DALHOUSIE UNIVERSITY
FACULTY OF COMPUTER SCIENCE

CSCI 3120 Operating Systems

Assignment 2: A Simple Shell for Linux

Due: 16:00, Oct. 6, 2023

- **Teaching Assistants:**
  - Kamran Awaisi (km521977@dal.ca)
  - Hui Huang (huihuang@dal.ca)
  - Arka Ghosh (arka.ghosh@dal.ca)
  - Yitong Zhou (yt760204@dal.ca)
  - Shiyun Wang (sh776410@dal.ca)

- **Help Hours: FCS Learning Center (Goldberg 233) or via the Channel "Office Hour - TAs" on MS Teams:**
  - Monday: 1:00pm-2:00pm, Kamran Awaisi
  - Tuesday: 1:00pm-2:00pm, Hui Huang
  - Wednesday: 1:00pm-2:00pm, Arka Ghosh
  - Thursday: 1:00pm-2:00pm, Yitong Zhou
  - Friday: 1:00pm-2:00pm, Shiyun Wang

---

1. Assignment Overview

In this assignment, you need to design and implement a C program that serves as a shell (i.e. command-based interface) for Linux. The shell accepts user commands and executes each command using a separate process. In addition, the shell provides a history feature that allows users to access the recently-entered commands.

2. Important Note

There is a zero-tolerance policy on academic offenses such as plagiarism or inappropriate collaboration. By submitting your solution for this assignment, you acknowledge that the code submitted is your own work. You also agree that your code may be submitted to a plagiarism detection software (such as MOSS) that may have servers located outside Canada unless you have notified me otherwise, in writing, before the submission deadline. Any suspected act of plagiarism will be reported to the Faculty's Academic Integrity Officer in accordance with Dalhousie University's regulations regarding Academic Integrity. Please note that:
1) The assignments are individual assignments. You can discuss the problems with your friends/classmates, but you need to write your program by yourself. There should not be much similarity between your program and others' programs.

2) When you refer to some online resources to complete your program, you need to understand the mechanism, then write your own code. Your code should not be similar to the online resources. In addition, you should cite the sources via comments in your program.
3) You may use AI-driven tools to assist your learning, but you are not allowed to use them to produce work to be submitted for course evaluations. Your submitted program should not be highly similar to others' programs.

3. Detailed Requirements

1) Overview: In this assignment, you need to design and implement a C program that serves as a shell for Linux. The shell accepts user commands and executes each command using a separate process. In detail, the shell gives the user a prompt, after which the next command is entered. The example below shows the prompt `CSCI3120>` and the user's next command: `cat prog.c` (note that this command displays the content of the file `prog.c` on the terminal using the UNIX command `cat`).

```
CSCI3120> cat prog.c
```

One technique for implementing a shell is to have the parent process first read what the user enters on the command line (in this case, `cat prog.c`), and thereafter create a separate child process that executes the command. Unless otherwise specified, the parent 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 using one of the system calls in the `exec()` family.

A C program that provides the general operations of a shell is shown in the code snippet below. The `main()` function displays the prompt `CSCI3120>` and outlines the steps to be carried out after the input from the user has been read. The `main()` function continually loops as long as `should_run` equals 1; when the user enters `exit` at the prompt, your program will set `should_run` to 0 and terminate.

```
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

#define MAX_LINE 80 /* The maximum length command */

int main(void)
{
      char *args[MAX_LINE/2 + 1]; /* an array of character strings */
      int should_run = 1; /* flag to determine when to exit program */

      while (should_run) {
            printf("CSCI320>");
            fflush(stdout);

      /**
```

```
        * After reading user input, the steps are:
        * (1) Fork a child process using fork()
              Note that: The fork() system call is invoked inside a while
              loop. If the while loop is not well programmed, it will
              invoke fork() continuously, potentially leading to many
              child processes. When there are too many dangling processes
              under your account, your CSID will be suspended. You are
              encouraged to use the commands in Appendix 2 to monitor your
              processes. And if you cannot log into "csci3120.cs.dal.ca"
              using your CSID any more, you need to contact with
              cshelp@cs.dal.ca to unlock your CSID.
        * (2) The child process will invoke execvp()
              Note that:
              (i) If the user input is an invalid command, execvp() will
              not replace the child process with a binary executable.
              Instead, execvp() returns the value of "-1". In this case,
              the child process should check the return value and invoke
              exit() system call to terminate itself. Otherwise, the child
              process will keep running, potentially leading to the
              situation where the parent waits for the child to terminate
              and the child becomes a clone of the parent (note that, in
              this case, the child would create its own child process if
              the user enters another command).
              (ii)If the user input is a valid command, exit() does not
              need to be invoked because the child process will be
              replaced with the binary executable corresponding to the
              valid command, and the executable can be terminated
              properly. The details can be found in Section 3.4.b.c.

        * (3) Parent will invoke wait()
        */
        }

        return 0;
}
```

Specifically, this assignment consists of two tasks:

a) Creating the child process and executing the command via the child process.
b) Modifying the shell to allow a history feature.

The details of these tasks will be described in the following sections.

2) Creating a Child Process: The first task is to modify the `main()` function in the code shown above so that a child process is created using `fork()` and thereafter the child process replaces itself with the command specified by the user using `execvp()`. To accomplish this task, your program needs to parse what the user has entered into separate tokens and store the tokens in an array of character strings (i.e. `args` in above code). For example, if the user enters the command `ps -ael` at the `CSCI3120>` prompt, the values stored in the `args`

array are (note that the user did not enter NULL; however, it needs to be added explicitly because the following function execvp() requires NULL):

```
args[0] = "ps"
args[1] = "-ael"
args[2] = NULL
```

This `args` array will be passed to the `execvp()` function, which has the following prototype:

```
execvp(char *command, char *params[]);
```

Here, `command` represents the command to be executed and `params` stores the parameters for this command. For this assignment, the `execvp()` function should be invoked using the following command:

```
execvp(args[0], args)
```

Please note that:

a) A webpage illustrating how to use execvp() can be found here: https://www.geeksforgeeks.org/exec-family-of-functions-in-c/

b) The user input is essentially a series of symbols (i.e. characters, numeric digits, or punctuation marks) with the new line character (i.e. '\n') at the end. Please note that:
   a. The tokens in the user input, such as "ps" and "-ael", are separated by space characters (i.e. ' ').
   b. There is only one space character between each pair of neighboring tokens.
   c. The new line character should not be part of the last token. Otherwise, execvp() will not be executed properly.

c) The user input is composed of 1 symbol at the least and 80 symbols at the most. In addition, the user input is composed of 41 tokens at the most.

d) There are a variety of different methods that you can use to get the user input. One simple method is based on fgets(). Here are two links about fgets():
   a. https://www.geeksforgeeks.org/fgets-gets-c-language/
   b. https://man7.org/linux/man-pages/man3/fgets.3p.html
   Another method that you could use is based on scanf().

e) User input parsing: There are a variety of different ways to parse user input and generate the required tokens. One simple way is to utilize "strtok()". Here are two links about this function:
   a. http://www.cplusplus.com/reference/cstring/strtok/
   b. http://man7.org/linux/man-pages/man3/strtok.3.html
   Note that:
   a. strtok() breaks a string into a sequence of zero or more nonempty tokens. On the first call to strtok(), the string to be parsed should be specified as a parameter. In each subsequent call that parses the same string, the corresponding parameter must be NULL.
   b. strtok() could adopt multiple delimiters. For example, " ,.-" is a string including four delimiters (note that the first delimiter is a space character); " \n" is a

string including two delimiters (note that the second delimiter is a new line character). Here is one example illustrating how to use multiple delimiters: https://www.cplusplus.com/reference/cstring/strtok/

f) FYI: Here are two webpages on C strings and how to use an array of pointers to character to store a list of strings:
   a. https://www.tutorialspoint.com/cprogramming/c_strings.htm
   b. https://www.tutorialspoint.com/cprogramming/c_array_of_pointers.htm

3) Creating a History Feature: The second task is to modify the shell program so that it provides a history feature that allows the user to save the recently-entered commands along with their process IDs. The user will be able to list the command history by entering the following command at the `CSCI3120>` prompt:

```
history
```

The user should also be able to save up to 10 commands with this feature. The most recently entered command will always be 1. This means that the older commands will be pushed down each time a new command is entered, thus always keeping only the 10 most recent commands in history.

As an example, assume that the history consists of the commands (from most to least recent):

```
ps -ael, ls -l, top, cal, who, date
```

The command `history` will output:

```
ID    PID  Command
1     4339 ps -ael
2     4336 ls -l
3     4335 top
4     4332 cal
5     4330 who
6     4299 date
```

where:

a) ID is the command ID in history (ranging from 1 to 10, 1 being the most recent),
b) PID is the process identifier of the process that executed the command
c) Command is the command executed by the user (including the arguments)

Please note:

a) You can obtain the PID of the child process via the fork() system call.
b) The maximum PID on Linux depends on the specific Linux version. You can use the following command to check the maximum value: "cat /proc/sys/kernel/pid_max".
c) For simplicity, in this assignment, the command "history" is considered a special command. When "history" is executed, no child process is "forked" and the command "history" will not be added to command history.

4) Additional Requirements: Your program should also satisfy the following requirements.

a) Your program should support two techniques for retrieving and executing the commands from the command history:
    a. When the user enters !!, the most recent command in the history is executed (e.g. in the above case, !! will execute `ps -ael`).
    b. When the user enters a single `!` followed by an integer *N*, the *N*th command in the history is executed (e.g. in the above case, !3 should execute `top`) .

    Note that:
    a. The real command (not the symbol "!!" or "!N") executed in this fashion should be displayed on the user's screen so that the user knows which command is executed.
    b. In addition, the real command (not the symbol "!!" or "!N") should be placed in the history as the most recent command.
    c. You can assume that N is in the range of 1 to 10.

b) The program should also manage basic error handling.
    a. If there are no commands in the history, entering !! or `!N` or `history` should result in a message "`No command in history!`" on the screen. In this case, !! or `!N` or `history` is NOT added to the history.
    b. If there are commands in the history, but there is no command corresponding to the user input "!N", the program should output "`Such a command is NOT in history!`". In this case, "!N" is NOT added to the history.
    c. Any other invalid command should result in a message "`Invalid command!`" on the screen. In this case, the invalid command is added to the command history.
        i. Note that the return value of the exec() family of system calls is "-1" when there is an error, you can utilize this feature to detect invalid commands. The details of the exec() family of system calls can be found here: http://man7.org/linux/man-pages/man3/execvp.3.html
        ii. Also note that after checking the return value of the exec() family of system calls and confirming the user input is an invalid command, the child process needs to invoke exit() system call so that the child process terminates itself. Otherwise, the child process (which is the same as the parent process in this scenario because exec() did not overwrite the child process successfully) will continue running as a second shell (i.e. a shell on top of the original shell). When the second shell exists, if the user enters `exit` at the prompt, the second shell will terminate. The termination of the second shell will be detected by the parent shell using the wait() system call. Afterwards, the parent shell will continue to receive user input. Consequently, overall, the user needs to enter `exit` more than once to terminate your program. If your program needs multiple `exit` to be terminated, it is likely that the invalid command is not properly handled. The details of exit()system call can be found here: http://man7.org/linux/man-pages/man3/exit.3.html
        iii. In the scenario that a valid command is entered, the child process does not need to invoke exit() system call explicitly because the child process

will be able to overwrite itself with the valid command, and the valid command will terminate itself properly.

c) Your program must use the GNU C library (i.e. glibc) to implement the required features. Here are a few links about glibc:
   a. https://www.gnu.org/software/libc/documentation.html
   b. https://www.kernel.org/doc/man-pages/
   c. http://man7.org/linux/man-pages/index.html

d) Compiling and running your program on "csci3120.cs.dal.ca" should not lead to errors or warnings. To compile and run your program on "csci3120.cs.dal.ca", you need to be able to access the command-line interface of "csci3120.cs.dal.ca". In addition, you need to be able to upload a file to or download a file from "csci3120.cs.dal.ca".
   a. To access the command-line interface of "csci3120.cs.dal.ca", several different methods could be used. Here are two widely used methods:
      i. MS Windows Computer: You can use the software tool "putty" on MS Windows computers. With "putty", the connection type should be set to "SSH". Note that "putty" can be downloaded via the following link: https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html .
      ii. Mac and Linux Computer: On Mac and Linux computers, you can use the command "ssh" to access "csci3120.cs.dal.ca" via the program called "Terminal".
   b. To transfer files between your computer and "csci3120.cs.dal.ca", several different methods could be used. Here are two widely used methods:
      i. MS Windows and Mac Computer: Cyberduck is popular tool used to transfer files between two computers via SFTP (i.e. SSH File Transfer Protocol). You can download Cyberduck from the following webpage: https://cyberduck.io/ . This document includes an appendix that provides an introduction to Cyberduck.
      ii. Linux Computer: On Linux computers, you can use the command "scp" to transfer files. Here is a tutorial on the command "scp": https://www.linuxtechi.com/scp-command-examples-in-linux/.
   c. You should upload your program to "csci3120.cs.dal.ca" to verify that it works well on "csci3120.cs.dal.ca". After verifying that your program works on "csci3120.cs.dal.ca". You should submit your assignment in brightpace. For submission details, please proceed to the submission section of this document. Note that the TA will NOT look at your files on "csci3120.cs.dal.ca". Instead, the TA will retrieve your submission from brightspace.

e) For simplicity, we assume that the commands supplied by the users are external commands. How to know whether a command is external or not can be found here: https://www.geeksforgeeks.org/internal-and-external-commands-in-linux/

f) For simplicity, we assume that the commands supplied by the users do not include pipe or redirection symbol. In addition, the following symbols are not included in the commands from the users: "||", "&&", "&", ";".

5) Readme File: You need to complete a readme file named "Readme.txt", which includes the instructions that the TA could use to compile and execute your program on "csci3120.cs.dal.ca".

6) Submission: Please pay attention to the following submission requirements:

a) You should place "Readme.txt" in the directory where your program files are located.
b) Your program files and "Readme.txt" should be compressed into a zip file named "YourFirstName-YourLastName-ASN2.zip". For example, my zip file should be called "Qiang-Ye-ASN2.zip".
c) Finally, you need to submit your zip file for this assignment via brightspace.

Note that there is an appendix at the end of this document, which includes the commands that you can use to compress your files on "csci3120.cs.dal.ca". In addition, there is another appendix, which illustrates how to view and kill your processes on "csci3120.cs.dal.ca".

4. Grading Criteria

The TA will use your submitted zip file to evaluate your assignment. The full grade is 18 points. Your submission will be evaluated from the following perspectives. The specific rubric used by the marker is included in a separate file.

a) "Readme.txt" with the correct compilation/execution instructions is provided [1 Point]
b) User commands are properly parsed and executed via a child process [8 Points]
c) History feature saves commands and their arguments along with their process ID [4 Points]
d) Commands are properly executed via a new process when they are provided using !! and !*N* [2 Points]
e) Basic error handling is done as specified in Section 3.4.b of this assignment [2 Points]
f) Proper programming format/style (e.g. release the memory that is dynamically allocated when it is not needed any more; proper indentation/variable names, etc.). [1 Point]

Please note that when "Readme.txt" is not provided or "Readme.txt" does not include the compilation/execution instructions, your submission will be compiled using the standard command `gcc -o A2 A2.c (or your filename)`, and you will receive a zero grade if your program cannot be successfully compiled on "csci3120.cs.dal.ca".

5. Academic Integrity

At Dalhousie University, we respect the values of academic integrity: honesty, trust, fairness, responsibility and respect. As a student, adherence to the values of academic integrity and related policies is a requirement of being part of the academic community at Dalhousie University.

1) What does academic integrity mean?

Academic integrity means being honest in the fulfillment of your academic responsibilities thus establishing mutual trust. Fairness is essential to the interactions of the academic community and is achieved through respect for the opinions and ideas of others. Violations of intellectual honesty are offensive to the entire academic community, not just to the individual faculty member and students in whose class an offence occur (See Intellectual Honesty section of University Calendar).

2) How can you achieve academic integrity?

- Make sure you understand Dalhousie's policies on academic integrity.
- Give appropriate credit to the sources used in your assignment such as written or oral work, computer codes/programs, artistic or architectural works, scientific projects, performances, web page designs, graphical representations, diagrams, videos, and images. Use RefWorks to keep track of your research and edit and format bibliographies in the citation style required by the instructor. (See http://www.library.dal.ca/How/RefWorks)
- Do not download the work of another from the Internet and submit it as your own.
- Do not submit work that has been completed through collaboration or previously submitted for another assignment without permission from your instructor.
- Do not write an examination or test for someone else.
- Do not falsify data or lab results.

These examples should be considered only as a guide and not an exhaustive list.

3) What will happen if an allegation of an academic offence is made against you?

I am required to report a suspected offence. The full process is outlined in the Discipline flow chart, which can be found at:
http://academicintegrity.dal.ca/Files/AcademicDisciplineProcess.pdf and includes the following:
a. Each Faculty has an Academic Integrity Officer (AIO) who receives allegations from instructors.
b. The AIO decides whether to proceed with the allegation and you will be notified of the process.
c. If the case proceeds, you will receive an INC (incomplete) grade until the matter is resolved.
d. If you are found guilty of an academic offence, a penalty will be assigned ranging from a warning to a suspension or expulsion from the University and can include a notation on your transcript, failure of the assignment or failure of the course. All penalties are academic in nature.

4) Where can you turn for help?

- If you are ever unsure about ANYTHING, contact myself.
- The Academic Integrity website (http://academicintegrity.dal.ca) has links to policies, definitions, online tutorials, tips on citing and paraphrasing.
- The Writing Center provides assistance with proofreading, writing styles, citations.

- Dalhousie Libraries have workshops, online tutorials, citation guides, Assignment Calculator, RefWorks, etc.
- The Dalhousie Student Advocacy Service assists students with academic appeals and student discipline procedures.
- The Senate Office provides links to a list of Academic Integrity Officers, discipline flow chart, and Senate Discipline Committee.

Appendix 1: How to Use Zip and Unzip on "csci3120.cs.dal.ca"

To compress:

```
zip squash.zip file1 file2 file3
```

To uncompress:

```
unzip squash.zip
```

this unzips it in your current working directory.

Appendix 2: How to View/Kill Your Processes on "csci3120.cs.dal.ca"

Due to programming mistakes, you might leave a number of running processes on "csci3120.cs.dal.ca". When you leave too many running processes "csci3120.cs.dal.ca", your "csci3120.cs.dal.ca" account could be locked temporarily (then you need to talk to CS Help to unlock your account). To view/kill your processes on "csci3120.cs.dal.ca", you can use the following commands. Please keep an eye on your processes on "csci3120.cs.dal.ca".
1) Command to display the processes belonging to a specific ID (i.e. UID): **ps -u UID**
2) Command to kill a process using process ID (i.e. PID): **kill -9 PID**
3) Command to kill all processed belonging to a specific user ID (i.e. UID): **pkill -u UID**

Appendix 3: Introduction to Cyberduck

1. Optional Registration: Cyberduck is free software that supports file transfers via SSH File Transfer Protocol (i.e. SFTP). You can choose to pay for the registration to avoid seeing the

donate/buy window when you quit the program. But it is perfectly ok to use Cyberduck without the registration. The only drawback is that you will see the donate/buy window when it is terminated (in this case, you can click on “Later” to completely terminate the program).

2. Set Up Connection: SFTP (SSH File Transfer Protocol) is used to set up the connection
    a. Start Cyberduck
    b. Click on the icon “Open Connection”
    c. Set Protocol to “SFTP (SSH File Transfer Protocol)”
    d. Set Server to “csci3120.cs.dal.ca”
    e. Enter your CSID and password
    f. Check “Add to keychain” on macOS or check “Save Password” on MS Windows
    g. Click on “Connect”
3. Upload/Download Files: “drag-and-drop" is supported
    a. Uploading: select a file on your computer and thereafter drag the file into the Cyberduck interface
    b. Downloading: select a file in the Cyberduck interface and thereafter drag the file to the proper destination