Network Programming Project 1 - NPShell

NP TA

Deadline: Sunday, 2021/10/24 23:55

1 Introduction

In this project, you are asked to design a shell named **npshell**. The npshell should support the following features:

1. Execution of commands
2. Ordinary Pipe
3. Numbered Pipe
4. File Redirection

More details will be defined in Section 3.

2 Scenario of using npshell

2.1 Structure of Working Directory

Example:

```
working_dir
|-- bin             # Executables may be added to or removed from bin/
|   |-- 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 messages to STDERR
|-- test.html
|-- npshell
```

2.2 Scenario

The following is a scenario of using the npshell.

```
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  npshell  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  # pipe STDOUT to the first command of next line
% number                  # STDIN is from the previous pipe (removetag)
   1
   2 Test
   3 This is a test program
   4 for ras.
   5
% removetag test.html |2  # pipe STDOUT to the first command of next next line
```

```
% ls
bin  npshell  test1.txt  test2.txt  test.html
% number                  # STDIN is from the previous pipe (removetag)
   1
   2 Test
   3 This is a test program
   4 for ras.
   5
% removetag test.html |2  # pipe STDOUT to the first command of next next line
% removetag test.html |1  # pipe STDOUT to the first command of next line
                          # (merge with the previous one)
% number                  # STDIN is from the previous pipe (both two removetag)
   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  npshell  test1.txt  test2.txt  test.html
% number > test3.txt
% cat test3.txt
   1 bin
   2 npshell
   3 test1.txt
   4 test2.txt
```

```
   5 test.html
% 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       # pipe both STDOUT and STDERR
                                 # to the first command of the next line
% 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 moves the executable "date" into $working_dir/bin/
% date
Mon Oct  4 15:12:35 CST 2021
% exit
bash$
```

3 Specification

3.1 NPShell Behavior

1. Use "% " as the command line prompt. Notice that there is one space character after %.
2. The npshell parses the inputs and executes commands.
3. The npshell terminates after receiving the **exit** command or **EOF**.
4. There will **NOT** exist the test case that commands need to read from STDIN.

3.2 Input

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

   Examples:

   ```
   % ls -l | cat
   % ls > hello.txt
   % cat hello.txt |4        # no space between "|" and "4"
   % cat hello.txt !4        # no space between "!" and "4"
   ```

4. Only **English alphabets (uppercase and lowercase)**, **digits**, **space**, **newline**, **"."**, **"-"**, **":"**, **">"**, **"|"**, and **"!"** may appear in test cases.

3.3 Built-in Commands

1. Format

   - **setenv [var] [value]**
     Change or add an environment variable.
     If **var** does not exist in the environment, add **var** to the environment with the value **value**.
     If **var** already exists in the environment, change the value of **var** to **value**.

     Examples:

     ```
     % setenv PATH bin         # set PATH to bin
     % setenv PATH bin:npbin   # set PATH to bin:npbin
     ```

   - **printenv [var]**
     Print the value of an environment variable.
     If **var** does not exist in the environment, show nothing.

     Examples:

     ```
     % printenv LANG
     en_US.UTF-8
     % printenv VAR1           # show nothing if the variable does not exist
     % setenv VAR1 test
     % printenv VAR1
     test
     ```

   - **exit**
     Terminate npshell.

2. Built-in commands will appear solely in a line.

3. Built-in commands will not pipe to other commands, and no commands will pipe to built-in commands.

3.4 Unknown Command

1. If there is an unknown command, print error message to **STDERR** with the following format: **Unknown command: [command].**

   Examples:

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

2. You do not need to print the arguments of unknown commands.

   Examples:

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

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

   Examples:

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

4. Messages piped to unknown commands will disappear.

   Examples:

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

3.5 Ordinary Pipe and Numbered Pipe

1. You need to implement **pipe** (**cmd1** | **cmd2**), which means the STDOUT of the left hand side command will be piped to the right hand side command.

   Examples:

```
% ls | cat   # The output of command "ls" acts as the input of command "cat"
bin
npshell
test.html
```

2. You need implement a special piping mechanism, called **numbered pipe**. There are two types of numbered pipe (**cmd** |**N** and **cmd** !**N**).

3. |**N** means the **STDOUT** of the left hand side command will be piped to **the first command of the next N-th line**, where $1 \leq N \leq 1000$.

4. **!N** means both **STDOUT** and **STDERR** of the left hand side command will be piped to **the first command of the next N-th line**, where $1 \leq N \leq 1000$.

5. |**N** and **!N** will only appear at the end of the line.

6. The line with build-in command or unknown command also counts as one line for numbered pipe, but the empty line does not.

   Examples:

```
% ls |2
% ctt
Unknown command: [ctt].
% cat       # The output of command "ls" acts as the input of command "cat"
bin
npshell
test.html
% ls |2
% setenv PATH bin
%           # Press Enter
% cat       # The output of command "ls" acts as the input of command "cat"
bin
npshell
test.html
```

7. When the output of one command is piped (no matter ordinary pipe or numbered pipe), it is likely that the output of this command exceeds the capacity of the pipe. In this case, you still need to guarantee that **all the output of this command is piped correctly**.

   Examples:

   ```
   % cat large_file.txt | number  # All output of "cat" should be piped correctly
   <many outputs...>
   % cat large_file.txt |1        # All output of "cat" should be piped correctly
   % number
   <many outputs...>
   ```

   (**Hints: You can think about when the npshell should wait the child process.**)

8. The number of piped commands may exceed the maximum number of processes that one user can run. In this case, you still need to guarantee that **all commands are executed properly**.

   Examples:

   ```
   # Suppose the process limit is 512 and there are 1000 "cat" in one line.
   # All commands in this line should be executed properly.
   % ls | cat | cat ..... | cat
   bin
   npshell
   test.html
   ```

9. There will **NOT** exist the test case that one pipe is full and needs to be read by the following process (not forked yet), but the process limit is reached.

3.6 File Redirection

1. You need to implement **standard output redirection** (**cmd > file**), which means the output of the command will be written to files.

   Examples:

   ```
   # The output of command "ls" is redirected to file "hello.txt"
   % ls > hello.txt
   % cat hello.txt
   bin
   npshell
   test.html
   ```

2. If the file already exists, the file should be overwritten (not append).
3. You do not need to handle appending for standard output redirection (>>).
4. You do not need to implement standard input redirection (<).
5. You do not need to implement standard error redirection (2>).
6. You do not need to handle outputting to multiple files for the same command.
7. You do not need to handle outputting to both the file and the pipe for the same command.

Examples:

```
% ls > f1.txt > f2.txt          # This will not happen
% ls > hello.txt | cat          # This will not happen
% ls > hello.txt |2             # This will not happen
% cat f1.txt | number > f2.txt  # This may happen
```

4 Requirements and Limitations

1. You can only use C/C++ to do this project. Other third-party libraries are **NOT** allowed.
2. In this project, **system()** function is **NOT allowed**.
3. Except for the three built-in commands (setenv, printenv, and exit), you **MUST** use **the exec family of functions** to execute commands.
4. You **MUST** create **unnamed pipes** to implement ordinary pipe and numbered pipe. Storing data into temporary files is **NOT allowed** for ordinary pipe and numbered pipe.
5. You should handle the forked processes properly, or there might be zombie processes.
6. You should set the environment variable **PATH** to **bin/** and **./** initially.

   Examples:

   ```
   bash$ ./npshell            # execute your npshell
   % printenv PATH            # initial PATH is bin/ and ./
   bin:.
   ```

7. You should **NOT** manage environment variables by yourself. Functions like **getenv()** and **setenv()** are **allowed**.
8. The commands **noop**, **number**, **removetag**, and **removetag0** are offered by TA. Please download them from E3 and compile.

   Examples:

   ```
   g++ noop.cpp -o $working_dir/bin/noop
   ```

9. The executables **ls** and **cat** are usually placed in the folder /bin/ in UNIX-like systems. You can copy them to your working directory.

   Examples:

   ```
   cp /bin/ls /bin/cat $working_dir/bin/
   ```

10. During demo, TA will prepare these commands (executables) for you, so you do **NOT** need to upload them. Besides, TA will copy additional commands to **bin/**, which is under your working directory. Your npshell program should be able to execute them.
11. We will use NP servers for demo. **Make sure your npshell can be executed in NP servers.**

5 Submission

- E3:
  (a) Create a directory named your **student ID**, put your source code files and Makefile into the directory.
  DO NOT put anything else in it (e.g., noop, removetag, test.html, .git, __MACOSX).
  (b) You must provide a **Makefile**, which compiles your source code into one executable named **npshell** with a single **make** command. The Makefile and the executable should be placed at **the top layer of the directory**. We will use this executable for demo.
  (c) zip the directory and upload the .zip file to E3.
  Attention!! we only accept .zip format

  ```
  Example:
  0856000
      |-- Makefile
      |-- npshell.cpp
      |...
  ```

  Zip the folder 0856000 into 0856000.zip and upload 0856000.zip to E3

- Bitbucket:
  (a) Create a **private** repository with name: **${Your_Student_ID}_np_project1** inside the workspace **nycu_np_2021** and the project **np_project1**.

  `e.g., 0856000_np_project1`

  (b) You can push anything to Bitbucket, but make sure to commit **at least 5 times**.

6 Notes

1. We take plagiarism seriously. **You will get zero points on this project for plagiarism.**
2. You will lose points for violating any of the rules mentioned in this spec.
3. NP projects should be run on NP servers. Otherwise, your account may be locked.
4. Any abuse of NP server will be recorded.
5. Do not leave any zombie processes in the system.