Project: Shell

Purpose: The purpose of this project is to familiarize you with the mechanics of process control through the implementation of a shell user interface. This includes the relationship between child and parent processes, the steps needed to create a new process, shell variables, and an introduction to user-input parsing and verification.

Task 1.1: Shell interface.

The shell (command line) is just a program that continually asks for user input, perhaps does something on the user's behalf, resets itself, and again asks for user input. *Design and implement a basic shell interface that supports the execution of other programs and a series of built-in functions.* The shell should be robust (e.g., it should not crash under any circumstance beyond machine failure).

Basic: The prompt should look like this:

```
prompt$
```

Advanced: The prompt should look like this:

```
machinename@username:~$
```

where **`machinename`** *and* **`username`** *should change depending on the machine and user.*

Task 1.2: Shell programs/commands.

Basic: Implement the basic functionality of the following programs: **`wc`**, **`grep`**, **`df`**, **`cmatrix`**.

Intermediate: Provide a few options and/or arguments for at least two programs. Additional points for creativity (e.g. implementing something that does not exist in bash, or differently than it is done in bash).

Advanced: Allow piping or at least redirecting output to a text file.

Task 1.3: System calls.

Basic: Within the C-programming example of your choice, implement the following system calls: **`fork()`**, **`wait()`**, and **`exec()`**.

Intermediate: Within the C-programming example of your choice, implement **`clone()`**, **`execle()`**.

Additionally: Carefully explore and then implement the **`forkbomb`**.

Task 1.4: Add some colors to your shell and name it.

Task 1.5: Provide a concise and descriptive answer to the following questions.

Question 1.5.1: If we have a single-core, uniprocessor system that supports multiprogramming, how many processes can be in a running state in such a system, at any given time?

Question 1.5.2: Explain why system calls are needed for a shared memory method of inter-process communication (IPC). If there are multiple threads in one process, are the system calls needed for sharing memory between those threads?

Question 1.5.3: Consider the following sample code from a simple shell program. Now, suppose the shell wishes to redirect the output of the command not to STDOUT but to a file "foo.txt". Show how you would modify the above code to achieve this output redirection.

```
command = read_from_user();
int rc = fork();
if(rc == 0) { //child
  exec(command);
}
else {//parent
  wait();
}
```