3.1

Increase NQENT in include/queue.h to accommodate per-receiver process blocked sender queues. Explain in Lab4Answers.pdf why your change of NQENT works.

The new definition of NQENT that I provide is:

#define NQENT (NPROC + 4 + NSEM + NSEM + NPROC + NPROC)

This works because each process has its own private queue, so there needs to be space for a head and a tail for each process (NPROC + NPROC). We do not need to add anymore space than that because each process can only belong in one queue at a time. i.e. additional space os only needed for the head and the tail for the prblockedsenders queue in each process.

3.3

Test your kernels mods for implementing bsend(). Describe in Lab4Answers.pdf what test scenarios you have considered to gauge correctness.

I created three processes from main: (1) first sending process (2) second sending process (3) a receiving process.

I tested to make sure that sending one message from (1) to (3) still works as expected.

The next test was to send multiple messages from (1) to (3) back to back so that the process gets put in the PR_SND state. Process (3) should receive messages in the order that they are received and should receive all the processes.

The next test was to send multiple message from (1) to (3) back to back and send messages from (2) to (3) back to back at the same time. The receiving process should receive the messages in the order that they are sent and should receive all the processes.

These gauge a level of correctness because some aspect of the new implementation was tested. The processes sending queues, prsndmsg and other functionalities all work as expected.

4.2

With knowledge of how XINU's context-switch works, the attacker finds the address at which the return address of ctxsw() (which returns to its caller resched()) has been pushed. Explain in Lab4Answers.pdf how you determine this address.

After the function exits ctxsw() it will return to resched by using a call to ret which returns to the corresponding value in the stack. In x86, the value just under the base pointer (i.e ebp + 4) is the value at which the function will return. Therefore, overwriting this address to the desired return address will yield the correct results.

4.3

Explain in Lab4Answers.pdf your method for accomplishing the second and third goals.

Second goal: the code of quietmalware() finds out the address of the local variable x of victimA() and modifies its value from 5 to 9

Unfortunately, the only solution that I found to work was to iterate through the stack (starting at the address just above the return address (base pointer + 8)) and for every value that is equivalent to five, change it to 9.

Third goal: Third, after quietmalware() is finished it jumps to the instruction of resched() that ctxsw() would have returned to

In order to accomplish this, in attackerB.c, I push the value of the original return address onto the stack. Then, in quietmalware.c, the original return address is popped off the stack and is used to overwrite the existing return address. This ensures that quietmalware() will return to where ctxsw() was suppose to return.

Bonus:
In Problem 3, the possibility exists that processes are blocked trying to send to a receiver, but the receiver terminates without reading all messages. That is, one or more processes are queued in the receiver's blocked sender queue when it undergoes termination. One approach is to dequeue such processes, insert them into XINU's ready list, and have bsend() return SYSERR. Describe a solution in Lab4Answers.pdf that follows this approach and is backward compatible with the solution of Problem 3. No need to implement the solution, but the description should be sufficiently detailed. That is, all kernel functions and data structures that are required to be modified should be listed and the changes needed specified.

Following the approach described above, when a process is terminated, we must loop through the blockedsenders queue until it is empty. This could potentially be done in userret. For each pid that is dequeued, we have to change the state back to PR_READY and insert the process into the readylist by calling insert. In order to make sure that bsend returns a SYSERR we can add another flag into the process table, prsndfailed. Just after the process is dequeued, we set this flag to true; furthermore, in bsend.c, when the call to resched() returns, we check this flag to make sure that it is not true. If it is, return SYSERR. In addition, we have to change the prsenderflag to false.