Network Programming Project 1 - NPShell

Deadline: Tuesday, 2020/10/20 18:20

I. Introduction

In this project, you are asked to design a shell with special piping mechanisms.

II. Scenario of using npshell

A. Some important settings

1. The structure of the working directory

```
working_dir
├─ bin                # The directory contains executables
│   ├─ cat
│   ├─ ls
│   ├─ noop           # A program that does nothing
│   ├─ number         # Add a number to each line of input
│   ├─ removetag      # Remove HTML tags and output to STDOUT
│   └─ removetag0     # Same as removetag, but outputs error
└─ test.html            messages to STDERR.
```

2. In addition to the above executables, the following are built-in commands supported by your npshell
   a) setenv
   b) printenv
   c) exit

B. Scenario

```
bash$ ./npshell     # execute your npshell
% printenv PATH     # initial PATH is bin/ and ./
bin:.
% setenv PATH bin   # set PATH to bin/ only
% printenv PATH
bin
% ls
bin             test.html
% ls bin
cat   ls   noop   number   removetag   removetag0
% cat test.html > test1.txt
% cat test1.txt
<!test.html>
<TITLE>Test</TITLE>
<BODY>This is a <b>test</b> program
for ras.
</BODY>
```

```
% removetag test.html

Test
This is a test program
for ras.

% removetag test.html > test2.txt
% cat test2.txt

Test
This is a test program
for ras.

% removetag0 test.html
Error: illegal tag "!test.html"

Test
This is a test program
for ras.

% removetag0 test.html > test2.txt
Error: illegal tag "!test.html"
% cat test2.txt

Test
This is a test program
for ras.

% removetag test.html | number
   1
   2 Test
   3 This is a test program
   4 for ras.
   5
% removetag test.html |1  # this pipe will pipe STDOUT to next command
% number                 # the command's STDIN is from previous pipe
   1
   2 Test
   3 This is a test program
   4 for ras.
   5
% removetag test.html |2  # |2 will skip 1 line of commands and then
% ls                      # pipe STDOUT to the next next command
bin             test1.txt
test.html       test2.txt
```

```
% number                    # the command's STDIN is from the
   1                        # previous pipe (removetag)
   2 Test
   3 This is a test program
   4 for ras.
   5
% removetag test.html |2  # pipe STDOUT to the next next command
% removetag test.html |1  # pipe STDOUT to the next command (merge
                          # with the previous one)
% number                  # STDIN is from the previous pipe
   1
   2 Test
   3 This is a test program
   4 for ras.
   5
   6
   7 Test
   8 This is a test program
   9 for ras.
  10
% removetag test.html |2
% removetag test.html |1
% number |1
% number
   1    1
   2    2 Test
   3    3 This is a test program
   4    4 for ras.
   5    5
   6    6
   7    7 Test
   8    8 This is a test program
   9    9 for ras.
  10   10
% removetag test.html | number |1
% number
   1    1
   2    2 Test
   3    3 This is a test program
   4    4 for ras.
   5    5
% ls |2
% ls
bin             test1.txt
test.html       test2.txt
```

```
% number > test3.txt
% cat test3.txt
   1 bin
   2 test.html
   3 test1.txt
   4 test2.txt
% removetag0 test.html |1
Error: illegal tag "!test.html" # output error message to STDERR
% number
   1
   2 Test
   3 This is a test program
   4 for ras.
   5
% removetag0 test.html !1 # this pipe will pipe both STDOUT and STDERR
                           # to the next command
% number
   1 Error: illegal tag "!test.html"
   2
   3 Test
   4 This is a test program
   5 for ras.
   6
% date
Unknown command: [date].
# TA manually move the executable `date` into ${working_dir}/bin/
% date
Mon Oct  5 15:12:35 CST 2020
% exit
bash$
```

III. Requirements and Hints

A. In this project, the commands `noop`, `number`, `removetag`, `removetag0` are offered by TA. Please download them from E3, compile them and put these executables into the folder ${working_dir}/bin/.
e.g., `g++ noop.cpp -o ${working_dir}/bin/noop`

B. `ls` and `cat` are usually placed in the folder /bin/ in UNIX-like systems. Please copy them into the folder ${working_dir}/bin/
e.g., `cp /bin/ls /bin/cat working_dir/bin/`

C. During demo, TA will copy additional commands to bin/, which is under your working directory. Your npshell program should be able to execute them.

D. You must use exec-based functions to run commands, except for built-in commands (`setenv`, `printenv` and `exit`).
You must not use functions like system() or some other functions to do the job.

E. When you implement output redirection (>) to a file, if the file already exists, the file should be overwritten. (not append)

F. You don't have to worry about outputting to both file and pipe for the same command.
```
% ls > test.txt | cat # this will not appear
```

G. You don't have to implement input redirection from a file (<)

H. You can only implement the npshell with C and C++, other third-party libraries are **NOT allowed**.

IV. Specification

A. Input

1. The length of a single-line input will not exceed 15000 characters.
2. Each command will not exceed 256 characters.
3. There must be one or more spaces between commands and symbols (or arguments), but no spaces between pipe and numbers.

```
% cat hello.txt | number
% cat hello.txt |4
% cat hello.txt !4
```

4. There won't exist any `'/'` character in test cases.

B. NPShell Behavior

1. Use `"% "` as the command line prompt. Notice that there is **one space character** after %.
2. The npshell terminates after receiving the **`exit`** command or **EOF**.
3. Notice that you must handle the forked processes properly, or there might be zombie processes.
4. Built-in commands (`setenv`, `printenv`, `exit`) will appear solely in a line. No command will be piped together with built-in commands.

C. `setenv` and `printenv`

1. The initial environment variable PATH should be set to bin/ and ./ by default.

```
% printenv PATH
bin:.
```

2. setenv usage: `setenv [variable name] [value to assign]`
3. printenv usage: `printenv [variable name]`

```
% printenv QQ # Show nothing if the variable does not exist.
% printenv LANG
en_US.UTF-8
```

4. The number of arguments for setenv and printenv will be correct in all test cases.

D. Numbered-Pipes and Ordinary Pipe

1. `|N` means the **STDOUT** of the left hand side command should be piped to the first command of the next N-th line, where $1 \leq N \leq 1000$.
2. `!N` means both **STDOUT** and **STDERR** of the left hand side command should be piped to the first command of the next N-th line, where $1 \leq N \leq 1000$.
3. `|` is an ordinary pipe, it means the STDOUT of the left hand side command will be piped to the right hand side command. It will only appear **between two commands**, not at the beginning or at the end of the line.
4. The command number still counts for unknown commands.

```
% ls |2
% ctt
Unknown command: [ctt].
% number
1 bin/
2 test.html
```

5. setenv and printenv count as one command.

```
% ls |2
% printenv PATH
bin:.
% cat
bin
test.html
```

6. Empty line does not count.

```
% ls |1
%              # press Enter
% number
1 bin/
2 test.html
```

E. Unknown Command

1. If there is an unknown command, print error message as the following format: Unknown command: [command].

e.g.

```
% ctt
Unknown command: [ctt].
```

2. You don't have to print out the arguments.

```
% ctt -n
Unknown command: [ctt].
```

3. The commands after unknown commands will still be executed.

```
% ctt | ls
Unknown command: [ctt].
bin/    test.html
```

4. Messages piped to unknown commands will disappear.

```
% ls | ctt
Unknown command: [ctt].
```

F. Submission

1. Create a directory named as **your student ID**, put all files into the directory.
2. You **MUST** use **GNU Make** to build your project and compile your source code into **one executable** named **npshell**. The executable and Makefile should be placed at the **top layer of the directory**. We will use this executable for demo.
   You are NOT allowed to demo if we are unable to compile your project with **a single `make` command**.
3. Upload only your code and Makefile.
   Do NOT upload anything else (e.g. noop, removetag, test.html, **.git**, **__MACOSX**)
4. **zip** the directory and upload the .zip file to the E3 platform
   ATTENTION! We only accept .zip format
   e.g.
   Create a directory 0856053, the directory structure may be:

```
0856053
├── Makefile
├── shell.cpp
└── shell.h
```

   zip the folder 0856053 into 0856053.zip and upload 0856053.zip onto E3

5. Commit to BitBucket
   Create a **private** repository with name: *${Your_Student_ID}_np_project1* inside the **nctu_np_2020** team, and set the ownership to **nctu_np_2020**.

```
e.g. 0856053_np_project1
```

You can push anything you need onto bitbucket (including `removetag`, `noop`, `test.html`), but **please make sure to commit at least 5 times**.

G. We take plagiarism seriously.

All projects will be checked by a cutting-edge plagiarism detector.
You will get zero points on this project for plagiarism.
Please don't copy-paste any code from the internet, this may be considered plagiarism as well.
Protect your code from being stolen.

V. Notes

A. NP project should be run on NP servers (to be announced), otherwise, your account may be locked.
B. Any abuse of NP server will be recorded.
C. Don't leave any zombie processes in the system.
D. You will lose points for violating any of the rules mentioned in this spec.
E. Enjoy the project!