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

COMP3511 Operating System (Spring 2022)

Simplified Linux Shell (Multi-level Pipe)

Released on 28-Feb-2022 (Monday)   Due on 20-Mar-2022 (Sunday) at 23:59

Introduction

The aim of this project is to help students understand **process management** 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 that supports multi-level pipes. The program name is `mpipe`.

Suppose there are 4 files in the current working directory:

```
mpipe mpipe.c in.txt out.txt
```

Here is a sample usage of the non-interactive command line interpreter.

```
$> ./mpipe < 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

After running the program, the contents of `in.txt` and `out.txt` are as follows:

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

`ls`	`mpipe`
	`mpipe.c`
	`in.txt`
	`out.txt`

Getting Started

`mpipe_skeleton.c` is a starting point. To get started, rename the file as `mpipe.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 and run `mpipe.c`

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

Restrictions

In this assignment, you **CANNOT** use `system` 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 meaningless to directly use the system function to process the whole command.

You should use 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.

Multi-level Pipes

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.

Assumptions

You can assume that the input format is valid.

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

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

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 | sort | wc -l
```

The above command has 3 segments.
The first segment has one 2 arguments: `ls` and `-l`
the second segment has 1 argument: `sort`
The third segment has 2 arguments: `wc` and `-l`

Example 4:

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

The input may contain several empty space characters.
You only need to handle 2 space characters: tab(`\t`) and space( ).
The above example is useful to test whether you handle the tabs and spaces correctly.

Sample Input Files

In total, 10 sample input files are provided:

- Case 1-4: A single command with different number of arguments
- Case 5-6: A 2-level pipe
- Case 7-10: A multiple-level pipe

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. You can always run the sample Linux executable on the same machine and the same current working directory to double-check the expected output.

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 mpipe
```

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 before your submission, even you are running your own Linux OS. Remote login is supported on all CS Lab 2 machines, so you are not required to be physically present in CS Lab 2.

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. Example of hard coding: a student may simply detect the input and then display the corresponding output without implementing the program
2. Make sure to test your program in one of our CS Lab 2 machines (not your own desktop/laptop computer)
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.

***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 (MOSS) will be used to identify cheating cases. **DON'T** do any cheating!*

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

Submission

File to submit:

`mpipe.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 of the lecture notes)