3.4

I created 4 process, but did not resume(create()) the first process, which was the receiver. I initialized a global variable with the PID of that receiver process. The next 3 processes sent the messages ‘H’, ‘I’, and ‘!’ to the original receiver. In each sender process, I called xfctnl(IPCSND,IPCBLK). Once the 3 processes were in the queue, I resumed the receiving processes and when I ran XINU, the receiver program printed out “HI!”. Alternatively, when I commented out the xfctnl function in the sender processes, the receiver program printed out “H”. Leading me to believe that my send() blocking/non-blocking implementation works. Much like a semaphore queue, I think the processes should be added back to the readylist - the message being sent may be minimal to the program.

My tests are in system/lab4test.c and can be accessed by main().

4

I acquired both the value for the base of the stack and the length of the stack via the process table entry of currpid. If the **value** of the pointer was greater than the base of the stack, or less than the base of the stack minus the stack length, and error occured. If not, my specific test case prints out the message and senderid.

5.2

Since we know the length of ctxsw, we are able to find ebp for resched() and then able to edit the return value replacing it with the function pointer to funcB() instead of going back to sleepms(). I personally implemented it a different way using C instead of inline assembly and found the correct address that way (offset stkptr by 30). When overwriting the return address, we are essentially tricking the process saying that funcB called resched(). For 5.3 I used inline assembly.

5.3

This question required knowledge of CDECL & caller/callee convention. Since I knew we were replacing the return address of resched in 5.2, before overwriting it I saved it separately in a global variable to be used later by funcB(). I used inline assembly to overwrite the return address of funcB (offset ebp by 4) to make it go back to sleepms().

Bonus

When calling resched(), it is assumed interrupts are already disabled thus when changing the return address of resched() the attacker stays in kernel mode. For sleepms(), it enables interrupts (after disabling them) before it calls return so there is a fundamental difference in Linux/Windows but in XINU since there is no separation of user/kernel mode, there is no fundamental difference.