Brow1325
Connor Brown

3.1:

a) Do some detective work to locate the places where NPROC is defined. Based on what you find, determine how many processes can exist in XINU.
    i) NPROC is defined in the process.h file. It allows for 8 total processes.
b) Locate where the constant NULLPROC is defined and determine its value. Using detective work, determine the C type of of the priority field in the process table of XINU. Do not just provide the final answer but describe the procedure you followed to arrive at the answer. Specify the maximum and minimum priority values, at least in principle, that a XINU process may take. Where in this range does the priority of the NULL process fall?
    i) NULLPROC is defined in process.h. We can see from the definition of INITPRIO in process.h that the initial priority of the NULLPROC is 20. The process table that stores processes is a struct that is also defined in process.h. A process can have a priority anywhere from -32768 to 32767 as pri16 is defined as an int16 and so the priority is the range of those values.

3.3

b) Which runs first after using fork()? Child or parent?
    i) Based on my findings it would seem that parent runs first after using fork. However, it seems both the parent and child have the same priority so it is possible that child could execute first instead of the parent.
c) Which process will run first after rcreate() is called when the process running main() is created in 3.3(a)?
    i) In rcreate() the randomness of which executes first is more apparent as there are more processes that get run after the fork and so when the priorities are the same it can execute either task first.

3.4

a) Explain how passing of the two arguments is accomplished following (i.e., being consistent with) the CDECL convention.
    i) It is consistent with CDECL convention to call trap as we are given the call for trap as well as the necessary id for the interrupt,
b) Based on the code you find in panic(), explain what happens to XINU at the end dividing by zero in an app generates interrupt 0.

i) Based on the code in panic(), after it is done printing the message for handling the error, it puts the computer into an infinite loop to keep it busy. Forcing the user to reboot.

3.5

a) Inspect the code of this virtual caller in system/ and determine what it does.
    i) The create function works by creating a new process that begins its execution at the funcaddr. After initializing all the variables it places the process into the process table and pushes all the arguments needed for the process into the stack for that process. And when it is done it is given a process id.
b) In the special case where the process that terminates is the last process outside the NULL process, determine what happens by continuing to follow XINU's source code. In the chain of functions being called starting with the virtual function to which main() returns, what is the last function that is called, where is its source code, and what does it do?
    i) The last function called is the resume function which is called after the create function is called to create the shell in main. The source code is located in resume.c. It unsuspends a process making it ready to be executed.
c) In environments where power consumption is an important issue -- e.g., battery powered mobile devices, or cloud computing servers where overheating must be prevented -- is XINU's approach to "shutting down" after the last process has terminated a good solution? What might be a better solution? Explain your reasoning.
    i) XINU’s approach to “shutting down” can be a good solution if it is going to be a long time till there is going to be another process running.as it can conserve energy. But if there is going to be a short interval until another process starts then it would be better to not shut down as it would use more energy starting up again in order to run the process.

Bonus:

a) In Linux/UNIX system programming, we say that the fork() is special in that it is the only system call that "returns twice." What does that mean?
    i) We say that it returns twice because when we fork we create a new process. At that point we have two processes and so the system call would in turn be “returned twice”, meaning once for each process.
b) Why does this feature of fork() not apply to XINU's create()?
    i) This feature does not apply to create() because after running create() the process is not ready and cannot be triggered, but after we fork(), the child

process is ready as well and so that is why it works for fork() and not for create().

c) We say that Linux/UNIX's execve() system call is special in that it is the only system call that does not return when it succeeds. What does this mean?
    i) It does not return when it succeeds because execve() us used as a way to run executable files. But it does this mainly as an overlay and instead of creating a new process, it keeps the same process id and then replaces all of its text, data, heap, and stack. And then the process runs as normal and so there is no need to return.
d) Why does this not apply to XINU's create()?
    i) This does not apply to XINU’s create() because when we create a process we have to add it to a process table, but execve does not have to create a new process, just change one that is already running.