DALHOUSIE UNIVERSITY
FACULTY OF COMPUTER SCIENCE

CSCI 3120 Operating Systems

# Assignment 1: A Simple Shell for Linux

## Due: 11:55pm, Oct. 6, 2019

- **Teaching Assistants:**
  - Lauchlan Toal (lc790935@dal.ca),
  - Patricia Kibenge-MacLeaod (p.kibenge@dal.ca)
  - Hui Huang (huihuang@dal.ca)
- **Help Hours at CS Learning Center (2nd Floor of Goldberg Building):**
  - Tuesdays: 1pm-3pm, Hui Huang
  - Thursdays: 2pm-4pm, Patricia Kibenge-MacLeaod
  - Fridays: 12pm-2pm, Lauchlan Toal

---

### 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.

### 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 illustrates 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 then 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 taken after 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>

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

int main(void)
{
char *args[MAX_LINE/2 + 1]; /* command line arguments */
int should_run = 1; /* flag to determine when to exit program */

   while (should_run) {
      printf("CSCI3120>");
      fflush(stdout);

      /**
       * After reading user input, the steps are:
       * (1) fork a child process using fork()
       * (2) the child process will invoke execvp()
       * (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 forked and executes the command specified by the user. This will require parsing what the user has entered into separate tokens and storing 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:

```
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)
```

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 by using the 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 (maximum from 1 to 10, 1 being the most recent),
b) PID is the process identifier of the process that executed the command, and
c) Command is the command executed by the user (including the arguments)

Please note that you can obtain the PID of the child process via the fork() system call.

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

a) Your program should also 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`)

   Any command executed in this fashion should be echoed on the user's screen. The command should also be placed in the history buffer as the most recent command.

b) The program should also manage basic error handling.
   a. If there are no commands in the history, entering !! should result in a message "`No commands in history`."
   b. If there is no command corresponding to the number entered with the single !, the program should output "`No such command in history`".
   c. Any invalid commands should result in an error but still be added to the history buffer.
c) Your program must use the GNU C library (i.e. glibc) to implement the required features.
d) Compiling and running your program on bluenose.cs.dal.ca should not lead to errors or warnings.
   a. To compile and run your program on bluenose, you need to be able to upload a file to or download a file from bluenose. Here is a tutorial on bluenose downloading/uploading: https://web.cs.dal.ca/~society/#/
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/

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 bluenose.

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-ASN1.zip". For example, my zip file should be called "Qiang-Ye-ASN1.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 bluenose.

## 4. Grading Criteria

The TA will use your submitted zip file to evaluate your assignment. The full grade is 20 points. The details of the grading criteria are presented as follows.

- "Readme.txt" with the correct compilation/execution instructions is provided [1 Point]
- User commands are properly parsed and executed via a child process [8 Points]

- History feature saves commands and their arguments along with their process ID [5 Points]
- Commands are properly executed via a new process when they are provided using !! and !*N* [4 Points]
- Basic error handling is done as specified in Section 3.4.b of this assignment [2 Points]

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 A1 A1.c (or your filename)`, and you will receive a zero grade if your program cannot be successfully compiled on bluenose.

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: How to Use Zip and Unzip on Bluenose

To compress:

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

To uncompress:

```
unzip squash.zip
```

this unzips it in your current working directory.