EEE3535-01 **Fall 2023**

Assignment 1: Shell

Due: Wednesday, Oct. 4, 2023, 11:59PM

1 Introduction

- The objective of this assignment is to implement a simple shell in xv6-riscv.
- A shell in Linux is a user program that reads command inputs and executes them. If you open a terminal, you should see something similar to the following trailed by a "`$`" sign waiting for your input.

```
username@ubuntu $
```

- Try an `ls` command to display a list of files in the `/usr/bin/` directory.

```
username@ubuntu $ ls /usr/bin
alias       awk         bg          bison       bzip2           c++
cc          cpp         curl        env         find            g++
gcc         git         grep        ...
```

- For the `ls /usr/bin/` command input, the shell creates a child process using `fork()` and makes the child execute the command via `exec()`.

```
int pid = fork();
if(pid > 0) {
    // The parent process (i.e., shell) waits for the child to finish.
    wait(0);
}
else if(pid == 0) {
    // The child process executes the command.
    char *argv[3] = {"ls", "/usr/bin/", 0};
    exec(argv[0], argv);
}
```

- The shell repeats such execution for every user command in a loop until `exit` is entered.

```
username@ubuntu $ exit
```

- There is not only one shell program in Linux but many. Common shells are `bash`, `csh`, `ksh`, `tsh`, `zsh`, etc. You can check the current shell type as follows.

```
username@ubuntu $ echo $SHELL
/bin/bash
```

- Since the shell is a user program, you can make it run another shell as a child process, such as the following.

```
username@ubuntu $ bash
bash $
```

2 Implementation

- To start the assignment, go to `xv6-riscv/`, download `shell.sh`, and run the script to update xv6-riscv.

```
username@ubuntu $ cd xv6-riscv/
username@ubuntu $ wget https://icsl.yonsei.ac.kr/wp-content/uploads/shell.sh
username@ubuntu $ chmod +x shell.sh
username@ubuntu $ ./shell.sh
```

- If the update is successful, you can run a new shell program called `ysh`. The only working commands in the `ysh` skeleton code are `cd` and `exit`, which do not invoke programs but are reserved keywords in the shell.

```
username@ubuntu $ make qemu

...

qemu-system-riscv64 -machine virt -bios none -kernel kernel/kernel -m 128M -smp 1
-nographic -global virtio-mmio.force-legacy=false -drive file=fs.img,if=none,
format=raw,id=x0 -device virtio-blk-device,drive=x0,bus=virtio-mmio-bus.0

EEE3535 Operating Systems: booting xv6-riscv kernel
EEE3535 Operating Systems: starting sh
$ ysh
EEE3535 Operating Systems: starting ysh
$ exit
EEE3535 Operating Systems: closing ysh
$
```

- The following shows a part of the `ysh` skeleton code in `user/ysh.c`.

```
/* user/ysh.c */

...

int main(int argc, char **argv) {

    ...

    // Read and run input commands.
    while((cmd = readcmd(buf)) && !runcmd(cmd)) ;

    ...
}

// Run a command.
int runcmd(char *cmd) {
    if(!*cmd) { return 0; }                    // Empty command

    // Skip leading white space(s).
    while(*cmd == ' ') { cmd++; }
    // Remove trailing white space(s).
    for(char *c = cmd+strlen(cmd)-1; *c == ' '; c--) { *c = 0; }

    if(!strcmp(cmd, "exit")) { return 1; }      // exit command
    else if(!strncmp(cmd, "cd ", 3)) {          // cd command
        if(chdir(cmd+3) < 0) { fprintf(2, "Cannot cd %s\n", cmd+3); }
    }
    else {
        // EEE3535 Operating Systems
        // Assignment 1: Shell
    }
    return 0;
}
```

- The `main()` gets a user command from `readcmd()` and executes it through `runcmd()`. The `readcmd()` function is complete in the skeleton code. Do not modify `readcmd()` and `main()` functions.

- `runcmd()` first checks if `cmd` is an empty string. Then, it removes leading and trailing white spaces for a non-empty string.

- If the command is `exit`, it returns 1, which will break the while loop in `main()` and end the `ysh` program. If the first three characters of `cmd` are `"cd "`, it changes to the specified directory.
- Otherwise, the command is supposed to execute a program. This part of the function is left for the assignment.
- `runcmd()` returns 0 for no errors or a non-zero value for an exit condition (e.g., fork failure).
- You are allowed to add functions or structs to `ysh.c` or introduce additional variables in `runcmd()`. But do not modify `readcmd()` and `main()` functions.

3 Validation

- Every shell has a different parsing scheme for command inputs, so they work differently when complex commands are put together. But simple commands run the same because there cannot be different interpretations.
- This assignment will test if your `ysh` implementation can handle simple commands with pipe (`|`), series (`;`), background (`&`), and/or output redirect (`>`) symbols.

1. **A single command**: The shell forks a child and makes it run a program. The maximum number of args (i.e., length of `argv[]`) that a command can have is 16 in xv6-riscv (i.e., `#define max_args 16`).

```
$ ysh
EEE3535 Operating Systems: starting ysh

$ forktest
fork test
fork test OK

$ grep RISC README
but is implemented for a modern RISC-V multiprocessor using ANSI C.
You will need a RISC-V "newlib" tool chain from

$ ls cat echo init kill
cat            2 3 33656
echo           2 4 32536
init           2 7 33000
kill           2 8 32448
```

2. **Two commands with a pipe**: The command on the left side of a pipe forwards its output to the input of the right-side command. There can be two or more child processes depending on the implementation.

```
$ whoami | grep ID
Student ID:   2023143535

$ grep xv6 README | wc
5 54 319
```

3. **Two commands in a series**: The command on the right of a semicolon is executed after the left-side command is done using `wait()`.

```
$ stressfs; wc stressfs0
stressfs starting
write 0
write 1
write 2
write 3
write 4
read
read
read
read
read
0 1 10240 stressfs0
```

4. **A background command**: An `&` sign at the end of a command makes the shell execute the program in the background. The `grind` program slowly prints `BABABABA` $\cdots$, which never ends. While `grind` runs in the background, the shell can process another command.

```
$ grind &
$ forktest
fork test
fork test OK
$ BABABABABABABABABABABABABABABABABABABABA ... // grind never ends.
```

5. **Redirected output**: The command output on the left side of a > sign is redirected to a file. The file should be opened with `O_WRONLY|O_CREATE|O_TRUNC` flags.

```
$ cat README > README2
$ cat README2
v6 is a re-implementation of Dennis Ritchie's and Ken Thompson's Unix
Version 6 (v6).  xv6 loosely follows the structure and style of v6,

...
```

6. **Everything all at once**: Can you handle this?

```
$ ysh
EEE3535 Operating Systems: starting ysh
$ usertests > usertests.out &; stressfs | grep read | wc; cat usertests.out; exit
5 5 25
usertests starting
test copyin: OK
test copyout: OK
test copyinstr1: OK
test copyinstr2: OK
test copyinstr3: OK
test rwsbrk: OK
EEE3535 Operating Systems: closing ysh
$
```

4 Submission

- In the `xv6-riscv/` directory, execute the `tar.sh` script, which will create a tar file named after your student ID (e.g., `2023143535`).

```
$ ./tar.sh
$ ls
2023143535.tar  kernel  LICENSE  Makefile  mkfs  README  tar.sh  user
```

- Upload the tar file (e.g., `2023143535.tar`) on LearnUs. Do not rename the file.

5 Grading Rules

- The following is the general guideline for grading. A 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; a grader may add a few extra rules without notice for a fair evaluation of students' efforts.

  -5 points: The submitted tar file includes redundant tags such as a student name, `hw1`, etc.

  -5 points: The code has insufficient comments. Comments in the skeleton code do not count. You must clearly explain what each part of your code does.

  -5 points each: The validation section shows six test cases. Each failed test will lose 5 points.

  -30 points: No or late submission.

  Final grade = F: The submitted tar file is copied from someone else. All students involved in the incidents will get Fs for the final grade.

- Your teaching assistant (TA) will grade your assignments. If you think your assignment score is incorrect, discuss your concerns with the TA. Always be courteous when contacting the TA. If no agreements are 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 the TA and instructor: `https://icsl.yonsei.ac.kr/eee3535`
- Arguing for partial credits for no valid reasons will be regarded as a cheating attempt; such a student will lose the assignment scores.