3.1

I incremented NQENT by 2*NPROC. Because by the definition of Xinu's queuetab, each element from index 0 to NPROC-1 corresponds to a process ID, and elements queuetab[NPROC] through queuetab[NQENT] correspond to the heads or tails of lists. Since we need a queue for every process, which has one head and one tail for each queue, so I add 2*NPROC.

3.3

I have considered 6 tests to gauge correctness. Each test prints prhasmsg, prsenderflag and prblockedsenders and checks whether the outputs are as expected on conditions such as:

1. receive is called when prhasmsg is false
2. receive is called when prsenderflag is true and we perform dequeue
3. receive is called when prhasmsg is true and prsenderflag is false
4. bsend is called when prhasmsg is true and we perform enqueue
5. bsend is called when prhasmsg is false

test 1(if receiver does nothing, do senders pile up in queue?):

create 5 sender processes and 1 receiver process

all other processes calls bsend() and receiver does nothing.

Result: print 0 letter and queue has 4 processes left

test 2(if senders do nothing, does receiver block wait?):

create 0 sender processes and 1 receiver process

The receiver calls receive() repeatedly.

Result: print 0 letter and queue has 0 process left.

test 3(test if prblockedsenders has correct number of sender processes left in queue):

create 6 processes, 10 processes calls bsend() once, one calls receive() five times.

Result: print 5 letters and queue has 4 processes left

test 4(test if receive and bsend works on normal condition):

create 6 processes, 5 processes calls bsend() once, one calls receive() five times.

Result: print 5 letters and queue has 0 process left

test 5(test if receive and bsend work repeatedly):

create 6 processes, 5 processes calls bsend() infinitely, one calls receive() infinitely.

Result: consistently print letters and queue's state is as expected

test 6(to test if NQUENT is large enough and queue is working properly when processes max out):

create 95 sender processes and 1 receiver process to reaches maximum NPROC.

all other processes calls bsend() and receiver process calls receive() 95 times.

Result: print 95 letters and queue has 0 process left

4.2

I read through the code of create.c and ctxsw. And I found out that from top to bottom (lowest address to highest), the stack has 8 general purpose register, 1 flag, 1 ebp for process exit. Then there is the function address. So, I got the prstkptr and went up by 10 to get the function address, I changed the function address to the function address of hellomalware.

4.3

To accomplish the second goal, which is find where the local variable x is at, I made an iterator at prstkbase. I iterated through the prstkbase to find where the local variable is at in the stack. Then I modified its value to 9.

To accomplish the third goal, I saved the original second half of resched before changing function address to quietmalware. Then after the modification, I moved the prstkptr 1 position up, which is where the return address resides. And then I changed the return address to the saved value. So now after quietmalware is finished, The victim process returns from resched() to sleepms() which returns to victimA() and outputs the second kprintf().

Bonus question:

We can do similar as wakeup.c. First we will have a while loop that checks if sender queue is not empty. In the while loop, we first dequeue() each process. Since receiver process is no longer valid, we don't want these bsend processes to proceed what's left in its instructions and attempt to write receiver's message buffer when they are made current again. Thus, we can do something similar to the idea of

hijacking. We will modify the function address to detour to a function that does nothing but return SYSERR. If necessary, we will also modify the return address so that once bsend process execute return SYSERR, it has a correct place to return to. After the modification is done, we will call ready(). The while loop will also be wrapped around resched_cntl(DEFER_START) and resched_cntl(DEFER_STOP). So once all processes are put into ready list, resched() will be called.