Brow1325
Connor Brown

3.1

a. Increase NQENT in include/queue.h to accommodate per-receiver process blocked sender queues. Explain in Lab4Answers.pdf why your change of NQENT works.
    i. We increase NQENT by an additional NPROC. We do this change because at most the number of blocked processes would be equal to NPROC.

3.3

a. Test your kernels mods for implementing bsend(). Describe in Lab4Answers.pdf what test scenarios you have considered to gauge correctness.
    i. All Test Scenarios use processes that run either sendMessage or receiveMessage.

```
pid32 receiver  = create(receiveMessage, 1000, 20, "receiver 1", 1, 7);
pid32 receiver2  = create(receiveMessage, 1000, 10, "receiver 2", 1, 7);

pid32 sender1   = create(sendMessage, 1000, 20, "sender 1", 2, receiver, '1');
pid32 sender2   = create(sendMessage, 1000, 20, "sender 2", 2, receiver, '2');
pid32 sender3   = create(sendMessage, 1000, 20, "sender 3", 2, receiver, '3');

pid32 sender4   = create(sendMessage, 1000, 20, "sender 4", 2, receiver2, '4');
pid32 sender5   = create(sendMessage, 1000, 20, "sender 5", 2, receiver2, '5');
pid32 sender6   = create(sendMessage, 1000, 20, "sender 6", 2, receiver2, '6');

resume(sender1);
resume(sender2);
resume(sender3);

resume(sender4);
resume(sender5);
resume(sender6);

resume(receiver);
resume(receiver2);
```

```
void sendMessage(pid32 rPID, char msg) {
        bsend(rPID, msg);
}

void receiveMessage(int number) {
        char msg = ' ';
        int  i = 0;

        for (i = 0; i < number; i++) {
                msg = receive();
                kprintf("Message received from sender %d: \"%c\"\n", i+1, msg);
        }
}
```

ii. Test scenario 1: 1 sender and 1 receiver
    1. Send “1” to the receiver.
    2. Expected output: Message received from sender 1: “1”
    3. Same output: YES
iii. Test scenario 2: 3 sender and 1 receiver
    1. Sender 1 sends “1”, Sender 2 sends “2”, Sender 3 sends “3”
    2. Expected output: Message received from sender 1: "1"
       Message received from sender 2: "2"
       Message received from sender 3: "3"
    3. Same output: Yes
iv. Test scenario 3: 2 sender and 2 receiver
    1. Sender 1 for receiver 1 sends “1”, and sender 1 for receiver 1 sends “2”
    2. Expected output: Message received from sender 1: "1"
       Message received from sender 1: "2"
    3. Same output: Yes
v. Test scenario 4: 6 sender and 2 receiver
    1. Sender 1 for receiver 1 sends “1”, Sender 2 for receiver 1 sends “2”, Sender 3 for receiver 1 sends “3” and Sender 1 for receiver 2 sends “4”, Sender 2 for receiver 2 sends “5”, Sender 3 for receiver 2 sends “6”
    2. Expected output: Message received from sender 1: "1"
       Message received from sender 2: "2"
       Message received from sender 3: "3"
       Message received from sender 1: "4"
       Message received from sender 2: "5"
       Message received from sender 3: "6"
    3. Same output: Yes

4.2

a. 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.
    i. To determine the address that we want to attack we first get the address of the process table for the victim. Then we get the address to the stack pointer. From there we get the address of the base pointer two entries higher. From the address of the base pointer we move one more value higher and then we have the pointer to the return address.

4.3

a. Explain in Lab4Answers.pdf your method for accomplishing the second and third goals.
b. Second, the code of quietmalware() finds out the address of the local variable x of victimA() and modifies its value from 5 to 9.

c. Third, after quietmalware() is finished it jumps to the instruction of resched() that ctxsw() would have returned to.
    i. To accomplish the second goal we start by creating two unsigned long * variables called myBP and myRet. We then use asm to move ebp onto myBP in order to get the base pointer of the process in the stack. From there we set another unsigned long * variable called myNewX equal to the address *myBP + 68 in order to get the address where the local variable x is stored. We then set the value stored in that address equal to 9.
    ii. To accomplish the third goal we set myRet equal to myBP + 1 in order to get the address where the return address is stored. We then set the address to a global variable called victimRet that we set to the victims return address in attackerB.c that causes it to return to the point in victimA.c right after it wakes.

Bonus

a. 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.
    i. Add an if check at the end to check if there is a bad PID. This will check that the receiver is terminated or not. If it is a bad PID then we do a while loop through the receivers prblockedsenders queue and dequeue the entry. We then change the senders state to PR_READY and call ready(). After we return SYSERR.