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

COMP3511 Operating System (Fall 2024)

PA1: Simplified Interactive Linux Shell

Release on 22-Sep (Sun) Due on 12-Oct (Sat) at 23:59

Introduction

This project aims to enhance students' understanding of **process management**, **input and output redirection**, and **inter-process communication** within an operating system. By completing this project, students will acquire essential skills to develop a practical system program using relevant Linux system calls. This experience will empower them to effectively manage processes and facilitate smooth communication between them. We strongly encourage all students to attend the project-related lab for support and hands-on learning.

Program Usage

Your goal is to implement a simplified version of a Linux shell program.
The program name is `myshell`.
Here is the simplest usage:

```
$> ./myshell
Myshell (pid=4442) starts
ITSC> exit
Myshell (pid=4442) ends
$>
```

`$>` represents the system shell prompt (i.e., your system shell, not our PA1 shell program).
The command (`./myshell`) launches our PA1 shell program.

When `myshell` starts, it displays the process ID (`pid`). Please note that the process ID will be different every time when you launch the PA1 shell program.

Our shell program supports the `exit` command. When your shell program exits, it should display the process ID. The start process ID and the end process ID must be the same.

After that, our shell program terminates. You can type commands to the system shell again.

GCC compiler in the lab environment

The default gcc compiler installed in our lab environment will be used.

```
$> gcc --version
gcc (GCC) 11.4.1 20231218 (Red Hat 11.4.1-3)
Copyright (C) 2021 Free Software Foundation, Inc.
```

Getting Started

You don't need to start from scratch.
`myshell_skeleton.c` is the starting point.
To start, please remember to rename the file as `myshell.c`

Read carefully the documentation in the provided code. The skeleton code file provides you many useful helper functions. 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. These 2 functions are too powerful which 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: Start/End the Shell

Implement the exit command handling. Here is a sample test case:

```
$> ./myshell
Myshell (pid=11004) starts
ITSC> exit
Myshell (pid=11004) ends
$>
```

You need to replace ITSC with your own ITSC account name. For example, if your ITSC account is cspeter@connect.ust.hk, you should replace ITSC using cspeter.
For post-graduate students taking this course, DON'T use ITSC alias.

Feature 2: Customized Ctrl-C (SIGINT) handling

Implement a simple customized Ctrl-C (SIGINT) handling. Here is a sample test case:

```
$> ./myshell
Myshell (pid=6449) starts
ITSC> ^CMyshell (pid=6449) terminates by Ctrl-C
$>
```

From the keyboard, press Ctrl-C on the shell prompt. You should see ^C on the screen.
A customized message will be displayed before terminating myshell. A syscall `signal` is used to implement this feature.

Feature 3: 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 (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`). Assume the file is in the current directory. A sample input file redirection usage:

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

Like 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
```

In this project, you are required to handle at most 1 input redirection (<) and at most 1 output redirection (>) in a command.

Feature 4: 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 with at most 8 pipe segments

Some Examples

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 pipe 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.

Given Test Cases

The given test cases are released. You can see the exact commands in the following table:

Test Case (all characters are in one line)	**Description**

`exit`	The exit command handling is given in the skeleton code.
	
	`$> ./myshell`
	`Myshell (pid=11004) starts`
	`ITSC> exit`
	`Myshell (pid=11004) ends`
	
	Make sure ITSC is replaced with your own ITSC account name. Otherwise, you will lose points in this simple case.

`ls`	Running the simplest ls command After running this command, you should see the names of the file in the current working directory
`ls -l -h`	Running a command and there are some tabs and spaces in between the parameters. You should see the output which is equivalent to running the command: `ls -lh`
`echo a1 a2 a3 a4 a5 a6 a7`	This test case is useful to test the upper bound of the number of arguments.
`wc -l < myshell.c`	Assume myshell.c is located in the same directory of the executable of the myshell program, this command counts the number of lines of myshell.c For example, if your current myshell.c contains 200 lines, it should display a number 200
`ls -lh > tmp_out_only.txt`	The output of the command: ls -lh will be redirected to a text file tmp_out_only.txt You can assume tmp_out_only.txt does not exist in the current directory.
`ls | sort`	This test case is a basic 2-level pipe command
`ls | sort -r | sort | sort -r | sort | sort -r | sort | sort -r`	This test case is useful to test the upper bound of the number of pipe segments
`Press Ctrl-C via keyboard`	A customized message will be displayed before terminating myshell

Hidden Test Cases

The hidden test cases won't be released before the project deadline. You cannot see the exact commands, but you can see the description about the hidden test cases.

Code	Description

Hidden01	This test case involves the system shell and 3 myshell programs
	I call these myshell programs: myshell1, myshell2, and myshell3
	
	In the system shell, run ./myshell (i.e., start myshell1)
	Run ./myshell inside myshell1 (i.e., start myshell2)
	Run ./myshell inside myshell2 (i.e., start myshell3)
	
	Run the ps command inside myshell3 to show the current process table.
	Here is a sample process table (the table is different every time):
	
	`PID TTY TIME CMD`
	`10945 pts/1 00:00:00 tcsh`
	`12113 pts/1 00:00:00 myshell`
	`12114 pts/1 00:00:00 myshell`
	`12115 pts/1 00:00:00 myshell`
	`12116 pts/1 00:00:00 ps`
	
	Run exit to quit myshell3, myshell2, and myshell1
	The control should be returned to the system shell.
	Run ps inside the system shell. Here is a sample process table:
	
	`PID TTY TIME CMD`
	`10945 pts/1 00:00:00 tcsh`
	`12117 pts/1 00:00:00 ps`
	
	Please note that all process IDs may be different every time.

Hidden02	A mix of input redirection and output redirection (input, and then output) After that, run cat command to display the content of the text file.
Hidden03	A mix of input redirection and output redirection (output, and then input) After that, run cat command to display the content of the text file.
Hidden04	Run a 2-level pipe command. After that, run another 2-level pipe command.
Hidden05	Run a 2-level pipe command. After that, run another 3-level pipe command.
Hidden06	Start myshell1, myshell2, myshell3 like Hidden01 In myshell3, press Ctrl-C once. All shells should be terminated. Please check the sample Linux executable if you don't understand the expected result.

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
```

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. 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.
2. Automatically 0 marks if `system` or `popen` function is used in your code
3. **Correctness of the given test cases (50 marks)**
    a. The given test cases are equally weighted
    b. The sum will be normalized to 50 marks
    c. There won't be partial credits for each test case
    d. You **CANNOT hard-code** the given test cases.
        i. For example, your program cannot simply use strcmp to compare the text "ls" (one of the given test cases), and then run the "ls" command using execlp("ls","ls"). It is hard-coding because your program only handles the given test cases.
4. **Correctness of the hidden test cases (50 marks)**
    a. The hidden test cases are equally weighted.
    b. The sum will be normalized to 50 marks.
    c. There won't be partial credits for each test case.

Plagiarism

***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!*

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

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)