Hands-on Exercise Two

Finish all the tasks below. Submit the code for all questions, that is, submit three ***.c*** programs. Use the code name in the format: e2_t#_XXXX.c, where XXXX indicate the last four digits in your student ID, # is the task no. Add the proper descriptions as the comments at the beginning of the code, including author, student ID, function description, and create date. Refer to Rubrics for the grading criteria.

You must do this assignment on Linux (Three ways have been introduced in the class for you to access Linux).

The *goal* of this homework assignment is to use *system calls* to create your own *simple command line interpreter* (similar to the "Command Prompt" program on Microsoft Windows, or the "Terminal" program on Mac OS X, or shell programs on Linux).

Here is the sample code of a small C program that uses the *printf* and *fgets* functions from the C standard library to do input and output:

```
/**********************************************************/
#include <stdio.h>
#define SIZE 1024
int main(void) {
   char prompt[] = "Type a command: ";
   char buf[SIZE];
   // Ask the user to type a command:
   printf(prompt);
   // Read from the standard input the command typed by the user (note
   // that fgets also puts into the array buf the ’\n’ character typed
   // by the user when the user presses the Enter key on the keyboard):
   fgets(buf, SIZE, stdin);
   // Replace the Enter key typed by the user with ’\0’:
   for(int i = 0; i < SIZE; i++) {
      if(buf[i] == ’\n’ || buf[i] == ’\r’) {
         buf[i] = ’\0’;
         break;
      }
   }
   // Execute the command typed by the user (only prints it for now):
   printf("Executing command: %s\n", buf);
   return 0;
}
```

/**/

Task 1

In this task, you will learn how to use system calls instead of C standard library functions.

1. Compile this program on your computer and make sure you can execute it.
2. Understand the programs on the *slides 28* in *ch2_OS_Structures lecture*. Then rewrite the above program by using the system calls rather than C standard library functions, i.e., replace *printf* and *fgets* functions.
3. Make sure you *#include* in your program the correct header files for all the system calls you are using.
4. Run the resulting program that is supposed to work exactly like the original program above.

Task 2

In this task, you will practice on the creation of a child process and execute it. In the program from task 1, instead of just printing back to the screen what the user typed, we now want to use what the user typed as the name of a program to execute as a new child process. So do the following:

1. remove the printing code at the end of the program (the code that appears after the "Execute the command typed by the user" comment and before the "return 0;");
2. replace that printing code with code to create a new child process; this new child process is going to use what the user typed as the name of the program to execute in the new process.
3. Refer to example on slide 23 in *ch3_Processes lecture*. Use the *fork* and *execlp* (or *execvp refer to appendix*) system calls like we saw in class; use *buf* as the first and second arguments of *execvp.* If the *execvp* system call fails then make sure that the new child process immediately dies, otherwise both the parent and child will keep running your program!
4. Add code so that the parent process waits until the child process is dead:
5. Use the wait system call as in the example on slide 23 in *ch3_Processes lecture*.
6. Make sure you *#include* in your program the correct header files for all the system calls you are using;
7. Make sure your program prints error messages if it fails to create the new child process or if the child process fails to execute the program specified by the user. This will help you debug your code in this question and in the next question.
8. Run the program and test that it correctly creates a child process that executes the program indicated by the user. Test your program by typing for example: */usr/bin/xcalc* at the prompt of your program; Linux's calculator must then appear on the screen.

Task 3

At this point, your program reads input (command) from the user only once, and creates a new child process only once. To transform your program into a simple command line interpreter, the only thing left to do is to change the code of your program to use a loop. In this way your program asks the user for input, creates a new child process, waits for the child process to end, then asks the user for input again, creates a new child process again, waits for the child process to end again, then repeats the whole thing again, over and over.

So do the following:

1. Add a "while" loop around the code that executes the command typed by the user (around all the code you modified in the previous question, and nothing else);
2. The test of the loop must stop the loop if the text typed by the user is the word "exit", otherwise the code inside the loop must be executed (use the *strcmp* function from the C standard library to compare strings);
3. After your program has finished waiting for the child process to die, the code inside the "while" loop must print the prompt again and read the next input from the user. The code must then also replace the Enter key typed by the user with a '\0' before the user input is tested again at the beginning of the "while" loop. To do all this, simply copy the input/output code and "for" loop which are at the beginning of your program (before the "while" loop) and paste this code inside the "while" loop (at the end of the code inside the "while" loop).
4. Running the program must then allow the user to type and execute multiple commands one after the other, and each time your program must create a new process for that command, until the user types "exit", at which point your program stops.
5. Try the following program name as the commands.

```
==============================================================
Type a command: /usr/bin/xcalc
Type a command: /usr/bin/xeyes
Type a command: /usr/bin/xload
Type a command: exit
==============================================================
```

The program you have created is then a very simple version of the "Terminal" program (CLI) on Linux.
Submit above 3 tasks into iSpace.