[redacted]

Shell

[redacted]

Important!

[redacted]

You **must not** "fork bomb" the [redacted] server. Doing so will **hurt your grade**. A 25-point deduction will be applied for these offenses. You **must** ensure that you are working on one of the vcf cluster nodes when developing and testing your code.

[redacted]

1 Project Description

For this project you will implement a simple UNIX shell (similar to BASH) that supports basic functionality related to launching processes, redirecting input/output, and more.

This project may be viewed as an extension to lab 06. Your code for lab 06 may be helpful as you work on the project.

1.1 Prompt [15 points]

Your shell should read commands from the standard input. Your shell's prompt should include the text "`1730sh:`" followed by the current working directory, the `$` symbol, and a single space. If the current working directory contains the user's home directory, then ~ should be used.

Examples (note the ~ and the space after the `$`):

```
1730sh:~$
```

```
1730sh:~/mydir$
```

```
1730sh:/home/myid$
```

If the user does not type a command but simply hits "Enter", your shell should display a new prompt and awaits the next command.

The shell should continue to loop on user input until the `exit` command is issued by the user.

1.2 Change Directory [10 points]

The user should be able to use the `cd` command to change the current working directory. The change should be reflected in your shell's prompt. Upon initial launch of your shell program, please set the current working directory to the user's home directory. **Please do not hard code your own user home directory into your source code**. You should use a function such as `getenv(3)` to retrieve the user home directory.

The following use cases of `cd` should be supported:

```
1730sh:∼$ cd
```

```
1730sh:∼$ cd ∼
```

```
1730sh:∼$ cd mydir
```

```
1730sh:∼$ cd ..
```

1.3 Launch Processes [30 points]

Your shell should support launching processes via the `fork(2)` and `exec(3)` calls. Your shell should pass arguments to the launched program. For example, in

```
1730sh:∼$ cat file1.txt file2.txt
```

the arguments `file1.txt` and `file2.txt` should be passed to `cat`.

Upon completion of the executed process, your shell should resume its input loop that waits for the next command.

1.4 Redirect I/O [15 points]

Your shell should support standard input and output redirection. The following forms should be supported:

1. Redirect standard input using <

   ```
   1730sh:∼$ COMMAND < input.txt
   ```

   This redirects the standard input of `COMMAND` to come from `input.txt`.

2. Redirect standard output using > (truncate) and >> (append)

   ```
   1730sh:∼$ COMMAND > output.txt
   ```

   This redirects the standard output of `COMMAND` to overwrite `output.txt`.

   ```
   1730sh:∼$ COMMAND >> output.txt
   ```

   This redirects the standard output of `COMMAND` to append to `output.txt`.

3. A command may also redirect both standard input and output:

   ```
   1730sh:∼$ COMMAND < input.txt > output.txt
   ```

   ```
   1730sh:∼$ COMMAND >> output.txt < input.txt
   ```

   In this case, you may assume at most one of the output redirection operators (`>` or `>>`) will be used.

2 Nonfunctional Requirements [30 points]

Your submission needs to satisfy the following nonfunctional requirements:

- **Directory Setup [5 points]:** [redacted] For this project, you should place your source code into a file called `shell.c`. The executable should be named `shell`.
- **Documentations [5 points]:** Your code must be documented using Javadoc style comments. Use inline documentation, as needed, to explain ambiguous or tricky parts of your code.

- `makefile` [**5 points**]: You need to include a `makefile`. The expectation is that the grader should be able to type `make clean` and `make` to clean and compile/link your submission, respectively.

  When you compile, you must pass the following options to `gcc`:

  ```
  -Wall -pedantic-errors
  ```

  Other compiler/linker options may be needed in addition to the ones mentioned above.

- [redacted]

- **Compiler Warnings** [**5 points**]: Since you should be compiling with both the `-Wall` and `-pedantic-errors` options, your code is expected to compile without `gcc` issuing any warnings.

- **Memory Leaks** [**5 points**]: Since this project may (or may not) make use of dynamic memory allocation, you are expected to ensure that your project implementation does not result in any memory leaks. We will test for memory leaks using the `valgrind` utility.

- **Libraries:** You are allowed to use any of the C standard libraries, but with the following restrictions: When reading or writing to a file (including standard input and output) are concerned, you need to use low-level calls to **`read(2)`**, **`write(2)`**, and related functions. Functions like **`scanf(3)`**, **`gets(3)`**, **`puts(3)`**, **`fopen(3)`**, **`fgets(3)`**, **`fscanf(3)`**, and other related **`f`**-functions are **not** low-level I/O functions and therefore cannot be used. The **`printf(3)`** function is acceptable when you need to display simple output messages to the screen. Whenever possible, program output should be unbuffered. You are NOT allowed to use `popen(3)` and `system(3)`. **Failure to adhere to this non-functional requirement will cause you to earn no points for the portions of your program that make use of these forbidden system calls/functions.**

3 Example Usages and Tips

```
1730sh:~$ cd mydir
1730sh:~/mydir$ COMMAND1
1730sh:~/mydir$ COMMAND2 >> output.txt
1730sh:~/mydir$ COMMAND3 arg1 arg2 > output.txt
1730sh:~/mydir$ COMMAND4 arg1 < input.txt
1730sh:~/mydir$ exit
```

Regarding the `exec(3)` family of functions:

	"p" execs	"non-p" execs
/bin/ls	✓	✓
ls	✓	

As seen above, the exec functions which search the path (i.e. those which take a "filename") also work with absolute pathnames. My suggestion is to use those (e.g., execvp(3)).

You may assume that there will be **exactly one** space between tokens in the inputs to your program.

Some system calls and library functions needed for this project may not be explicitly discussed in class. You are encouraged to look up online/textbook, or consult fellow classmates/TAs/instructor if you are unsure what to use in order to fulfill certain project requirements.

[redacted]

[redacted]

[redacted]

Note: the "submit" command may not work on the vcf cluster nodes. You may need to exit back to the main [redacted] server before you can execute the "submit" command.