CSC 389 Programming Project 2
Shell History Enhancement

Project 2 is built on Project 1 with the following additional requirements:

1. A **history file** (e.g., "**history.bin**" for a binary file or "**history.txt**" for a text file) that stores the command history and occurrence count which can be retrieved and updated in later shell sessions;
2. A new command "**mfu**" that displays the top 5 **most frequently used** commands and their corresponding occurrence count;
3. Your source code should be adequately **commented** for better readability. Also, C is a block-structured programming language. Blocks include functions and control structures (such as if-else, while-loop, for-loop) where **indentation** should be applied properly.

Example of the **history file** operation

If in the first shell session, the following 8 commands are entered:

(least recent) `ps, ls -l, cal, ps, ls, date, cal, exit` (most recent)

Since the "exit" command ends the session, it does not need to be stored in the history file. Therefore, the history file should contain the following information:

"cal"	(2 occurrences)
"date"	(1 occurrence)
"ls"	(1 occurrence)
"ps"	(2 occurrences)
"ls -la"	(1 occurrence)

a) The exact history file content format (including binary or text) can be decided by you. It is recommended that data (command itself and its occurrence count) be written into the file in certain order (most recent first or least recent first) so that the "**recent**" command (see Project 1 Part II requirement) can work properly after the file data is read in again in a later session. Otherwise, the "recent" information needs to be stored in the file for later use.

b) When the second (or later) shell session starts, it should read the information from the history file before processing any user commands. Then, each newly entered command should be added into the history buffer accordingly (including updating the occurrence count).

c) When the "exit" command is entered to end a session, the updated history buffer information should be written into the history file again.

A C program example showing how to write/read structure data to/from a binary file can be found in the attachment.

Example of the "**mfu**" command

When the command "**mfu**" is entered, the shell should display the **current top 5** most frequently used commands and their corresponding occurrence count line by line in a non-increasing order (i.e., highest count first). If there are less than 5 such commands, the shell should display all of them. For example (if there are only three),

"cal"	(2 occurrences)
"date"	(1 occurrence)
"ls"	(1 occurrence)

In summary, a successful implementation (for both Project 1 and Project 2) would typically utilize the following components:

Programming Components	*References*
Process management system calls: fork(), wait(), execvp(), exit().	Textbook 1.6.1 and Fig. 1-19.
History buffer and history file with command occurrence count; The command "mfu" (most frequently used).	The opposite concept of the Least Recently Used page replacement in textbook sections 3.4.6 and 3.4.7.
C programming: While-loop, for-loop, if-else control, pointer, array, multidimensional array, character string, string function, structures, file operations (fopen, fread, fwrite, fclose)	"C and Unix/Linux" folder in Blackboard: - C Programming (part 1): Basics - C Programming (part 2): Pointers and Arrays - C Programming (part 3): Structures - C Programming (part 4): Files

What to turn in?

(Due on **May 10** via Blackboard)

a) A copy of your source code (working version). For this project, it is preferred to have all your source code included in one single C file (e.g., project2.c).
At the **top of your source code file**, add the following information (as comments):

```
/* Author: (Your Name, UIN, UIS email)
   Compile: (how to compile your source code on the UISACAD server)
   Brief Description: (what does this program do?)
*/
```

b) **One document file** (Microsoft **Word** or **PDF** format) that contains the following **parts**:

i) There are several requirements specified in Project 1 and Project 2.
Describe which parts you have implemented successfully.
(1) Creating a child process (Project 1 Part I): the child process is created and the args parameters are passed to execvp() and the command is executed successfully;
(2) Creating a history feature (Project 1 Part II): the "recent", "!!", and "!N" commands work properly;
(3) The history file operation (Project 2): the operations of writing to and reading rom history file work properly;
(4) The command "**mfu**" (Project 2) works properly;
(5) Source code is adequately commented and indentation is applied properly (Project 2).

ii) Use **clear screenshots (with explanation)** to demonstrate that your code works (for those parts that you have successfully implemented).