0x16. C - Simple Shell

C | Group project | Syscall

By: Julien Barbier

Weight: 10

Project to be done in teams of 2 people (your team: Korede Faleye, Wittygod Okoro)

Project will start Nov 2, 2022 6:00 AM, must end by Nov 17, 2022 6:00 AM

will be released at Nov 16, 2022 1:12 AM

An auto review will be launched at the deadline

Concepts

For this project, we expect you to look at these concepts:

- Everything you need to know to start coding your own shell (/concepts/64)
- Approaching a Project (/concepts/350)

Background Context

Write a simple UNIX command interpreter.

^ "The Gates of Shell", by Spencer Cheng (/rltoken/AtYRSM03vJDrko9xHodxFQ), featuring Julien Barbier (/rltoken/-ezXgcyfhc8qU1DeUlnLUA)

Resources

Read or watch:

- Unix shell (/rltoken/f0YU9TAhniMXWlSXtb64Yw)
- Thompson shell (/rltoken/7LJOp2qP7qHUcsOK2-F3qA)
- Ken Thompson (/rltoken/wTSu31ZP1f7fFTJFgRQC7w)
- **Everything you need to know to start coding your own shell** concept page

man or help:

- `sh` (*Run `sh` as well*)

Learning Objectives

At the end of this project, you are expected to be able to explain to anyone (/rltoken/9LNz86CtOTos9oL3zxIO3A), **without the help of Google**:

General

- Who designed and implemented the original Unix operating system
- Who wrote the first version of the UNIX shell
- Who invented the B programming language (the direct predecessor to the C programming language)
- Who is Ken Thompson
- How does a shell work
- What is a pid and a ppid
- How to manipulate the environment of the current process
- What is the difference between a function and a system call
- How to create processes
- What are the three prototypes of `main`
- How does the shell use the `PATH` to find the programs
- How to execute another program with the `execve` system call
- How to suspend the execution of a process until one of its children terminates
- What is `EOF` / "end-of-file"?

Copyright - Plagiarism

- You are tasked to come up with solutions for the tasks below yourself to meet with the above learning objectives.
- You will not be able to meet the objectives of this or any following project by copying and pasting someone else's work.
- You are not allowed to publish any content of this project.
- Any form of plagiarism is strictly forbidden and will result in removal from the program.

Requirements

General

- Allowed editors: `vi`, `vim`, `emacs`
- All your files will be compiled on Ubuntu 20.04 LTS using `gcc`, using the options `-Wall -Werror -Wextra -pedantic -std=gnu89`
- All your files should end with a new line
- A `README.md` file, at the root of the folder of the project is mandatory
- Your code should use the `Betty` style. It will be checked using betty-style.pl (https://github.com/holbertonschool/Betty/blob/master/betty-style.pl) and betty-doc.pl (https://github.com/holbertonschool/Betty/blob/master/betty-doc.pl)
- Your shell should not have any memory leaks
- No more than 5 functions per file
- All your header files should be include guarded
- Use system calls only when you need to (why? (/rltoken/EU7B1PTSy14INnZEShpobQ))
- Write a `README` with the description of your project

- You should have an `AUTHORS` file at the root of your repository, listing all individuals having contributed content to the repository. Format, see Docker (/rltoken/UL8J3kgl7HBK_Z9iBL3JFg)

GitHub

**There should be one project repository per group. If you and your partner have a repository with the same name in both your accounts, you risk a 0% score. Add your partner as a collaborator. **

More Info

Output

- Unless specified otherwise, your program **must have the exact same output** as `sh` (`/bin/sh`) as well as the exact same error output.
- The only difference is when you print an error, the name of the program must be equivalent to your `argv[0]` (See below)

Example of error with `sh`:

```
$ echo "qwerty" | /bin/sh
/bin/sh: 1: qwerty: not found
$ echo "qwerty" | /bin/../bin/sh
/bin/../bin/sh: 1: qwerty: not found
$
```

Same error with your program `hsh`:

```
$ echo "qwerty" | ./hsh
./hsh: 1: qwerty: not found
$ echo "qwerty" | ./././hsh
./././hsh: 1: qwerty: not found
$
```

List of allowed functions and system calls

- `access` (man 2 access)
- `chdir` (man 2 chdir)
- `close` (man 2 close)
- `closedir` (man 3 closedir)
- `execve` (man 2 execve)
- `exit` (man 3 exit)
- `_exit` (man 2 _exit)
- `fflush` (man 3 fflush)
- `fork` (man 2 fork)
- `free` (man 3 free)
- `getcwd` (man 3 getcwd)
- `getline` (man 3 getline)
- `getpid` (man 2 getpid)

- `isatty` (man 3 isatty)
- `kill` (man 2 kill)
- `malloc` (man 3 malloc)
- `open` (man 2 open)
- `opendir` (man 3 opendir)
- `perror` (man 3 perror)
- `read` (man 2 read)
- `readdir` (man 3 readdir)
- `signal` (man 2 signal)
- `stat` (__xstat) (man 2 stat)
- `lstat` (__lxstat) (man 2 lstat)
- `fstat` (__fxstat) (man 2 fstat)
- `strtok` (man 3 strtok)
- `wait` (man 2 wait)
- `waitpid` (man 2 waitpid)
- `wait3` (man 2 wait3)
- `wait4` (man 2 wait4)
- `write` (man 2 write)

Compilation

Your shell will be compiled this way:

```
gcc -Wall -Werror -Wextra -pedantic -std=gnu89 *.c -o hsh
```

Testing

Your shell should work like this in interactive mode:

```
$ ./hsh
($) /bin/ls
hsh main.c shell.c
($)
($) exit
$
```

But also in non-interactive mode:

```
$ echo "/bin/ls" | ./hsh
hsh main.c shell.c test_ls_2
$
$ cat test_ls_2
/bin/ls
/bin/ls
$
$ cat test_ls_2 | ./hsh
hsh main.c shell.c test_ls_2
hsh main.c shell.c test_ls_2
$
```

Checks

The Checker will be released at the end of the project (1-2 days before the deadline). We **strongly** encourage the entire class to work together to create a suite of checks covering both regular tests and edge cases for each task. See task `8. Test suite`.

Tasks

0. Betty would be proud

mandatory

Write a beautiful code that passes the Betty checks

Repo:

- GitHub repository: `simple_shell`

Done? | Help | >_ Get a sandbox

1. Simple shell 0.1

mandatory

Write a UNIX command line interpreter.

- Usage: `simple_shell`

Your Shell should:

- Display a prompt and wait for the user to type a command. A command line always ends with a new line.
- The prompt is displayed again each time a command has been executed.
- The command lines are simple, no semicolons, no pipes, no redirections or any other advanced features.
- The command lines are made only of one word. No arguments will be passed to programs.
- If an executable cannot be found, print an error message and display the prompt again.
- Handle errors.
- You have to handle the “end of file” condition (`Ctrl+D`)

You don’t have to:

- use the `PATH`
- implement built-ins
- handle special characters : `"`, `'`, `` ` ``, `\`, `*`, `&`, `#`
- be able to move the cursor
- handle commands with arguments

execve will be the core part of your Shell, don't forget to pass the environ to it...

(/)

```
julien@ubuntu:~/shell$ ./shell
#cisfun$ ls
./shell: No such file or directory
#cisfun$ /bin/ls
barbie_j       env-main.c  exec.c  fork.c  pid.c  ppid.c    prompt   prompt.c  shel
l.c  stat.c         wait
env-environ.c  exec    fork    mypid   ppid   printenv  promptc  shell     stat test
_scripting.sh  wait.c
#cisfun$ /bin/ls -l
./shell: No such file or directory
#cisfun$ ^[[D^[[D^[[D
./shell: No such file or directory
#cisfun$ ^[[C^[[C^[[C^[[C
./shell: No such file or directory
#cisfun$ exit
./shell: No such file or directory
#cisfun$ ^C
julien@ubuntu:~/shell$ echo "/bin/ls" | ./shell
barbie_j       env-main.c  exec.c  fork.c  pid.c  ppid.c    prompt   prompt.c  shel
l.c  stat.c         wait
env-environ.c  exec    fork    mypid   ppid   printenv  promptc  shell     stat test
_scripting.sh  wait.c
#cisfun$ julien@ubuntu:~/shell$
```

Repo:

- GitHub repository: `simple_shell`

Done? | Help | >_ Get a sandbox

2. Simple shell 0.2

mandatory

Simple shell 0.1 +

- Handle command lines with arguments

Repo:

- GitHub repository: `simple_shell`

Done? | Help | >_ Get a sandbox

3. Simple shell 0.3

mandatory

Simple shell 0.2 +

- Handle the `PATH`
- `fork` must not be called if the command doesn't exist

```
julien@ubuntu:~/shell$ ./shell_0.3
:) /bin/ls
barbie_j       env-main.c  exec.c  fork.c  pid.c  ppid.c    prompt   prompt.c  shell
_0.3  stat    test_scripting.sh  wait.c
env-environ.c  exec    fork    mypid   ppid   printenv  promptc  shell     shell.c
stat.c  wait
:) ls
barbie_j       env-main.c  exec.c  fork.c  pid.c  ppid.c    prompt   prompt.c  shell
_0.3  stat    test_scripting.sh  wait.c
env-environ.c  exec    fork    mypid   ppid   printenv  promptc  shell     shell.c
stat.c  wait
:) ls -l /tmp
total 20
-rw------- 1 julien julien    0 Dec  5 12:09 config-err-aAMZrR
drwx------ 3 root   root   4096 Dec  5 12:09 systemd-private-062a0eca7f2a44349733e78
cb4abdff4-colord.service-V7DUzr
drwx------ 3 root   root   4096 Dec  5 12:09 systemd-private-062a0eca7f2a44349733e78
cb4abdff4-rtkit-daemon.service-ANGvoV
drwx------ 3 root   root   4096 Dec  5 12:07 systemd-private-062a0eca7f2a44349733e78
cb4abdff4-systemd-timesyncd.service-CdXUtH
-rw-rw-r-- 1 julien julien    0 Dec  5 12:09 unity_support_test.0
:) ^C
julien@ubuntu:~/shell$
```

Repo:

- GitHub repository: `simple_shell`

Done? | Help | >_ Get a sandbox

4. Simple shell 0.4

mandatory

Simple shell 0.3 +

- Implement the `exit` built-in, that exits the shell
- Usage: `exit`
- You don't have to handle any argument to the built-in `exit`

Repo:

- GitHub repository: `simple_shell`

(/)

☐ Done? | Help | >_ Get a sandbox

5. Simple shell 1.0

mandatory

Simple shell 0.4 +

- Implement the `env` **built-in**, that prints the current environment

```
julien@ubuntu:~/shell$ ./simple_shell
$ env
USER=julien
LANGUAGE=en_US
SESSION=ubuntu
COMPIZ_CONFIG_PROFILE=ubuntu
SHLVL=1
HOME=/home/julien
C_IS=Fun_:)
DESKTOP_SESSION=ubuntu
LOGNAME=julien
TERM=xterm-256color
PATH=/home/julien/bin:/home/julien/.local/bin:/usr/local/sbin:/usr/local/bin:/usr/sb
in:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
DISPLAY=:0
$ exit
julien@ubuntu:~/shell$
```

Repo:

- GitHub repository: `simple_shell`

☐ Done? | Help | >_ Get a sandbox

Done with the mandatory tasks? Unlock 11 advanced tasks now!