Problem 3.1

a)

Number of processes (NPROC) is defined in /include/process.h. From the definition, the maximum number of processes in the system should be 8.

b)

NULLPROC is located in /system/initialize.c in the function sysinit(). It has priority of 0, state of current, name of prnull, stack that is defined as NULLSTK.

When a process is created, the function been called is create(). So I checked create.c and there is a condition that if priority is less than 1, it will throw syserr. And from the comment on the argument, priority is always greater than 0. I found process priority is type pri16, which is int16 according to kernel.h,. And from limits.h, the maximum value of int16 is 32767. So process priority can be from 0 to 32767 (but normal process starts from 1, only null process can be priority 0). This indicates that the priority of null process is the lowest in the range, only runs if no other process is awaiting to run.

c)

No text-based answer is required.

Problem 3.2

No text-based answer is required.

Problem 3.3

a)

No text-based answer is required.

b)

In linux/Unix, according to the output in parentchild.c. The priority of both parent and the child is the same (which is 0 in my case). However, the order of outputs indicates that parent is running before child process switches in. The created process that runs main() will run first than the null process which calls rcreate(). Because the created process has priority of 20 while null process has priority of 0.

Problem 3.4

In cdecl, subroutine arguments are passed on the stack. Arguments passed to the callee function is pushed into stack before call is called in assembly where the first item in the stack is the first argument and the second item is the second argument. And in this case, the first argument is inum starting from $0 and the second argument is stack pointer which is %eax in intr.S. https://en.wikipedia.org/wiki/X86_calling_conventions

According to system/panic.c. When panic() is called, it will print the panic message and then stays in a busy while loop forever, since Xinu is on single core processor, all other processes will stop processing when the busy while loop is entered.

Problem 3.5

In the function create() in create.c, it sets the return address as INITRET, which by definition is userret() according to process.h. Userret is a function in system/userret.c that forces process to exit once it's called.

Kill() is called by userret() and is located in kill.c. It detects whether last user process completes (--prcount <= 1). If so, it calls xdone(). Xdone() is located in xdone.c and calls halt(). Halt() is inside intr.S. according to the code, halt will disable interrupts and enter a dead loop.

Extra credit

"fork() returns twice " means that when fork is performed, fork() returns 0 in the child process and child pid in the parent process, so that you get to know whether you are running in the parent or the child at the current time. I think "the feature of fork() not being applied to Xinu's create()" is because that Xinu puts a new created process in suspended state and resume() is needed to actually make the new process ready to run. So there is no need to return twice to indicate whether parent or child is running.

"execve doesn't return on success" means that exevce only returns -1 to inform its failure if it fails and does not return on success. Because If it succeeds, the code that called execve() will already be replaced by the successful execution of that function. So you don't need to return anything. Xinu doesn't use it because we don't need to worry about function return since what initret does is to force process to exit.