Project and Grader TA - Peter CHUNG (cspeter@cse.ust.hk)

COMP3511 Operating System (Spring 2024)

PA1: Simplified Linux Shell

Release on 24-Feb (Sat) Due on 08-Mar (Fri) at 23:59

Introduction

The objective of this project is to provide students with a comprehensive understanding of **process management**, **input and output redirection**, and **inter-process communication** in an operating system. Its successful completion will equip students with the practical skills required to develop system programs using relevant Linux system calls. These skills will enable them to proficiently manage processes and establish efficient communication channels between them.

To ensure a thorough grasp of the project's concepts and tools, we strongly encourage students to actively participate in the project-related lab sessions. Attending these sessions will enhance their learning experience and facilitate the practical application of the project's concepts.

Program Usage

Your goal is to implement a simplified version of a Linux shell program.

```
$> ./myshell
COMP3511 PA1 Myshell (Spring 2024)

Myshell (pid=4609) starts
ITSC FULL_PATH_TO_CURRENT_DIR> exit
Myshell (pid=4609) ends
$>
```

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

Upon launching the myshell program, it will display the process ID (pid) associated with it. It's important to note that the process ID will vary each time you start the shell program.

Our shell program includes support for the exit command. When you choose to exit the shell program, it will display the process ID associated with the start of the shell. It is crucial that the start process ID and the end process ID remain the same.

After displaying the process ID, the shell program will terminate, allowing you to input commands into the system shell once again. This ensures a seamless transition back to the system shell environment.

GCC compiler in the lab environment

In this semester, gcc version 11 is the default gcc compiler installed in our lab environment.

```
$> gcc --version
gcc (GCC) 11.4.1 20230605 (Red Hat 11.4.1-2)
Copyright (C) 2021 Free Software Foundation, Inc.
```

Getting Started

Rename the skeleton code file as (**myshell.c**). It is essential to closely review the documentation accompanying the provided code. The skeleton code file offers a wide range of helpful helper functions. Additionally, the related lab sessions will introduce important programming concepts that are directly relevant to the project. 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.

Assumptions

- The input format is valid.
- No commands with both redirection and pipe at the same time.
- Each command line has at most 256 characters (including NULL)
- Each command has at most 8 pipe segments.
- Each pipe segment has at most 8 arguments.
    - Note: `execvp` system call needs to store an extra `NULL` item to indicate the end of the parameter list.
    - You will find the constant is set to 9 (instead of 8) in the starter code.
    - For details, please read the comment lines in the starter code.
- Each command has 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
COMP3511 PA1 Myshell (Spring 2024)

Myshell (pid=4609) starts
ITSC FULL_PATH_TO_CURRENT_DIR> exit
Myshell (pid=4609) 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.
Post-graduate students may have an extra user-friendly email alias. Please don't use that alias as it affects the grading process.

Once the ITSC prompt is displayed, it is important to also display the full path of the current directory. This functionality has been implemented. The full path is used to verify the correctness of the change directory (cd) command below.

Feature 2: Simplified Change directory (cd) Command

Change directory (cd) is a special command which cannot be executed using exec* functions. We need to handle this command separately.

You don't need to implement a change directory like a system shell. For example, you don't need to support cd ~. The system shell supports special syntax such as cd ~, which is used to change to the home directory of the user.

In general, you need to support cd command like this:

- `cd <path>`
- <path> can be an absolute path or a relatively path. Please note that chdir() system call supports . (current directory) and .. (upper directory)

You need to invoke a special system call that is not covered in the lab notes.

```
int chdir(const char *path);
```

chdir() changes the current working directory of the calling process to the directory specified in *path.* On success, zero is returned. On error, -1 is returned.

Here are some possible usages of the change directory command:

```
$> ./myshell
COMP3511 PA1 Myshell (Spring 2024)

Myshell (pid=4644) starts
```

```
ITSC /homes/cspeter/comp3511_s2024/pa1_myshell> cd ..
ITSC /homes/cspeter/comp3511_s2024> ls
pa1_myshell
ITSC /homes/cspeter/comp3511_s2024> cd wrong_folder
Myshell cd command error
ITSC /homes/cspeter/comp3511_s2024> cd pa1_myshell
ITSC /homes/cspeter/comp3511_s2024/pa1_myshell> cd /
ITSC />
```

In this example:

- The starting directory is `/homes/cspeter/comp3511_s2024/pa1_myshell`
- Type `cd..`, the directory becomes `/homes/cspeter/comp3511_s2024`
- You should see that `pa1_myshell` is the only folder in the current directory.
- Type `cd wrong_folder`, an error message is shown.
- Type `cd pa1_myshell`, the current directory is updated appropriately.
- Type `cd /`, the current directory is changed to the root (/) directory.

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 <u>at most 1 input redirection (<)</u> and <u>at most 1 output redirection (>)</u> 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 <u>support multiple-level pipes</u> with at most <u>8 pipe segments.</u>

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 on the same line)	**Description**
`exit`	For details, please refer to the exit command above.

`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 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
`Note: The same case as the cd command example`	Change the directory using the cd commands. For details, please refer to the example above.

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	Run a cd command using a correct directory and a wrong directory. The exact command is not given to avoid hard-coding.

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. Otherwise, the given test cases part will be reset to 0 marks.
        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!*

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)