# CSE Project 2: UNIX Shell and History Feature

Wang Ziyi
515030910578
wzy_sjtu@foxmail.com

December 1, 2017

**Abstract**

This is the report for my CSE course project 2: UNIX Shell and History Feature in SJTU. In this project, I modified a C program which serves as a shell interface that accepts user commands, and executes each command in a separate process and creating a history feature for my shell which enables it to remember 10 latest commands and provide a shortcut to re-execute them.

## 1 Project Description

This project consists of modifying a C program which serves as a shell interface that accepts user commands and then executes each command in a separate process. A shell interface provides the user a prompt after which the next command is entered. One technique for implementing a shell interface is to have the parent process first read what the user enters on the command line and then create a separate child process waits for the child to exit before continuing. The separate child process is created using the fork() system call and the user's command is executed by using one of the system calls in the exec() family.
The functions and features of my project include:

- After the prompt "COMMAND->", the user input command and the shell will execute it.
- If the command ends with &, it will run in background.
- The shell can distinguish wrong commands and give an error message.

- Use <Control><D> to exit the shell.
- Use <Control><C> to enter the history interface, where there are 10 most recently executed commands with numbers.
- Input "r" to execute the most recently executed command at once.
- input "r x" to execute the most recently executed command which starts with "x".

# 2 Implementation

The implementation is organized into two parts:(1)creating the child process and executing the command in the child, and (2)modifying the shell to allow a history feature. In the source code, I have a main function which reads in the input command and execute the command if it is formal. Setup() is aimed at formulating the input, tokenizing it so that the array saves the arguments of the command, and additionally in this project, recognize if the command is "r" or "r x". The function DisplayHistory() is called when the user use <Control><C> to require the command history. The history list only stores the most recent 10 commands called, but the number of the commands keeps accumulating without clearing up.
The source code is based on the outline of simple shell and signal-handling program provided by the textbook, and part of the setup() function is cited from websites. Some modifications has been done to the reference codes to apply the specifications of the project. Check the detailed source code and its comments for more explanation.

# 3 Problems and Solutions

The first big problem I encountered is to comprehend the process of forking a child process and use exec() to execute the command. I read the provided code and the relevant chapters of the textbook to get the hang of the core idea of the whole work flow, and what the provided code meant.
The second problem is to create the history feature. To simply remember the executed commands and store them in a array is rather easy actually, but we also need to establish a shortcut to those commands in storage. As the requirement says, the command "r" and "r x" means execute the command in storage, rather than really execute some "r" command(which in fact doesn't exist at all). So in the setup() function, after parsing the input string into

arguments, we need a second check to see whether it is the "r" command, and react accordingly.
Moreover, the "r" command shouldn't go into the history list. Instead, the command it points to does. So in this situation, we need to replace the inputBuffer with the real command instead of "r" command.

# 4 Source Code

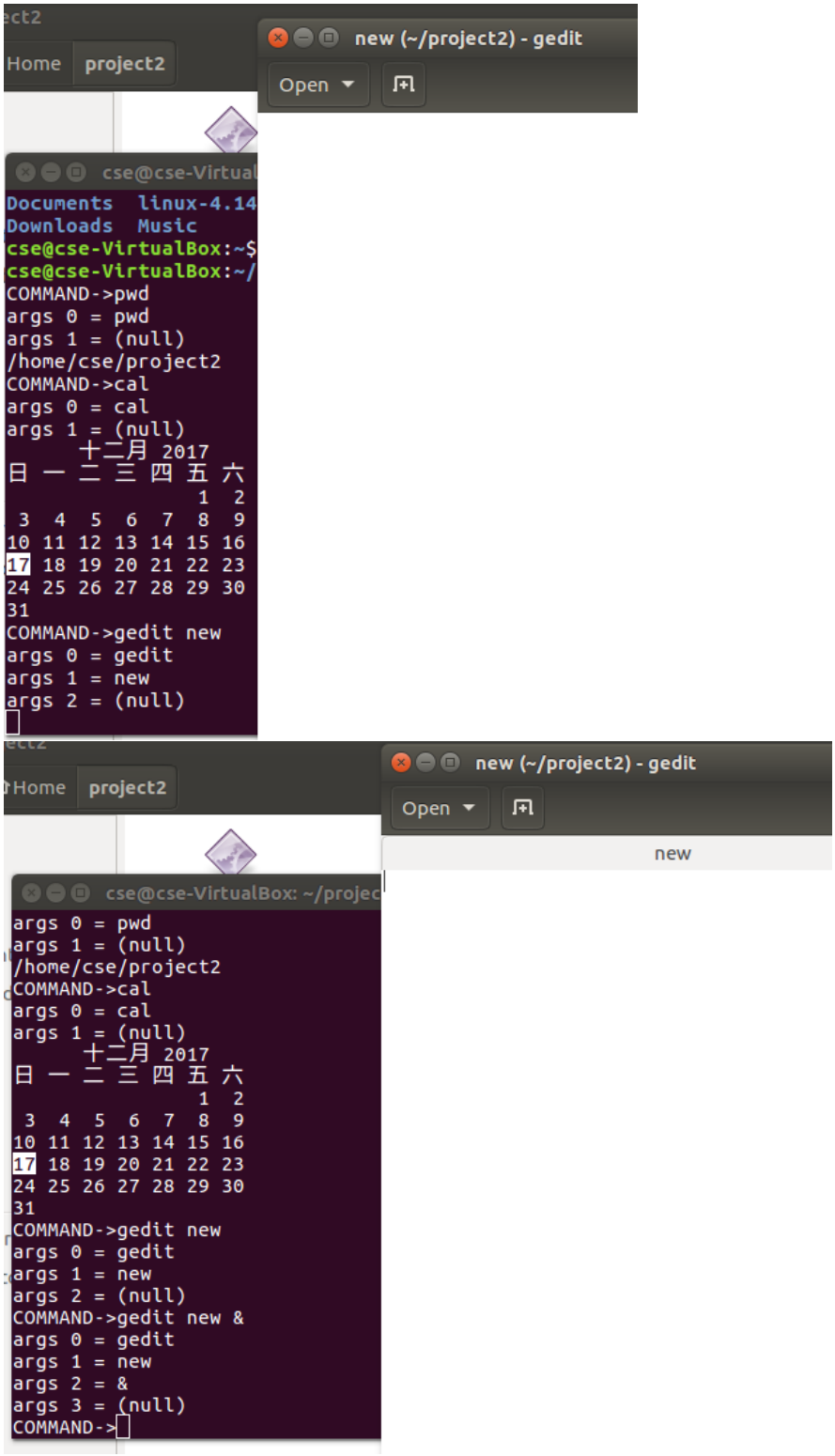
See the file **UnixShell.c**.

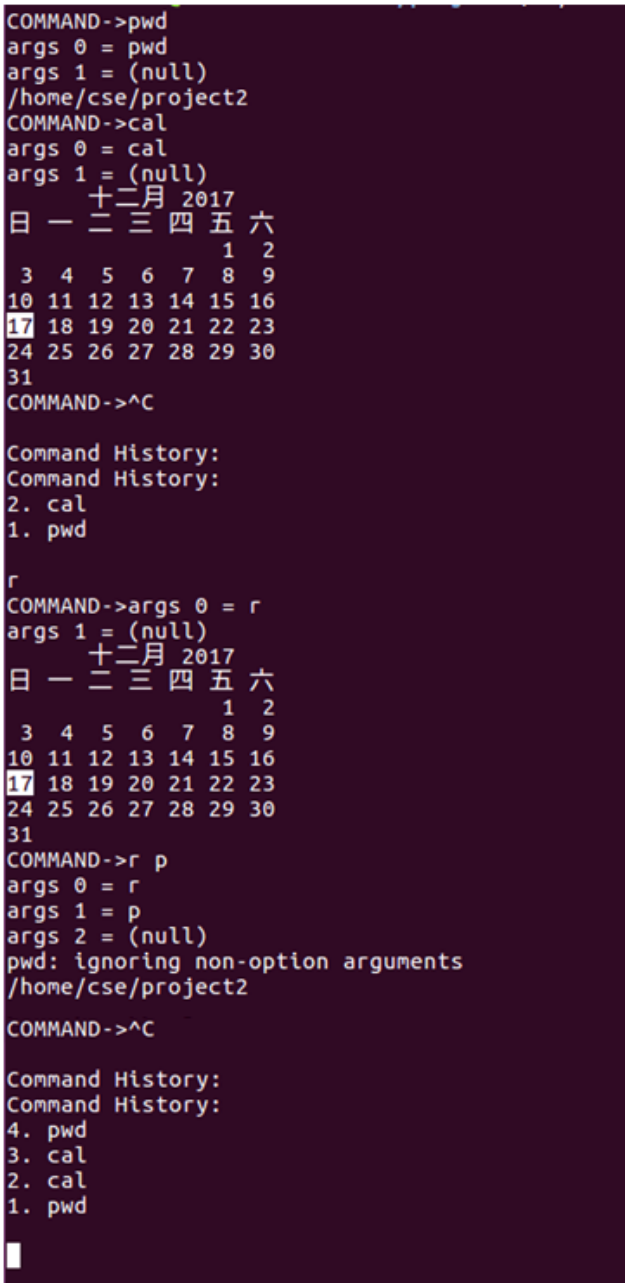
# 5 Conclusion

In conclusion, this project reinforces my understanding on the parent and child processes, and also the signal handling part. It demands some thinking on how to organize and modify your code so that it can achieve the desired outcome. The final result turns out fine and I truly benefited a lot from this project.

# 6 Test

Here are some snapshots for my shell:


Home
project2
new (~/project2) - gedit
Open
cse@cse-Virtual
Documents linux-4.14
Downloads Music
cse@cse-VirtualBox:~$
cse@cse-VirtualBox:~/
COMMAND->pwd
args 0 = pwd
args 1 = (null)
/home/cse/project2
COMMAND->cal
args 0 = cal
args 1 = (null)
十二月 2017
日 一 二 三 四 五 六
1 2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
COMMAND->gedit new
args 0 = gedit
args 1 = new
args 2 = (null)
Home
project2
new (~/project2) - gedit
Open
new
cse@cse-VirtualBox: ~/projec
args 0 = pwd
args 1 = (null)
/home/cse/project2
COMMAND->cal
args 0 = cal
args 1 = (null)
十二月 2017
日 一 二 三 四 五 六
1 2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
COMMAND->gedit new
args 0 = gedit
args 1 = new
args 2 = (null)
COMMAND->gedit new &
args 0 = gedit
args 1 = new
args 2 = &
args 3 = (null)
COMMAND->"


COMMAND->pwd
args 0 = pwd
args 1 = (null)
/home/cse/project2
COMMAND->cal
args 0 = cal
args 1 = (null)
十二月 2017
日 一 二 三 四 五 六
1 2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
COMMAND->^C
Command History:
Command History:
2. cal
1. pwd
r
COMMAND->args 0 = r
args 1 = (null)
十二月 2017
日 一 二 三 四 五 六
1 2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
COMMAND->r p
args 0 = r
args 1 = p
args 2 = (null)
pwd: ignoring non-option arguments
/home/cse/project2
COMMAND->^C
Command History:
Command History:
4. pwd
3. cal
2. cal
1. pwd