COMP 340: Operating Systems

Spring 2024

Project 1: Simple UNIX-style shell

Date Released: 02/08/2024

**Assignment:** In this exercise, you will build a simple UNIX-style shell.

To help you manage the complexity of this assignment, a **shell.c** file is provided for you as part of the assignment. This file contains the declaration of the functions described in the tasks below (Tasks 1-4), each of which implements some functionality required by the shell. The final task (Task 5) uses the functions in Tasks 1-4 to create a simple UNIX-style shell program in the main procedure.

**Task 1** (15 points) Implement the C function shell\_change\_dir() that takes a directory path and changes the current working directory to it. The function must return 0 on success and -1 otherwise.

*Hint:* Look up the man page for chdir.

**Task 2** (15 points) Implement the C function shell\_file\_exists() that takes a file path and determines if the file exists on the local filesystem. The function must return 0 if the file exists and -1 otherwise.

*Hint:* Look up the man page for stat.

**Task 3** (20 points) Implement the C function shell\_find\_file() that takes a file name and checks whether the file exists on the PATH. This function should traverse the directories in the PATH environment variable and find the first directory that contains the file. The function should prepend directory path to the file name and return the resulting file path in the pre-allocated buffer file\_path. The function must return 0 if the file is found on the PATH and -1 otherwise.

*Hint:* Look up the man pages for getenv, strdup, and strsep.

**Task 4** (20 points) Implement the C function shell\_execute() that takes an absolute file path and an array of command line arguments, and attempts to execute file with the arguments. This function should fork a new process, execute the file with the arguments, relinquish control to the child process, and wait for it to terminate.

*Hint:* Use the execv standard library call to create the new process image. If the call to execv returns, which is rare and indicates an error condition, be sure your program calls exit so that the child process terminates normally; it is likely the machine will hang otherwise. Look up execv for more information on this call and its behavior. Also, read the man pages for fork and wait.

**Task 5** (25 points) Combine and extend the functionality in Tasks 1-4 to construct a simple UNIX-style shell. The shell should allow the user to change the current working directory, run commands and executables in the current folder as well as the user’s PATH. The following pseudocode indicates how the main loop of your program might behave:

while (!exit)

{

* display prompt and wait for user input
* if command line contains non-whitespace characters
  + parse command line
  + if the specified command is “exit”, terminate the program taking care to release any allocated resources.
  + if the specified command is “cd”
    - change the current working directory to the specified directory path using shell\_change\_dir()
  + if the command is specified using an absolute path, exists in the user’s PATH or exists in the current folder
    - execute the command using shell\_execute()
* else
  + report an error message

}

The shell should terminate when the user enters “exit” at the prompt.

There are some subtleties not immediately obvious from the pseudocode above, so please think through these issues as you design your shell program.

**Rubric:** To receive credit, submit a .zip file via mygcc containing the following items on due date:

1. **Source code:** Submit a completed shell.c file with the functionality specified in Tasks 1-5 above. You may **not change** the function signatures.

**Your code must compile, link, and execute in the Linux environment using the class VMs: no compile equals no credit. Also, if your code crashes, no partial credit will be given**

In addition, you should include a comments section at the beginning of each of your files that provides information about the file and its intended purpose. For example,

/\*

Author: He (David) Zhang

Course: COMP 340, Operating Systems

Date: 10 September 2021

Description: This file implements the

Shell program

Compile with: gcc -o shell shell.c

Run with: ./shell

\*/

Your comments must include the commands necessary to compile the code and execute the program, as illustrated above.

The points available for each task are distributed according to the following weights:

Correctness: 100%

Each minor error: -20%

Each major error: -40%

Overall comments across all tasks are 5 points of the project grade.

**Extensions will not be granted for technology-related issues.** Leave yourself enough time to complete the assignment, submit the assignment using mygcc, and contact the instructor if you run into problems.

**Project Policies:**

* Assignments must be submitted electronically via my.gcc. Be sure to upload your files correctly the first time.
* This project is a group project. Every student needs to work as a part of a 2-person team. Each student must submit a code implementation.
* 20% of the grade will be weighed with the peer evaluation. Students are expected to turn in the peer evaluation form posted on mygcc at **the end of the semester**. If a student works on his or her own without a team, then 20% of the grade will be deducted from the project grade.
* Students are expected to keep the same team for the whole semester.

**Academic Integrity Policy:**

* Each team is expected to work on its own. Members of each team can work together, discuss ideas, and look at each other’s code if you have to.
* Students belonging to different teams should not discuss, share code that directly bears on this project, or look at each other’s code. Any instances of this will be considered a violation of the academic integrity policy of this course and will be reported to the SFRC committee.
* Use or possession of past solutions and similar solutions from online resources is strictly prohibited and is considered a violation of the academic integrity of this course.
* You may use online resources to look up how to use a function or a system call, but you may not copy code from online resources. Any copied code (whether cited or not) is considered a violation of the academic integrity policy of this course.