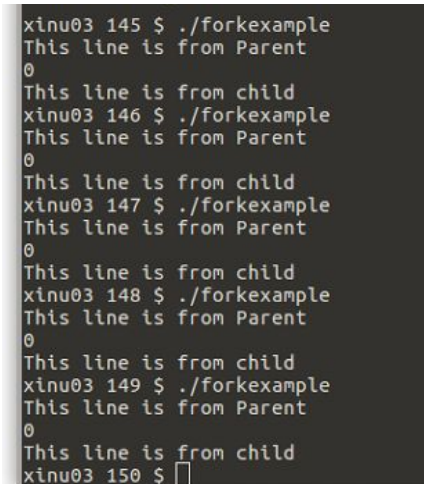
Kathleen Masterson
Lab1Answers

3.1
a. The source code of nulluser() is defined in system/initialize.c starting at line 43
b. In the header file process.h we can see that the ancestor of all processes assign itself the PID 0 - in line 23 we see #define NULLPROC 0
c. The ancestor process running the nulluser function does nothing during the rest of its existence
d. No, nulluser does not return after being called by the code in start.S
e. The source code for halt is located in intr.S, it does nothing forever
f. XINU ran the same as before when I removed the function call from start.S. This is probably because the nulluser function doesn't return because it has its own do nothing forever loop at the end of it. By removing halt we are just removing redundant functionality
g. By replacing the while-loop at the end of null user with a call to halt XINU once again ran the same as before. This is because the halt method does the same as the while-loop.. it did nothing forever. We simply replaced one way to loop forever with another way.

3.2
a. In Linux after a new process is created using fork the process that called fork becomes the parent process. The process we created using the function fork is now the child process.
b. When I ran test code on the frontend with a simple program that creates a new process using fork() the parent process was always executed before the child process. I included a screenshot of the program as well as repeated instances of the output showing the parent process is being executed first.



c. As an app programmer my preference on whether the child or parent process should run next in Linux would depend fully on the application I was currently working on. I would want it to be consistent so I can rely on it to determine the behavior of my application. But it would be a design choice. The child running first would be preference for me since then we could wait on it to run the parent process and could design my program accordingly.

d. -

e. newProcess uses fork which creates a process which is an exact copy of the process which called fork() and the new process is automatically the child process and you are then required to call execve to run code. With create you specify in the parameters the address of the function you want to run but you do not actually run the function, you would have to use “resume()” to do that.

f. Clone and Create are very similar. Both create child processes in order to execute a specified function.

g. They are very similar, posix_spawn can actually be used to replace many fork/exec pairs. Both new processes created from each of these methods get a copy of all the open file descriptors from the parent process that called this function.

h. The best way to create processes is determined on your systems/program’s needs. Posix_spawn is better than form when using systems that have difficulty with fork, but may not always be the best decision. Create does not require additional calls to execve to execute a function. However you need to include the create method in your program to use it.

3.3

-

3.4

-