3.1

a: The function nulluser is initialized in system/intialize.c

b: The definition for the ancestor process is in include/process.h. The ancesstor process is defined as NULLPROC with a value of 0 as the PID.

c: The ancestor process, for the rest of its existance, runs an infitie loop of doing nothing. This is for the purpase that the CPU will have soemthing to run when no other process is ready to execute.

d: No, nulluser never returns back to start.S

e: The source code of halt is located in system/intr.S and its purpose is to do nothing forever.

f: Removing the function call seems to have no effect/noticable difference as XINU runs the same as before.

g: I also was not able to find any noticable difference in the OS functionality.

3.2

a: When fork() is called in in Linux, the OS creates a new process that has a different PID than its parent, the child inherits all open file descriptors of the parent, stack and heap regions are copied and are the same until one tries to write to it (when this happens, a new page is created with the same data -- called copy on write), and the child process waits for the parent to reclaim it using wait or waitpid (otherwise a zombie process)

b: When running my own simple code, it seems the parent process always prints first. This isn't technically true because of context switching and output buffering, but gauging by just my observations, the parent goes first.

c: As an app programmer, you would want the child process to run first because of the overhead of copying the parent's memory. Having the child process finish first would reduce the overhead.
d: newProcess.c is in the system/ folder.

e: Create() simply initializes a process and leaves it hanging as it returns the pid. A call to resume() is needed. newProcess that I just created forks and executes the path given.

f: The call clone() in Linux doesn't copy the memory space, but rather modifies the same memory space (just like create). This works like threads and can be very dangerous. The child also starts in an entry function with clone (just like create). The child using clone has its own stack space.

g: posix_spawn functions the same as our newProcess implementation, but the posix call has a flag option to reduce overhead of the child process.

h: There is no default best way to create a process; the way of creating a process will be project/system dependent.

3.3

XINU greeting should now include my name and username in the specified format.

3.4

The main1.c and onandon.c run as specified as well as printing the proctable. When doing so, the pid of onandon doesnt show up.