COMP 4511 - System and Kernel Programming in Linux Programming Assignment #3

Assignment 3 - Simple Linux Shell (Part 2)

You should modify the simple Linux Shell (Part 1) with the addition of 2 key features:

- I/O redirection
- shell pipes

Please note that you should not break existing features in your submitted part 1.

Feature 1 - I/O Redirection

To accomplish this feature, you need to use the `dup()` or `dup2()` (recommended).

A process has 3 default file identifiers:

- `stdin` (standard input)
- `stdout` (standard output)
- and `stderr` (standard error output)

If the process reads from `stdin`, then the data that it receives will be directed from the keyboard to the `stdin` file descriptor. Similarly, data received from `stdout` and `stderr` are mapped to the terminal display. In a typical shell program, a user can redefine `stdin` or `stdout` to another input/output source. A "less than" character (i.e. '<') followed by a filename is used for input redirection. Similarly, a "greater than" character (i.e. '>') followed by a filename is used for output redirection.

For example, a command such as:

```
[DIR] myshell> test < main.cpp > output.txt
```

This command will create a child process to execute the "`test`" command. Before the "`test`" command is launched, `stdin` will be redirected to read the input stream from "`main.cpp`" and `stdout` will be redirected to write the output stream to "`output.txt`".

The shell can redirect I/O by manipulating the child process's file descriptors. A newly created child process inherits the open file descriptors of its parent, specifically the same keyboard for `stdin` and the terminal display for `stdout` and `stderr`.

The shell can change the child's file descriptors so that it reads and writes to files rather than to the keyboard and display.

Feature 2 - Shell Pipes

A pipe is represented in the kernel by a file descriptor, just like a file. A process creates a pipe by:

```
int ps[2];
pipe(ps);
```

Here, `ps[0]` is a file pointer (an index into the process's open file table) to the read end of the pipe and `ps[1]` is the file pointer to the write end of the pipe. For two or more processes to use pipes for IPC, a common ancestor of the process must create the pipe prior to creating the processes.

In a shell, a pipe symbol '|' is used to connect the output of the first command to the input of the second command. For example,

```
[DIR] myshell> ls | sort
```

The "`ls`" command lists the contents of the current directory. As the output of "`ls`" is already connected to "`sort`", it won't print out the content to the screen. After the output of `ls` has been sorted by `sort`, the sorted list of files appears on the screen.

Please note that you are required to support multiple pipes. For example:

```
[DIR] myshell> ps aux | grep root | sort | less
```

Sample output for I/O redirection and pipes

Sample output #1

```
[myshell] myshell> ./test < input.txt > out.txt
[myshell] myshell> cat out.txt
Hello World!
Hello World!
Hello World!
Hello World!
myshell> cat out.txt | wc -l
4
myshell>
```

The test program simply reads a file and outputs the file content to the screen. The `input.txt` includes 4 lines of "HelloWorld!"

Sample output #2

myshell should accept more than one pipes and support a combination of output redirection and pipes. In particular, myshell should support multiple pipes, as shown by the following example,

```
[myshell] myshell> ./test < input.txt | head -n 2 | wc -l > output.txt
[myshell] myshell> nano output.txt
```

The content of output.txt now should be 2. Note that the output redirection is always associated with the command it follows and NOT with the whole command line.