Homework one

COMP3033 OS 1001-1003 2st semester 2021-2022

Answer all the questions below. Once you are finished, write a report, copy **the code of Question 3 to the end of the report** and **submit the report** on the due date. Put your name (in English) and student ID number on the report. Also, **add some pictures of your program running**. Please write down any problems you faced during this process and how to fix them. In addition, **upload the C file** for Question 3 **on iSpace**.

Late homework assignments will not be accepted unless you have a valid written excuse (medical, etc.). You must do this assignment alone. No teamwork or "talking with your friends" will be accepted. No copying from the Internet. Cheating means zero.

You can do this assignment on a computer running Linux only.

In all cases, you will need a C compiler, such as:
- gcc on Linux Ubuntu (you should install Ubuntu desktop on VM VirtualBox, it includes gcc).

Make sure you start this homework assignment early so that you have time to find, download, and install all the software you might need!

0) The goal of this homework assignment is to use system calls to create your own simple command line interpreter.

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("%s", 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;
}
/*************************************************************/
```

Since this program uses only the C standard library, it can work without any change on all operating systems. Compile this program on your computer and make sure you can execute it. If you have any problem compiling or executing this program then contact us immediately for help!

Question 1

1) Instead of using the printf and fgets functions from the C standard library, we want instead to have the program directly use system calls to do input and output. So do the following:
- delete the "#include <stdio.h>" line from the beginning of the program (you will not need this line anymore since your program will no longer use functions from the C standard library);
- replace the printf and fgets function calls with system calls:

- replace "printf" with "write"; the first argument of write must be a "file descriptor" for the standard output stream, which is the integer 1;
- replace "fgets" with "read"; the first argument of read must be a "file descriptor" for the standard input stream, which is the integer 0;
- see Section 2 of https://www.kernel.org/doc/man-pages/ for more information about the read and write system calls.
- WARNING: there is a difference between the size of an array, and the length of the string stored inside an array. Be careful about this when writing or reading data using system calls.
- if necessary, use the strlen function from the C standard library to compute the length of a string that you want to print;
- make sure you #include in your program the correct header files for all the system calls you are using;

- when you print the user's input back to the screen, a final newline character might be missing, so just do one extra system call to print "\n" (the string "\n", which you can directly give as argument to the system call, not the character '\n') on the screen before the program ends.

The resulting program must then work exactly like the original program above.

Question 2

2) 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 in a new child process. So do the following:

- 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;");
- 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:

- use the wait system call as we saw in class.
- make sure you #include in your program the correct header files for all the system calls you are using;
- 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.

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.

Question 3

3) At this point, your program reads input 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 so that 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:

- 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);
- the test of the loop must stop the loop if the text typed by the user is the word "quit", otherwise the code inside the loop must be executed (use the strcmp function from the C standard library to compare strings);
- 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).

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 "quit", at which point your program stops.

For example, on a Linux computer:

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

The program you have created is then a very simple version of shell programs on Linux.

Here are a few extra instructions:
- (5 points) Give meaningful names to your variables so we can easily know what each variable is used for in your program.
- (5 points) Put comments in your code (in English!) to explain WHAT your code is doing and also to explain HOW your program is doing it.
- (5 points) Make sure all your code is properly indented (formatted). Your code must be beautiful to read.
- (10 points) Include some pictures of your program running, even if your program is not completely finished.

Failure to follow these instructions will result in you losing points.

Good Luck!