COMP2123 Programming technologies and tools

Mini Project – Advanced programming tools

Deadline: 2018 DEC 15 11:55PM

Hello! If you have any questions w.r.t. this project, please first visit our Project Discussion Forum to see if you can find our answer there. If not, please post your questions in the Forum. If you would like to ask through email, please feel free to send email to the email group in moodle, and also cc email to Dr. Cui (heming@cs.hku.hk). We are very happy to answer your questions.

- For all files, please submit them through Moodle, and click the submit button before the deadline.
- Please double check your submission, check ALL of these to ensure that your assignment is submitted properly:

  o Check your email for acknowledgement from Moodle, keep it for record.

  o Check the submission page again and check if the status is changed to "Submitted for grading".

  o Check the submission page to see all files you have submitted.

Please contact us as soon as possible if you encounter any problem regarding the submission. Enjoy this project and let us know if you need our help.

In this mini project, you will be guided to write a small shell-like program in C++ that uses some concepts of child processes. You will learn to use `gdb`, an advanced program debug tool, to help you develop and debug this small program. You will be guided to do self-learning on these new concepts and seek for further tutorials on how to use them.

You are encouraged to form a mini-group (maximum 2) to work on this project because this will promote you to better cooperate. You do not need to register the group members. But you have to state your group information (member's name and UID) in the very beginning of every source file of your program by comments. Each group should only make one submission.

Project Description

You are required to implement a command line interpreter program (shell) that operates in the following way: when you type in a command, the shell creates a new child process that executes the command; after it finishes, it prompts for the next user input. In this project, you are only required to implement two built-in commands, `exit` and `history`. For all the other commands of the shell program, your program should also handle them but you are allowed to do them in a cheating way: the `system()` in C++ is used to invoke a command execution from a C/C++ program.

Built-in commands history and exit

About **`history`** command, you can take a quick look at the man pages (linux command: `man history`). It displays an ordered list of all input commands that a shell received in a session. In this project, by default, the `history` command is only required to display 5 most recent commands (latest command shows first; if less than 5 commands has been executed, just show them all). The first requirement for your shell is that it records all commands it receives from the command line. Then when it receives the input command `history`, it shows the records.

In this project, you are required to implement an additional flag to the history command, which is `-sbu (SortByUsage)`. When a command input is `history -sbu`, the program should sort the history commands by execution duration (end time – start time) in descending order. This means when your shell program executes any commands, it should record the command execution time and store both the executed commands and its time information (you should design your own data structure and sort operation, please seek inspiration from course materials: Topic 5-8).

The other built-in command that your shell supports is **`exit`**, i.e. if the user inputs "`exit`" your shell program will exit. To implement the exit built-in command, you simply call `exit(0);` in your C++ shell program.

The formats for your built-in commands are:

`[optionalSpace]exit[optionalSpace]`

`[optionalSpace]history[optionalSpace]`

`[optionalSpace]history[oneOrMoreSpace][-sbu][oneOrMoreSpace]`

Executing non-built-in commands

Besides executing the built-in commands above, your program needs to executes all commands that it receives (i.e. all commands except `history` and `exit`) by using the following method: The C/C++ library function `int system(const char *command)` in `<stdlib.h>` can be used to execute the command specified by its string argument. Your shell will simply read input from the command line and use `system()` to execute it. Reference of `system()`: http://www.cplusplus.com/reference/cstdlib/system/.

To simply the implementation, you can assume that the program each time only takes one command and all commands have a length limit of 256 bytes. The executable should be `tinyshell` and it should use as a prompt `tinyshell>`.

Sample Input and Output

Executable and prompt:

```
user@yourmachine:~$ ./tinyshell

tinyshell>
```

History built-in command:

```
tinyshell>ls

[ls output]

tinyshell>cd directory

tinyshell>history

cd directory[two space]0.001s

ls[two space]0.002s
```

History sort by usage:

```
tinyshell>history -sbu

ls[two space]0.002s

cd directory[two space]0.001s
```

Debugging your program

In C/C++ development, memory management and usage of pointers are complicated and confused. Sometimes we need some external tools to help develop the program. In this project, we would like you to learn to use the `GNU GDB` tool to help debugging your program. Please refer to the **gdb-lab.pdf** to learn how to use `GNU GDB`.

How to create a child process and collect execution time

Normally, a shell executes commands in processes. In this project, meeting the requirements may not necessarily involve `fork()` to create child processes in C/C++. However, we encourage you to learn how to use `fork()`. Using `fork()` will count 10% of this project, and you may determine whether to try this advanced concept.

Process is a concept in operating system which means an instance of program that is being executed. In this project, a child process is the program that you execute each command and your parent/main process is your shell program that takes command input and collects information. A lot of tutorials/examples about how to quickly use simple process creating in C/C++ can be found online. Here are some references you may want to read:

https://www.geeksforgeeks.org/fork-system-call/

And here is a reference you may want to learn about how to create a child process, execute something and return the execution time:

https://stackoverflow.com/questions/5608984/measuring-time-of-parent-and-children-processes

Submission and Grade

Remember DO NOT directly copy other's code (including code in reference materials) into your project submission to avoid plagiarism. You should learn them and write by yourself.

Please submit your source code and compiled executable `tinyshell` through moodle via a `project.zip` file. You should also include a screenshot that records you using `gdb` to show the contents of your data structure (e.g., show the memory content of the first element of a linked list). Each group are only required to make one submission and make sure you includes your group member information at the beginning of every source code. Followed is the assessment scheme:

Correctness of executable – 50%

Use of fork() – 10%

Good data structure – 20%

Screenshot of `GNU GDB` usage – 20%