EEE3535-01 **Fall 2020**

Assignment 1: Shell

Due: Monday, Sept. 28, 2020, 11:59PM

1 Introduction

- The objective of this assignment is to implement a simple shell program that we will call it "Yonsei shell" or simply **ysh**. This assignment is independent of xv6-riscv.
- A shell in Linux or Unix [1] is a program that provides users with interfaces to the OS. It receives inputs from users and executes the given commands.
- When you first log into Linux (e.g., Ubuntu) or Unix (e.g., Mac) via terminal, you should notice something similar to the following tailed by a "$" sign waiting for your input. In Mac OS, it may prompt a "%" sign instead of $.

```
username@ubuntu $
```

- Try typing an `ls` command in the terminal. This command displays a list of files and directories in the `xv6-riscv/` directory that you should have installed during the Assignment #0.

```
$ ls xv6-riscv/
LICENSE     Makefile     README     kernel     mkfs     tar.sh     user
```

- Try another command, `grep`, in the terminal as follows. This command finds lines containing a keyword `riscv` in the `xv6-riscv/README` text file and displays the search results.

```
$ grep risc xv6-riscv/README
https://github.com/riscv/riscv-gnu-toolchain, and qemu compiled for
riscv64-softmmu. Once they are installed, and in your shell
```

- Both `ls` and `grep` are programs installed by default in Linux or Unix, and you can find their locations using a `which` command. You may guess that `/usr/bin/` directory is something analogous to `C:\Program Files\` directory in Windows, where program executables (i.e., `*.exe` files) are located at.

```
$ which ls
/usr/bin/ls

$ which grep
/usr/bin/grep
```

- You learned above that shell is a program. You provided this program with a text input such as `ls xv6-riscv/`. Then, the shell executed the `ls` program with the text string of `xv6-riscv/` as its input. In short, the shell program created and executed another process. This can be easily done by using a combination of `fork()` and `execvp()` system calls.
- When the user input of `ls xv6-riscv/` was given, the shell should have done something similar to the following. It creates a child process via a `fork()` system call and makes the child process execute the given `ls` program using an `execvp()` system call.

```
int pid = fork();
if(pid == 0) {
    char *argv[3] = {"ls", "xv6-riscv/", 0};
    execvp(argv[0], argv);
}
```

- The shell repeats calling `fork()` and `execvp()` indefinitely in a `while` loop until an `exit` command is put to terminate the shell. The `exit` command may also close the current terminal window, but it depends on how your terminal is configured to terminate.

```
$ exit
```

- There are multiple shell programs available in Linux or Unix. Common shell programs are bash, csh, ksh, tsh, zsh, etc. Typing the following command will show you what kind of shell you are in. In Ubuntu, the default shell is bash, and Mac OS uses zsh; old versions of Mac OS may use bash as well.

```
$ echo $SHELL
/bin/bash
```

- Since the shell itself is a program, can you make the shell execute a shell again? The following example makes the current shell run another bash shell. Obviously, it works.

```
$ bash
$
```

2 Implementation

- Download a C file named `ysh.c` uploaded on YSCEC along with this pdf file. The C file provides you with a skeleton code (i.e., basic code structure) to begin with.
- Compile the downloaded C source code using a `gcc` compiler as follows. The `-o ysh` directive tells the compiler that the compiled program will be named as `ysh`, and `ysh.c` is the C code it needs to compile. Do not use C++ compilers such as `g++` to compile the code. `-Wall` and `-Werror` are optional flags to tell the compiler to check the code rigorously for all possible errors and warnings. If the compilation is successful, you can execute the generated program by typing `./ysh`.

```
$ gcc -Wall -Werror -o ysh ysh.c
$ ./ysh
EEE3535 Operating Systems: starting ysh
$ exit
$
```

- The only commands that work in the skeleton code are `cd` and `exit` commands. These commands are not separate programs but built-in commands in the shell performed by the parent process (i.e., ysh itself instead of forking a child process). You have to implement the rest of code to make ysh handle other command inputs.
- There are numerous features to implement a complete shell program, but in this assignment we will work on only a few basic features. As far as you can make the following commands work correctly, your assignment will be done.

  ① A single command: The ysh shell should be able to execute a command with arbitrary number of inputs. The maximum number of inputs that the command can get is eight (i.e., `#define max_args 8`). This setting should be enough to cover the following examples.

```
$ date
Tue 08 Sep 2020 05:35:35 PM KST
```

```
$ ls ./
xv6-risv    ysh.c
```

```
$ grep -n while ysh.c
27:    while(*cmd == ' ') { cmd++; }
33:    // exit command: return -1 to terminate the while loop in main.
48:    while((fd = open("console", O_RDWR)) >= 0) {
55:    while((cmd = readcmd(buf))) {
```

  The `date` command displays the system date and time, and `ls ./` shows a list of files and directories in the current folder. `grep -n while ysh.c` searches the `ysh.c` file and displays all lines containing the keyword of `while` with their line numbers in the file. There are four such lines in the `ysh.c` file at line #27, 33, 48, and 55.

  ② Piped commands: Connect two commands via a pipe (i.e., `|` symbol) such that the command on the left of pipe passes its output to the one on the right side.

```
$ ps | grep ysh
120837 pts/0    00:00:00 ysh
```

```
$ nslookup www.yonsei.ac.kr | awk /yonsei/,0
Name:        www.yonsei.ac.kr
Address: 165.132.13.38
```

The `ps` command gives a list of currently running processes, and the piped `grep ysh` command finds which lines in the `ps` output include the keyword of `ysh`. `nslookup www.yonsei.ac.kr` returns the DNS query result of `www.yonsei.ac.kr` domain name, and `awk /yonsei/,0` finds a line containing the keyword of `yonsei` and displays all the lines of `nslookup` output from the found line to the end.

③ Serialized commands: If two commands are concatenated by a `;` sign, the shell executes the first command (on the left of semicolon) and then executes the second command after the first one is done. These two commands make no interactions with other. This option lets you type multiple commands in one line separated by semicolons. You can further extend the option to concatenate more commands not just two.

```
$ cd /; ls -l
total 1392348
lrwxrwxrwx   1 root root     7 Sep 28 23:59 bin -> usr/bin
drwxr-xr-x   4 root root  4096 Sep 28 23:59 boot
drwxrwxr-x   2 root root  4096 Sep 28 23:59 cdrom
drwxr-xr-x  18 root root  4420 Sep 28 23:59 dev
drwxr-xr-x 129 root root 12288 Sep 28 23:59 etc
drwxr-xr-x   4 root root  4096 Sep 28 23:59 home

...

$ echo EEE3535; echo Operating; echo Systems
EEE3535
Operating
Systems
```

The `cd /` command changes directory to the root of file system (e.g., `C:\` in Windows), and `ls -l` displays the detailed information of files and folders in the root directory. Since the `cd` command is performed by the parent process in the shell instead of forking a child process, the subsequent `ls` is affected by the preceding `cd` command. A series of `echo` commands in the next example are simply invoked one by one.

④ A series of piped commands: Similarly, you should be able to pipe a series of commands not just two.

```
$ ls /usr/bin | grep zip | wc -l
17
```

The example above counts how many programs in the `/usr/bin/` directory contain `zip` in their names. The result tells us that there total 17 such programs.

⑤ Invalid commands: Not all shell inputs are valid ones. The input commands may refer to non-existing programs in a machine, or they may be mistakenly typed in with some typos. The shell should be able to handle such errors.

```
$ ext
Command not found: ext

$ ls | grp xv6
Command not found: grp

$ ech Yonsei; echo University
Command not found: ech
University
```

⑥ Exit command: An `exit` command should cleanly break the `while` loop in `main()`. This is already implemented in the ysh skeleton code, and your implementation should not mess it up.

```
$ exit
$
```

3 Submission

- After the implementation of ysh is done, upload your `ysh.c` file on YSCEC. This is the only file you have to submit. Do not change the `ysh.c` file name by adding your student ID, name, project1, etc.

4 Grading Rules

- The following is a general guideline for grading the assignment. 30-point scale will be used for this assignment. The minimum score is zero, and negative scores will not be given. Grading rules are subject to change, and a grader may add a few extra rules for fair evaluation of students' efforts.

  -3 points: The `ysh.c` file is renamed and includes some tags such as a student ID.

  -5 points: A program code does not have sufficient amount of comments. Comments in the skeleton code do not count. You must make an effort to clearly explain what each part of the code intends to do.

  -5 points per feature: There are six numbered features listed in Section 2 from ① A single command to ⑥ Exit command. Failing to reproduce the examples will lose 5 points per numbered feature. A partly working feature will not earn any partial credits. If ysh fails to cleanly repeat the `while` loop after a command run, the tested feature is regarded as incomplete.

  -30 points: No or late submission. The shell is hard-coded to support only the described examples in Section 2, or the submitted code makes little efforts to complete the assignment. If `ysh.c` does not compile with `gcc` compiler with `-Wall` and `-Werror` flags, it is also regarded as no submission.

  F grade: A submitted code is copied from someone else. All students involved in the incident will be penalized and given F for final grades irrespective of assignments, attendance, etc.

- Your teaching assistant (TA) will grade your assignments. If you think your assignment score is incorrect for any reasons, discuss your concerns with the TA. Always be courteous when contacting the TA. In case no agreement is made between you and the TA, elevate the case to the instructor to review your assignment. Refer to the course website for the contact information of TA and instructor: `https://icsl.yonsei.ac.kr/eee3535`

- Begging for partial credits for no viable reasons will be treated as a cheating attempt, and thus such a student will lose all scores for the assignment.

References

[1] Phil Estes, "Linux vs. Unix: What's the Difference?" [Online] Available: `https://opensource.com/article/18/5/differences-between-linux-and-unix`