DEPARTMENT OF COMPUTER SCIENCE, HANYANG UNIVERSITY

# Simple User-level Unix Shell

**Operating Systems: Three Easy Pieces**

Programming Project with Linux

Due date: March 28, 2019

## 1 UNIX SHELL

Shell is a user interface for access to an operating system's services.Shell receives User's commands and passes on to the Kernel and get responses.

## 2 PROJECT SPECIFICATION

- This project is for the Linux, not xv6.
- In this programming assignment, you will implement a simple user-level command line interpreter. (also called shell)
- Basically, shell operates in this way
  1. User types command to the prompt.
  2. Shell creates a child process.
  3. Child process execute the command.

```
$ ./shell
prompt> ls -al
(output of "ls -al" is shown here ...)
prompt>
```

- Shell can be run in two ways : **Interactive mode** & **Batch mode**
- In interactive mode, shell displays a prompt and user types in a command at the prompt. (see above example)
- Each line may contain multiple commands separated with the ; character.
- Each command separated by ; character should be run simultaneously. It means that multiple processes run their command concurrently and parent process should wait all children before printing the next prompt. (As result, the output of this batched commands can be shown intermixed)
  - (ex) This is a valid command

```
prompt>  ls -al ; cat file ; pwd
```

- In batch mode, shell is started by specifying a batch file on its command line. The batch file contains the list of commands that should be executed.
- In batch mode, you don't need to display a prompt, but should echo each line you read from the batch file back to the user before executing it.
- In both interactive and batch mode, shell stops accepting new commands when it sees the **quit** command or reaches the end of the input stream (i.e.,the end of file or user types **Ctrl+D**)

## 3 EXAMPLES

- Interative mode

```
$ ./shell
prompt>
prompt> ls
(ls output is shown ...)
prompt> ls -al ; cat file
(outputs of two commands can be shown intermixed)
prompt> quit
$
```

- Batch mode

```
$ vi [batchfile]
ls
/bin/ls -al
ls ; pwd ; cat file
```

```
$ ./shell [batchfile]
ls
(output of "ls")
/bin/ls -al
(output of "/bin/ls -al")
ls ; pwd ; cat file
(outputs of "ls", "pwd", "cat file".
this can be shown intermixed)
$
```

# 4 Useful Hints

- You should be familiar with **fork()**, **execvp()**, and **wait()**/**waitpid()** system call to implement this project. See the man page.
  - In **execvp()**, the list of arguments must be terminated with a NULL pointer;
- When reading/parsing the command from input string, you may want to look at **fgets()**, **strtok()** function.

# 5 Submission

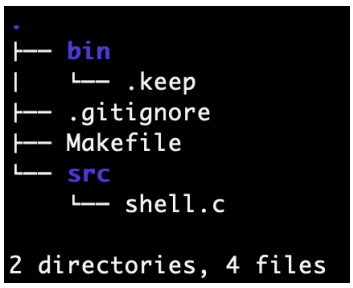
- Source code should be compiled with Makefile.(Source code that can't be compiled with make is not graded.)
- You should upload your code to hanyang gitlab repository. (note that email submission is not accepted!!)
- Every documentation should be written in your Gitlab Wiki.
  - Wiki: documentation and explanation storage. For example, analysis report, design documents, background information, dev note, etc. can be here.
- Follow the appropriate coding convention.
  - In your own source code, follow the HYU style.
- Provide a detailed description of the code.

## 5.1 PREPARING SUBMISSION

1. Make a new directory named "proj_shell" in ~/os2018 directory, which is already managed by your Git.

```
$ cd ~/os2019
$ mkdir proj_shell
```

2. Your project should follow general C/C++ project structure. You can download a sample project structure from piazza. (Project without the following structure is not graded.)
   - Your project have 2 folders
     - src: consists of source files (ex) .c files
     - bin: consists of output files of compiler (ex) execute files
   - Your project must have 4 files (before build)
     - shell.c : your source file. At **src** folder.
     - .gitignore : gitignore for binary files. At **proj_shell** folder.
     - Makefile : At **proj_shell** folder.
     - .keep : empty file to keep an empty directory. At **bin** folder.




   - After build, your project must have a executable file.
     - shell: executable file generated by make. At **bin** folder.

3. Build your project to confirm that it works well before submission

```
daktopia@daktopia-System-Product-Name:~/os2019/proj_shell$ tree -a
.
├── bin
|   └── .keep
├── .gitignore
├── Makefile
└── src
    └── shell.c

2 directories, 4 files
daktopia@daktopia-System-Product-Name:~/os2019/proj_shell$ make
gcc -g -Wall -I./include/   -c -o src/shell.o src/shell.c
gcc -g -Wall -I./include/ -o ./bin/shell src/shell.o -L./lib/ -lpthread
daktopia@daktopia-System-Product-Name:~/os2019/proj_shell$ ./bin/shell
prompt> quit
daktopia@daktopia-System-Product-Name:~/os2019/proj_shell$ make clean
rm -f ./bin/shell src/shell.o
```

## 5.2 Preparing Submission

1. Add your new project to the Git (now files are tracked by Git)

```
$ git add .
```

2. Commit files to the local repository

```
$ git commit -m "first commit of proj_shell"
```

3. Push local repository contents to the remote repository (Hanyang GitLab)

```
$ git push origin master
```

## 5.3 Policy on Academic Integrity

1. Do brainstorming with your class mates via on/offline.
   - Piazza can be a good dev community. Share what you think with your friends.
2. Do NOT share or copy the code.
   - Please keep the programming ethics. Your code will be kept in GitLab. Take responsibility for your code.

# 6 EVALUATION

1. The code must be compiled with a Makefile.
2. You should write specific commentary for your code.
3. You should write defensive source code, your shell should print out error case thoroughly.
   - for example, these are valid commands

```
$ ./shell
prompt>
prompt>   ls
prompt>  ls ; pwd
prompt> ls;; pwd
```

4. The last version(commit and push) before deadline will be pulled and evaluated. Any updates after the deadline are invisible to the test program.