Project TA - Peter CHUNG (cspeter@cse.ust.hk): Handling questions before the deadline
Grader TA - Wang ZHE (zwangec@cse.ust.hk): Grading and appeal handling after the deadline

COMP3511 Operating System (Fall 2022)

Simplified Linux Shell (Redirection + Multi-level pipe)

Released on 30-Sep-2022 (Friday)   Due on 16-Oct-2022 (Sunday) at 23:59

Introduction

The aim of this project is to help students understand **process management**, **input and output redirection**, and **inter-process communication** in an operating system. Upon completion of the project, students should be able to implement a useful system program using related Linux system calls.

Program Usage

In this assignment, you need to implement a non-interactive command line interpreter (i.e., a simplified version of a standard Linux shell program) that supports redirection and multi-level pipes. The program name is `myshell`.

Suppose there are 4 files in the current working directory:

```
myshell myshell.c in.txt out.txt
```

Here is a sample usage of `myshell`:

```
$> ./myshell < in.txt > out.txt
```

`$>` represents the shell prompt.
`<` means input redirection, which is used to redirect the file content as the standard input
`>` means output redirection, which is used to redirect the standard output to a text file

Thus, you can easily use the given test cases to test your program

Given the content of `in.txt`, here is the content of `out.txt` after running the program:

Content of `in.txt`	**Content in `out.txt`**

`ls`	`myshell`
	`myshell.c`
	`in.txt`
	`out.txt`

If you simply would like to print the expected output on the screen, you can use:

```
$> ./myshell < in.txt
```

Getting Started

`myshell_skeleton.c` is a starting point. You don't need to start from scratch.

To get started, rename the file as `myshell.c`

Read carefully the documentation in the provided code. You are not required to start from scratch as the base file already provides you many useful features (e.g. command line parsing). Necessary programming concepts will also be introduced during the related lab(s).

Please note that C programming language (instead of C++) must be used to complete this assignment. C is not the same as C++. C99 option is added to allow a more flexible coding style. Here is the command to compile `myshell.c`

```
$> gcc -std=c99 -o myshell myshell.c
```

Restrictions

In this assignment, you **CANNOT** use `system` or `popen` function defined in the C Standard library. The purpose of the project assignment is to help students understand process management and inter-process communication. It is because these 2 functions can directly process the whole command (including pipe and redirection).

You should use the related Linux system calls such as `pipe` and `dup2`. When connecting pipes, POSIX file operations such as `read, open, write, close` should be used. You should not use `fread, fopen, fwrite, fclose` from the C standard library.

Assumptions

You can assume that the input format is valid.
There won't be commands with both redirection and pipe at the same time.
We assume that each command line has at most 256 characters (including NULL)
We assume that there exists at most 8 pipe segments.
Each pipe segment may have at most 8 arguments

- Note: `execvp` system call needs to store an extra `NULL` item to represent the end of the parameter list. Thus, you will find the constant is set to 9 (instead of 8) in the starter code. For details, please read the comment lines provided in the starter code

We assume that there exists at most 1 input redirection and at most 1 output redirection.
For output redirection, you can assume the output file does not exist in the current working directory. In other words, the grader will remove the temporary output text files (i.e., `tmp*.txt`).
You only need to handle 2 space characters: tab(`\t`) and space `( )`.

Feature 1: Redirection

Instead of typing the command on the console, the input can be redirected from a text file. The file input redirection feature can be completed by using the `dup/dup2` system calls (they are discussed in the lab). The key idea is to close the default stdin and replace the stdin with the file descriptor of an input file.

We can use the following command to count the number of lines of the file (`myshell.c`). Here is a sample input file redirection usage:

```
$> wc -l < myshell.c
```

Like the file input redirection, the output can also be redirected to a text file. The file output redirection feature can be completed by using the `dup/dup2` system calls. The key idea is to close the stdout and replace the stdout with the file descriptor of an output file.

We can use the following command to redirect the output of the `ls` command to an output text file (`tmp_out_only.txt`). Here is a sample output redirection usage:

```
$> ls -lh > tmp_out_only.txt
```

Please note that we have test cases with a mix of both input and output redirection. For example:

```
$> wc -l < myshell.c > tmp_in_then_out.txt

$> wc -l > tmp_out_then_in.txt < myshell.c
```

Feature 2: Multi-level pipe

In a shell program, a pipe symbol (|) is used to connect the output of the first command as the input of the second command. For example,

```
$> ls | sort
```

The `ls` command lists the contents of the current working directory. As the output of `ls` is already connected to `sort`, it won't print out the content to the screen. After the output of `ls` has been sorted by `sort` command, the sorted list of files appears on the screen.

In this project, you are required to support multiple-level pipes.

Example 1:

```
$> echo a1 a2 a3 a4 a5 a6 a7
```

The above command has 1 pipe segment
That segment has 8 arguments
This above example is useful to test the upper bound of the number of arguments

Example 2:

```
$> ls | sort -r | sort | sort -r | sort | sort -r | sort | sort -r
```

The above command has 8 segments.
Each segment has either 1 argument or 2 arguments.
The above example is useful to test the upper bound of the number of pipe segments

Example 3:

```
$> ls           -l -h
```

The input may contain several empty space characters.
The above example is useful to test whether you handle tabs and spaces correctly.

Sample Input Files

In total, 10 sample input files are provided:

- Case 1-3: Simple commands with different number of arguments
- Case 4-7: Redirection commands
    - For Case 4, 6 and 7, `myshell.c` is required in the same directory
    - For Case 5-7, output text files will be created after executing the commands
      These temporary text files can be easily removed by `rm tmp*.txt`
- Case 8-10: Pipe commands

Please note that the output content is dependent on the machine and the current working directory. So, it is impossible to provide the sample output files. To double-check, you can always run the sample Linux executable on the same machine and the same current working directory.

Sample Executable

The sample executable (runnable in a CS Lab 2 machine) is provided for reference. After the file is downloaded, you need to add an execution permission bit to the file. For example:

```
$> chmod u+x myshell
```

Project TA - Peter CHUNG (cspeter@cse.ust.hk): Handling questions before the deadline
Grader TA - Wang ZHE (zwangec@cse.ust.hk): Grading and appeal handling after the deadline

Development Environment

CS Lab 2 is the development environment. Please use one of the following machines (`csl2wk`**`XX`**`.cse.ust.hk`), where **XX**=`01`…`40`. The grader will use the same platform. In other words, *"my program works on my own laptop/desktop computer, but not in one of the CS Lab 2 machines*" is an invalid appeal reason. Please test your program on our development environment (not on your own desktop/laptop) thoughtfully, even you are running your own Linux OS. Remote login is supported on all CS Lab 2 machines.

Marking Scheme

1. (100%) Correctness of the **10** provided test cases**. We don't have other hidden test cases**, but we will check the source codes to avoid students hard coding the test cases in their programs.
   a. Example of hard coding: a student may simply detect the input (it is possible because all test cases are released) and then display the corresponding output without implementing the simplified shell program
2. Please test your program on our development environment (not on your own desktop/laptop) thoughtfully
3. Please fill in your name, ITSC email, and declare that you do not copy from others. A template is already provided near the top of the source file.
4. Automatically get 0 marks if `system` or `popen` function is used in your code

***Plagiarism**: Both parties (i.e., students providing the codes and students copying the codes) will receive 0 marks. Near the end of the semester, a plagiarism detection software (**JPlag**) will be used to identify cheating cases. **DON'T** do any cheating!*

Submission

File to submit:

`myshell.c`

Please check carefully you submit the correct file.
In the past semesters, some students submitted the executable file instead of the source file. Zero marks will be given as the grader cannot grade the executable file.
You are not required to submit other files, such as the input test cases.

Late Submission

For late submission, please submit it via email to the grader TA.
There is a 10% deduction, and only 1 day late is allowed (Reference: Chapter 1)