3. To test qreceive() and qsend(), I created receiver and sender processes in main that call qreceive() and qsend(). For all test cases, I always printed the contents of prrecvqueue and the messages returned at the end of qreceive() to make sure they matched up. I sent multiple messages to qreceive() and all the message values matched. I also changed the priorities of the processes calling qsend() and qreceive() and all the values matched up. Also, I had multiple senders send messages to the receiver and all the message values in the queue and the message values returned from qreceive() matched up. Lastly, I sent more than RECVQSIZE messages to the receiver and the messages after the fifth message were not part of prrecvqueue because qsend() returned SYSERR. For all test cases, the PID(s) of the sender were all correct.

4. I did not change the content of resched(). In qreceiveb(), however, I did create a temporary variable, temp, to store the value of senderpid before I call resched(). I then set senderpid equal to the value of temp to preserve the value of senderpid. It is possible that senderpid may get corrupted when resched() is called, so utilizing a temporary local variable will prevent that issue. For all test cases, I always printed the contents of prrecvqueue and the messages returned at the end of qreceiveb() to make sure they matched up. I sent multiple messages to qreceiveb() and all the message values matched. I also changed the priorities of the processes calling qsendb() and qreceiveb() and all the values matched up. Also, I had multiple senders send messages to the receiver and all the message values in the queue and the message values returned from qreceiveb() matched up. I sent more than RECVQSIZE messages to the receiver to ensure that the correct PID(s) ended up in prsenderblkid. Lastly, I sent more than 8 messages to the receiver to make sure qsendb() returns SYSERR, which it did, since prsenderblkid would be full.

5.1. In the stack, the value above the address of the newly created process, which is what we are editing, is the address of userret. We must examine the value of the stack directly above the newly created process to see if it is equal to the address of userret. By checking each of these values in wrongturnI() and confirming they are equal, we can verify that the process is newly created.

5.2. In the ctxsw() stack, we must locate the address of ebp (prstkptr + 9) and set it equal to another variable, x, for example. Although we obtain the address of ebp from the ctxsw() stack frame, it also points to a location in the resched() stack. By dereferencing x, it would now be equal to the value of ebp which we will set to another variable, temp. By then incrementing temp by one and dereferencing it, temp is now equal to the value of eip, the return address of resched(), in the resched() stack, which is what we want to modify.

5.3. There were no consequences of overwriting ctxsw()'s state information. The attack successfully caused the process to jump to the attacker's malwareI() function.