Internet device driver adv os final

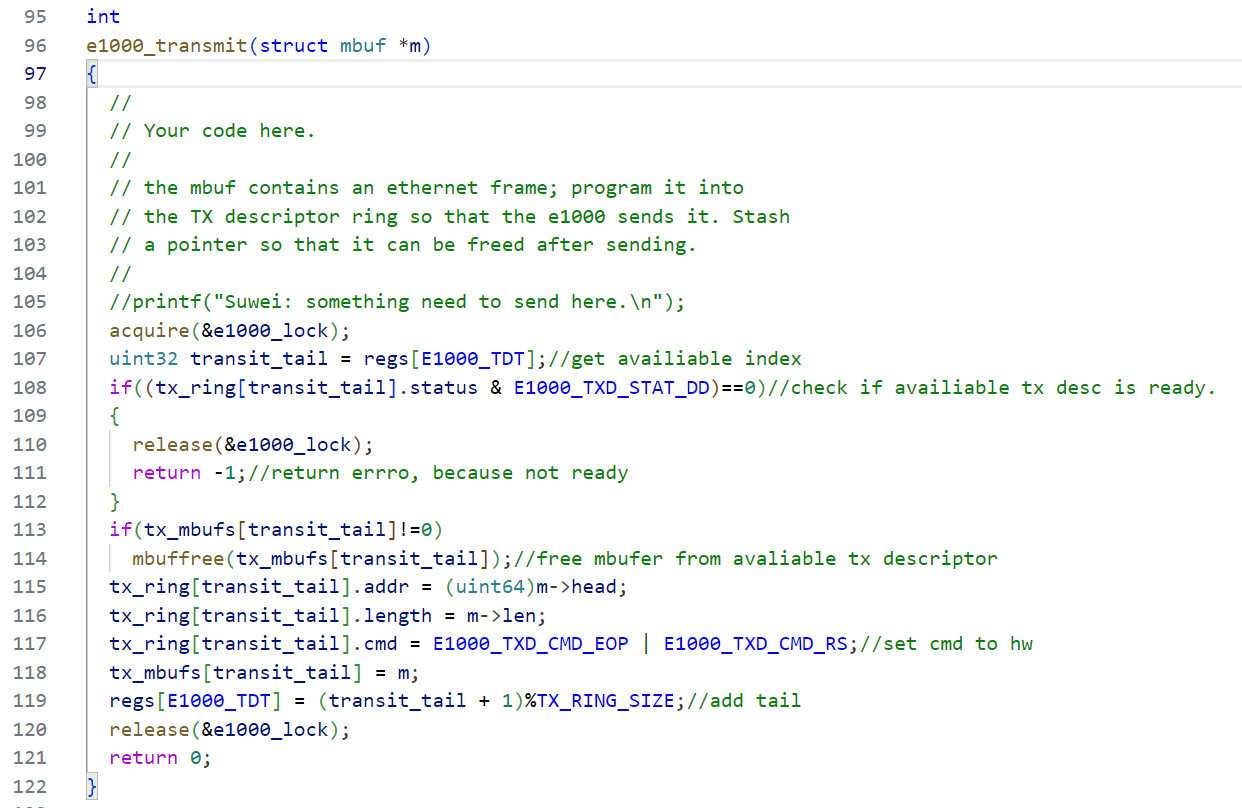
Student: Suwei Yang UID:u1429034

1. Explain the problem or question that the group explored in a paragraph or two;

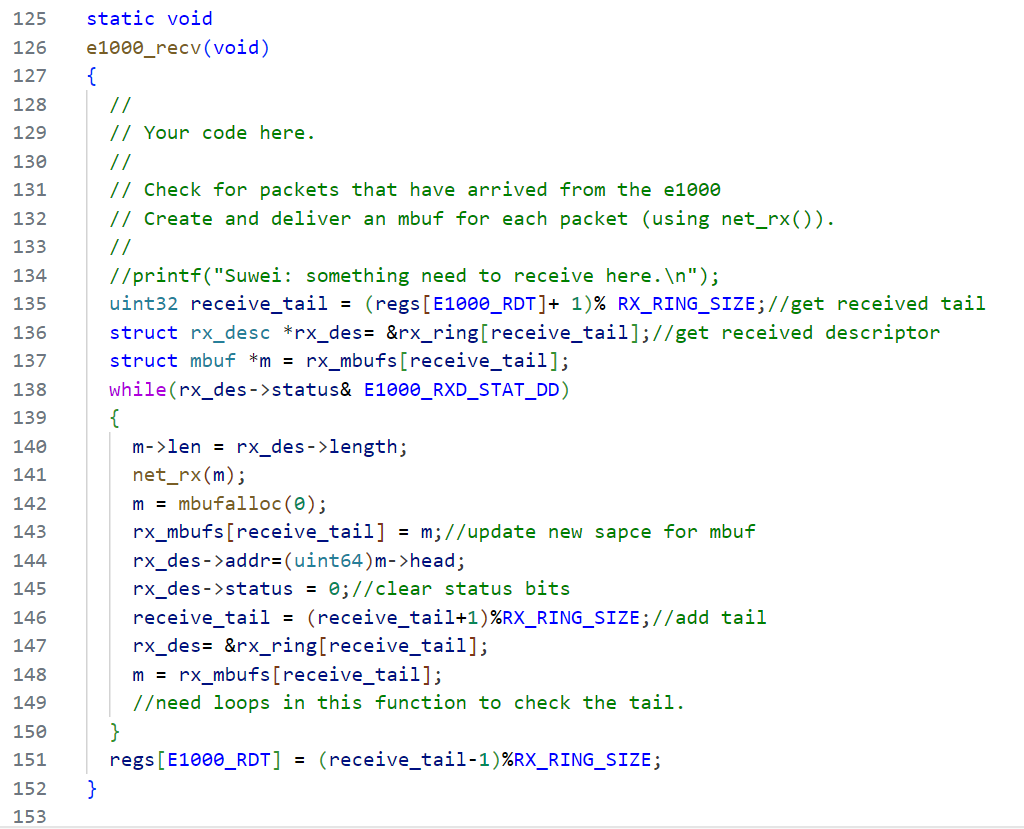
Before starting to work on this project, there are two things to do: read the E1000 manual and the driver code. From the manual, we can understand roughly the driver’s data structure, like descriptor, command, status…etc., and the driver’s transmitting and receiving flow.

Afterward, we have the basic knowledge to the code. We start from the driver code. There are two components to be finished, e1000\_transmit() and e1000\_recv(). We can complete them via the explanation on MIT website.

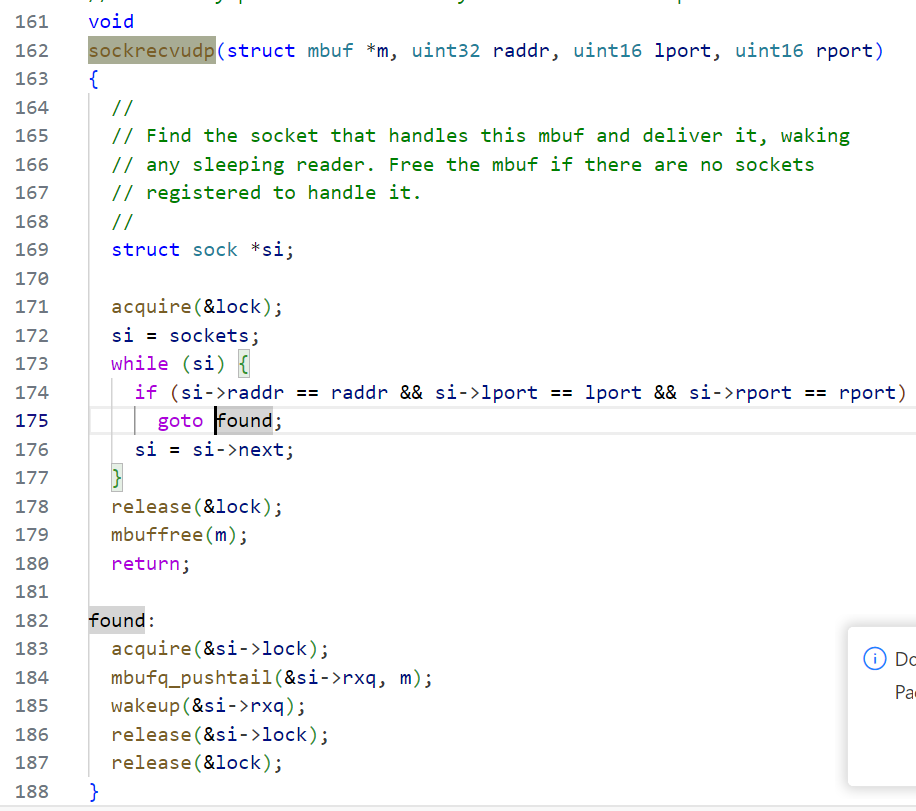
1. Explain the key aspects of the work you did;

To make the driver act, I need to finish two components left by MIT team: e1000\_transmit() and e1000\_recv(). They are used to set hardware before transmitting and after receiving message to/from server.

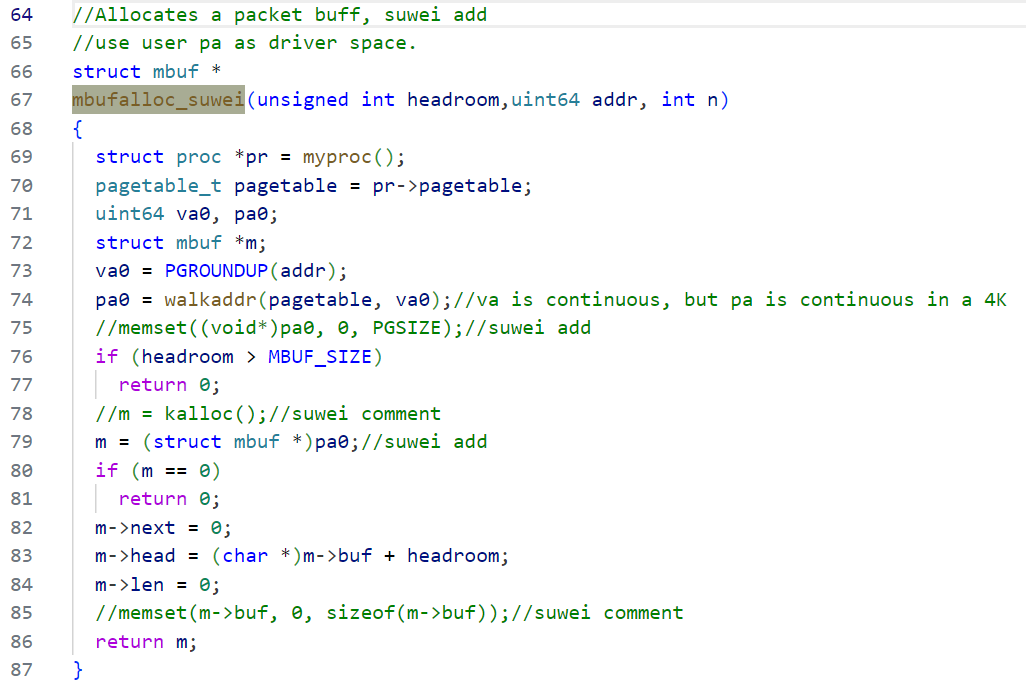
From e1000\_transmit(), we will get a buffer, mbuf, containing a couple of things: message length, content, IP, and port. We need to put information into corresponding registers and then issue a command afterward. Hardware will push the data to the server we specified. Finally, we need to increase the tail by 1 to tell the hardware the next available space for the next transmission.



In e1000\_rec(), we will need to handle the flow after receiving the message from server. Only two things that need to be finished: putting mbuf into net\_rx() and clearing space for following received package. In net\_rx(), driver will find a corresponding socket for received package, as shown below. Then, clear move can have a space for upcoming message.



After finishing the two components, the driver can pass the test script, and we can start doing some improvement. I found the they used copyin()/copyout() as the communication between user and kernel, which might be the bottleneck for the driver. Hence I created another function, mbufalloc\_suwei, to map mbuf into user memory space (shown below). Hoping to improve the performance by reducing memory movement during the kernel.

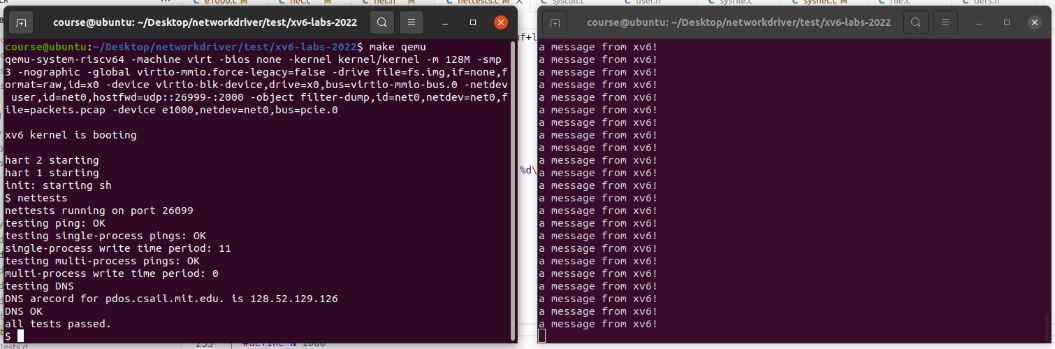
1. Explain what was interesting/exciting/promising about what you did;

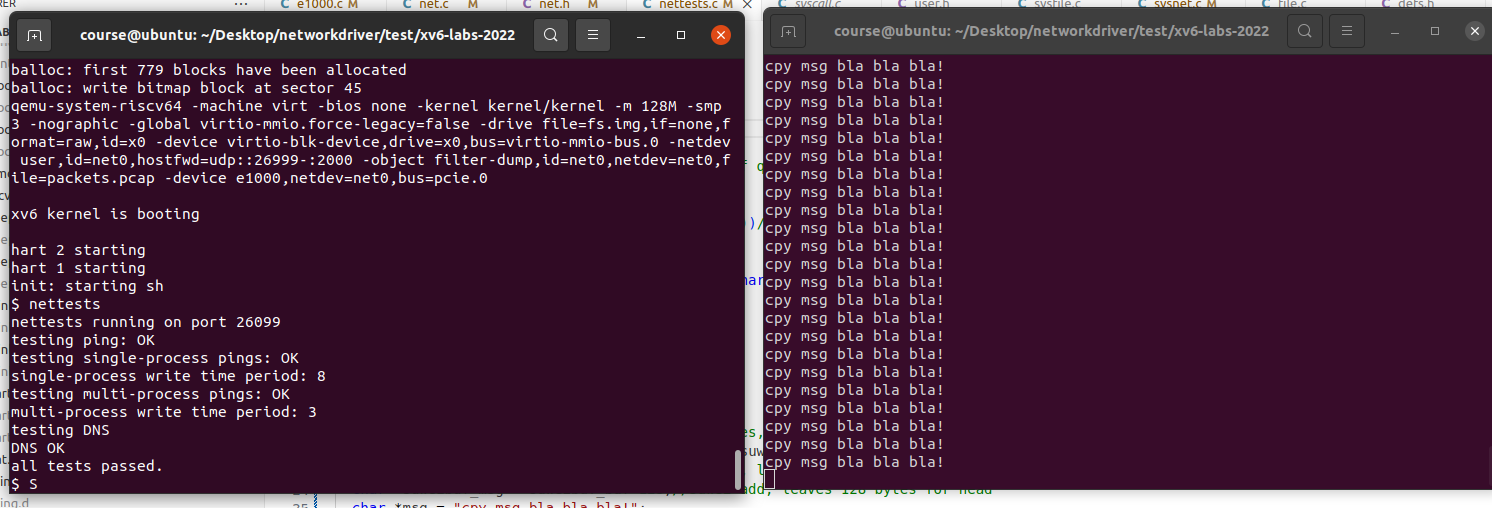
The most exciting part for me was when the driver sent package to server and when performance got different after I modified it. It still has some improved space like read flow. We can create buffer space for the message and map it directly to user space to further reduce the memory operation. However, it might take more time to implement and debug.

1. Explain what limitations you ran into in the work; what questions couldn't you answer; what prevented you from answering them;

From the result shown below, single process test improved after modification. I used a longer message to test afterward and it was 11ms(improved) v.s. 29ms(original). However, multi-process test elapsed time increased after the improvement. I didn’t figure out what they were waiting for. Especially when I increased the message length, they increased further. I guess it is probably related to the job in user space.

1. Result





|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Single process/ uptime() | T1 | T2 | T3 | T4 | T5 |
| Original | 11 | 12 | 13 | 15 | 14 |
| Improved | 8 | 9 | 15 | 11 | 15 |