Forking Processes Assignment

Operating Systems

In most OS classes students are assigned to implement parts of an OS. They are usually given a small partially implemented OS and asked to, for example, implement the Paging system, Scheduler, or other parts. Although this is a good way of learning those implemented parts, this requires lots of programming. I believe that this is not a good approach in our case (you can learn a lot more during the time you would spend programming). We learn the algorithms anyway, implementation is tedious and requires more programming time that I expect you to do in this course.

The projects I have chosen for this course are designed to make you write the types of programs that most of you haven't done before and I believe are important to be exposed to, as security professionals.

Concurrent/parallel programming is a very important part of efficient computation. To create concurrent processes in C/C++, you use "fork" and create children processes. You can then have child processes to execute programs. You should be careful when writing concurrent processing applications and when you create sub-processes. Creating too many sub-processes may crash your system. It is a good idea to check the list of all the processes you have running in your system and kill the ones that have not been terminated properly.

Using your UNIX/LINUX C compiler (you can use JMU's stu machine), run the following program with these 3 different options:

a) Displaying the output on the screen.

b) Directing the output to a file (using ">").

c) Append the output (using ">>") to another file few times.

<u>Questions:</u>

1. **How many times does the program display "I'm Alive!" for each of the above three cases?**
2. **Comment the fflush(stdout) statements out and run the program again. How many times does the program display "I'm Alive!" for each of the above three cases? Explain your answer.**

Run the program on our "stu" machine and on other Linux machines you have access to and compare your findings.

Please look up fork, getpid, and getppid instructions, from your Linux book or use the man page. A process executing a fork instruction creates a child process (after executing the fork command you have 2 processes, the parent process that executed the fork and the child process that was created by the fork).

Depending on your compiler you may have to include other header files.

```
 #include <stdio.h>
#include <stdlib.h>
main ()
{
        int k;
        printf ("Main Process' PID = %d\n", getpid());
        fflush(stdout);
        for (k = 1; k <= 3; k++)
        {
                fork ();
                printf ("k = %d, PPID = %d, pID = %d, I'm Alive!\n", k,    getppid(), getpid());
                fflush(stdout);
        }
}
```