CSC 389 Programming Project 1
Shell Interface

This project consists of designing a C program to serve as a shell interface that accepts user commands and then executes each command in a separate process.

The shell interface will print out this prompt:

COMMAND->

After seeing this, the user can enter the command (for example, "cat prog.c" typed without the quotation marks; for those that are curious, this command displays the file prog.c on the terminal).

One technique for implementing a shell interface 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 performs the command. Then 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 execvp() (which is one of the system calls in the exec() family).
A C program that provides the general operations of a command-line shell is supplied in Figure 1.

```
#include <stdio.h>
#include <unistd.h>

#define MAX_LINE 80 /* The maximum length command */
#define MAX_ARGS 10 /* The maximum number of arguments */

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

      while (should_run){
            printf("COMMAND-> ");
            fflush(stdout);

            /**
            * Follow these steps:
            * (1) read user input
            * (2) if the command inputted is "exit" then set should_run to 0
            * (3) fork a child process using fork()
            * (4) in the child process, invoke execvp()
            * (5) in the parent process, wait for the child to complete
            */
      }
}
```

Figure 1: Outline of simple shell

This project is organized into two parts: (1) creating the child process and executing the command in the child, and (2) modifying the shell to allow a history feature.

Part I – Creating a Child Process

The first task is to modify the main() function in Figure 1 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 (args in Figure 1). For example, if the user enters the command `cat prog.c` at the COMMAND-> prompt, the values stored in the args array are:

args[0] = "cat"
args[1] = "prog.c"
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 performed and params stores the parameters to this command. For this project, the execvp() function should be invoked as execvp(args[0], args).

Part II – Creating a History Feature

The next task is to modify the shell interface program so that it provides a history feature that allows the user to access the most recently entered 10 commands. That is, the user will be able to list the command history by entering the command

recent

at the COMMAND-> prompt. (This "recent" command is **not** to be passed to the execvp() function.)
As an example, assume that the history consists of the commands (from most to least recent):

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

The command "recent" will output:

```
6 ps
5 ls -l
4 top
3 cal
2 who
1 date
```

Your program should support two techniques for retrieving commands from the command history:

a) When the user enters !!, the most recent command in the history is executed.
b) When the user enters a single ! followed by an integer N, the Nth command in the history is executed.

Continuing our example from above, if the user enters !!, the `ps` command will be performed; if the user enters !3, the command `cal` will be executed. Any command executed in this fashion should be echoed on the user's screen. The command should also be placed in the **history buffer** (which you need to create) as the next command.

The program should also manage basic error handling. If there are no commands in the history, entering !! should result in a message "No commands in history." If there is no command corresponding to the number entered with the single !, the program should output "No such command in history."

(Note: This project material is adapted from an assignment in Operating System Concepts Essentials second edition by A. Silberschatz, P. Galvin, and G. Gagne, John Wiley & Sons, Inc.)

What to turn in?

(Due on **April 9 (Monday)** 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., shell.c).
b) One document file (Microsoft **Word** or **PDF** format) that contains the following **parts**:
   i) What's the proper **gcc command** to successfully compile your source code on the UISACAD Linux server? For example, are there any special options/flags needed (i.e., other than a simple **gcc shell.c**)?
   If your source code is located in multiple directories, please create a **makefile** so that a simple **make** command can work.
   ii) Use **screenshots** to demonstrate that your code works (for both Part I and Part II).