CS3103: Operating Systems

Spring 2022
Programming Assignment 1

1 Introduction

The purpose of this assignment is to help you better understand process management and system calls in Linux environment. To achieve this goal, you will use C/C++ to implement a mini **shell** that supports some basic functions including running programs in the background.

2 Requirements

2.1 Prompt for User Input [10%]

The mini shell should display the prompt for user input, which includes the information about the path to the current directory. For example, suppose the current directory is /home/assignment1 and you run the mini shell in the Linux terminal,

```
$./shell
```

it should prompt

```
minishell:/home/assignment1$
```

which indicates being ready for user input. Using **getcwd**() declared in <unistd.h> can obtain the current directory.

2.2 Changing the Current Directory [10%]

The mini shell should be able to change the current directory. The command should have the following form:

```
cd [path of the directory]
```

The content of the prompt should also change if the current directory changes. For example,

```
minishell:/home/assignment1$ cd /home
minishell:/home$
```

A useful system call is **chdir**() declared in <unistd.h>.

2.3 Listing the Content of the Current Directory [10%]

The mini shell should be able to list the content of the current directory using the **dir** command. Here is an example where there are two files and two sub-directories in the current directory:

```
minishell:/home/assignment1$ dir
.
..
file1
file2
dir1
dir2
minishell:/home/assignment1$
```

Useful functions include **opendir**(), **readdir**(), and **closedir**() declared in <dirent.h>.

2.4 Background Execution of Programs [20%]

The mini shell should also allow programs to run in the background:

```
run [path of executable file] [a list of arguments]
```

Running programs in the background means that the mini shell is able to accept the next command immediately after executing the **run** command. Below is an example:

```
minishell:/home/assignment1$ run ./test hello 2 5
minishell:/home/assignment1$
```

where "test" is an executable file and "hello 2 5" are the arguments. Also note that the mini shell should allow **ANY** number of processes to exist in the background simultaneously. You can use **fork**() and **execvp**() to implement this command so that the parent process accepts the user input and the child process executes background process.

2.5 Terminating Background Processes [10%]

The mini shell should have a **kill** command which allows the user to manually terminate a background process using the PID of the process:

```
kill [PID]
```

Hint: Using the **kill**() function declared in <signal.h> to send a SIGTERM signal can terminate a process.

2.6 Listing All Background Processes [20%]

The **ps** command should list the information of all processes that are running in the background. The information should include the PIDs and the paths of the running programs. Note that processes terminated by the **kill** command are **NOT** supposed to appear in the list. Here is an example with the correct behavior:

```
minishell:/home/assignment1$ run ./test test1 2 5
minishell:/home/assignment1$ run ./test test2 5 10
minishell:/home/assignment1$ ps
4096: ./test
4097: ./test
minishell:/home/assignment1$ kill 4097
minishell:/home/assignment1$ ps
4096: ./test
minishell:/home/assignment1$
```

2.7 Exiting the Shell [10%]

The **quit** command should cause the mini shell to exit. But before that, make sure all background processes have been terminated.

2.8 Handling Unexpected Inputs [5%]

You can never make any assumption about what users will input in your shell. For example, a user may input an empty line, an invalid command, or a wrong path of the executable file. You are encouraged to come up with these unexpected inputs as many as possible and handle them properly.

2.9 Programming Style [5%]

Your code should have a good programming style (with good readability and necessary in-program comments).

3 Bonus [Extra 10 Points]

Consider the case where a background program completes and exits normally. The **ps** command described above is not aware of the completion of a program and still lists the information of the completed program. Think about how to improve the **ps** command such that the completed programs can be automatically removed from the list.
Hint: A SIGCHLD signal will be sent to the parent process by the kernel whenever a child process terminates. You need to implement a signal handler to properly handle SIGCHLD, and use **signal**() declared in <signal.h> to bind SIGCHLD to the corresponding handler that you implement. **waitpid**() declared in <sys/wait.h> might be a useful system call when handling the SIGCHLD signal.

4 Hints

- Study the man pages of the system calls used in your program. For example, the following command displays the man pages of **kill**():

```
$ man 2 kill
```

- When you use **fork**(), it is important that you do not create a fork bomb, which easily eats up all the resources allocated to you. If this happens, you can try to use the command "kill" to terminate your processes (`http://cslab.cs.cityu.edu.hk/supports/unix-startup-guide`). However, if you cannot log into your account any more, you need to ask CSLab for help to kill your processes.

5 Helper Programs

5.1 args.cpp

This example program shows how to read a line from terminal, as well as parsing (cutting) the string using the **strtok**() function. To compile the program, use the following command:

```
$ g++ args.cpp -lreadline -o args
```

5.2 test.cpp

This program can be used to test your mini shell. It takes three arguments: the first argument is a single word to be displayed repeatedly; the second argument is the number of seconds between two consecutive displays of the word; the last argument is the number of times the word to be displayed. For example, the following command displays the word "running" 5 times in 2-second interval:

```
$./test running 2 5
```

6 Submission

- **Important Note:** You program will be tested on our CSLab Linux server (gateway.cs.cityu.edu.hk). You should describe clearly how to compile and run your program in the text file. **If an executable file cannot be generated and run successfully on our Linux servers, it will be considered as unsuccessful.**
- This assignment is to be done individually. You are encouraged to discuss the high-level design of your solution with your classmates but you **must** implement the program on your own. Academic dishonesty such as copying another student's work or allowing another student to copy your work, is regarded as a serious academic offence.
- Each submission consists of two files: a source program file (.cpp file) and a text file containing user guide, if necessary, and all possible outputs produced by your program (.txt file).
- Write down your name, eid and student ID in the first few lines of your program as comment.

- Use your student ID to name your submitted files, such as 5xxxxxxx.cpp, 5xxxxxxx.txt.
- Submit the files to Canvas.
- The deadline is **11:00am, 18-FEB-2022 (Friday)**. No late submission will be accepted.

7 Questions?

- If you have any questions, please submit your questions to the Discussion board "Programming Assignment #1" on Canvas.
- To avoid possible plagiarism, do not post your source code on the Discussion board.
- If necessary, you may also contact Mr ZHOU Zikang at zikanzhou2-c@my.cityu.edu.hk.