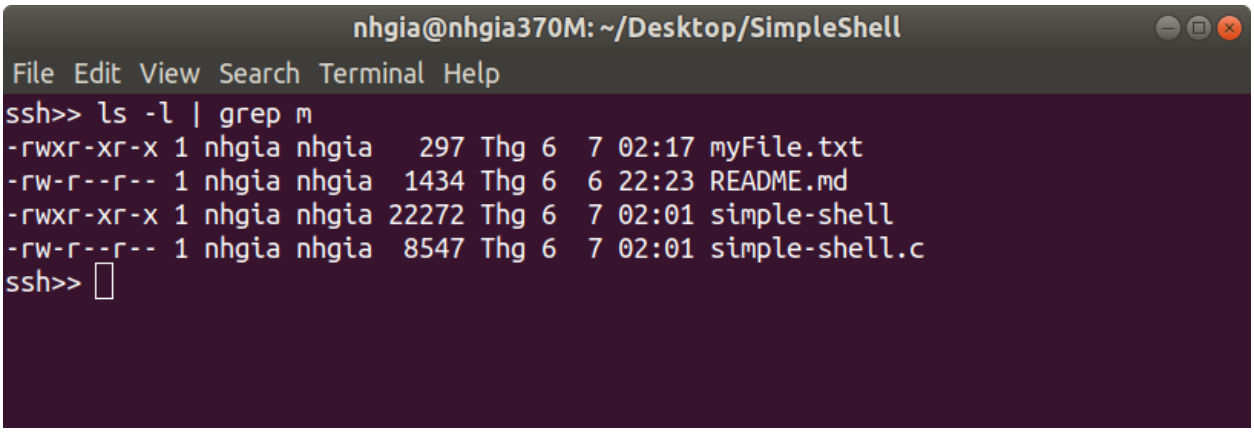
5. Output redirection



6. Input redirection



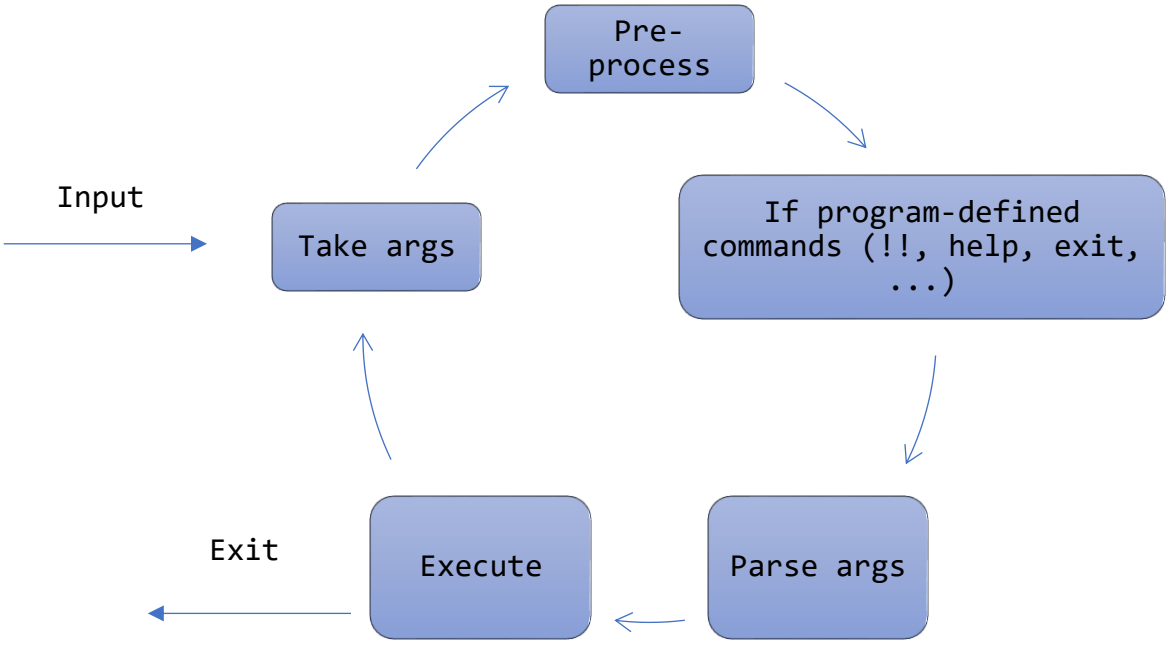
7. Communication via a Pipe



III. Code review

Overall, the program at run time works like normal C program but take input argument from user entered via Terminal.

The main work-flow of simple-shell can be illustrate by this:



- Pre-process: Flush the input-stream, get new one from fget(), add '\0' for end of user input.
- Next, check whether the input (first argument or args[0]) is special case or not. This includes show history ("!!" command), help menu, exit program, clear screen.
- Function parse() takes an input line and returns a zero-terminated array of char pointers, each of which points to a zero-terminated character string. This function loops until a binary zero is found, which means the end of the input line "command" is reached. If the current character of "command" is not a binary zero, parse() skips all white spaces and replaces them with binary zeros so that a string is effectively terminated. Once parse() finds a non-white space, the address of that location is saved to the current position of "args" and the index is advanced. Then, parse() skips all non-whtitespace characters. This process repeats until the end of string "command" is reached and at that moment "args" is terminated with a zero.
- The execute() function: After get everything from parse, now the argument passed into execute() is an array of character, which element is a part of input. For example: `ls -l | grep a` will be `args[0] = "ls", args[1] = "-l", args[2] = "|", args[3] = "grep", args[4] = "a"`. Next, it continues to check whether it is pipe (contains "|") or IO redirect (contains "<" or ">"),…
- After got what kind of command from input, we fork child process to execute input command through folk() function. Thanks for the pid_t data type and also epending on which kind of command, we separate child process logic:

(after fork 1st child process)

<table>
<tr><td>pid_t</td><td>fork() > 0
At parent process</td><td colspan="2">fork() == 0
At child process</td><td>fork() < 0
Error</td></tr>
<tr><td>Single command with child process</td><td>Wait for child process to be killed.</td><td colspan="2" rowspan="2">Execute command.</td><td rowspan="5">Exit()</td></tr>
<tr><td>Concurrent</td><td>Return to parent process immediately, no need to wait.</td></tr>
<tr><td>IO Redirect</td><td>Wait for child process to be finished.</td><td colspan="2">• If output: open file, redirection using dup2() function (send output to opened file).
• If input: open file, redirection using dup2() function (send file to input).</td></tr>
<tr><td rowspan="3">Pipe</td><td rowspan="3">Execute 1st command.</td><td>#1 child</td><td>#2 child</td></tr>
<tr><td>Fork 2nd child process.
Write to child #2.</td><td>Read from child #1.
Execute 2nd command.</td></tr>
<tr><td colspan="2">Wait for both child process to be killed.</td></tr>
</table>

- END -