- Which method for mutual exclusion provided the shortest wait time?

  The interrupt provides the shortest wait time
- Can you think of other ways to measure the performance of mutual exclusion methods? Explain.

  To let them finish a task with same number of processes and see how much time they need to finish them.

  Or compare the iteration num and print them out when at fixed time cost.

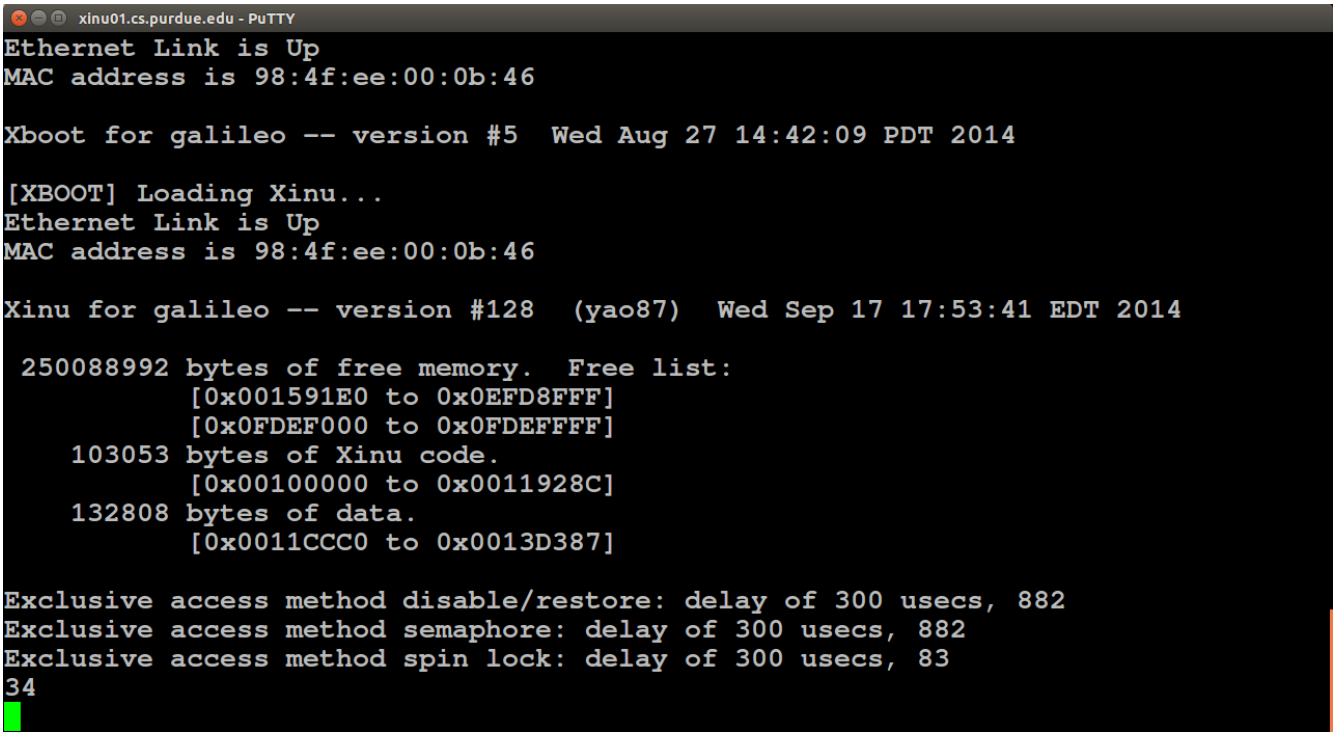
  Anyway I think this method is the best.
- Are there more methods for mutual exclusion that you can think of other than the three you tested? How would the performance compare to the three you tested?

  We can use message, send and receive which basically has the almost the same function in Xinu. We can send a message when every process is ready and receive to run the process, just like semaphores.

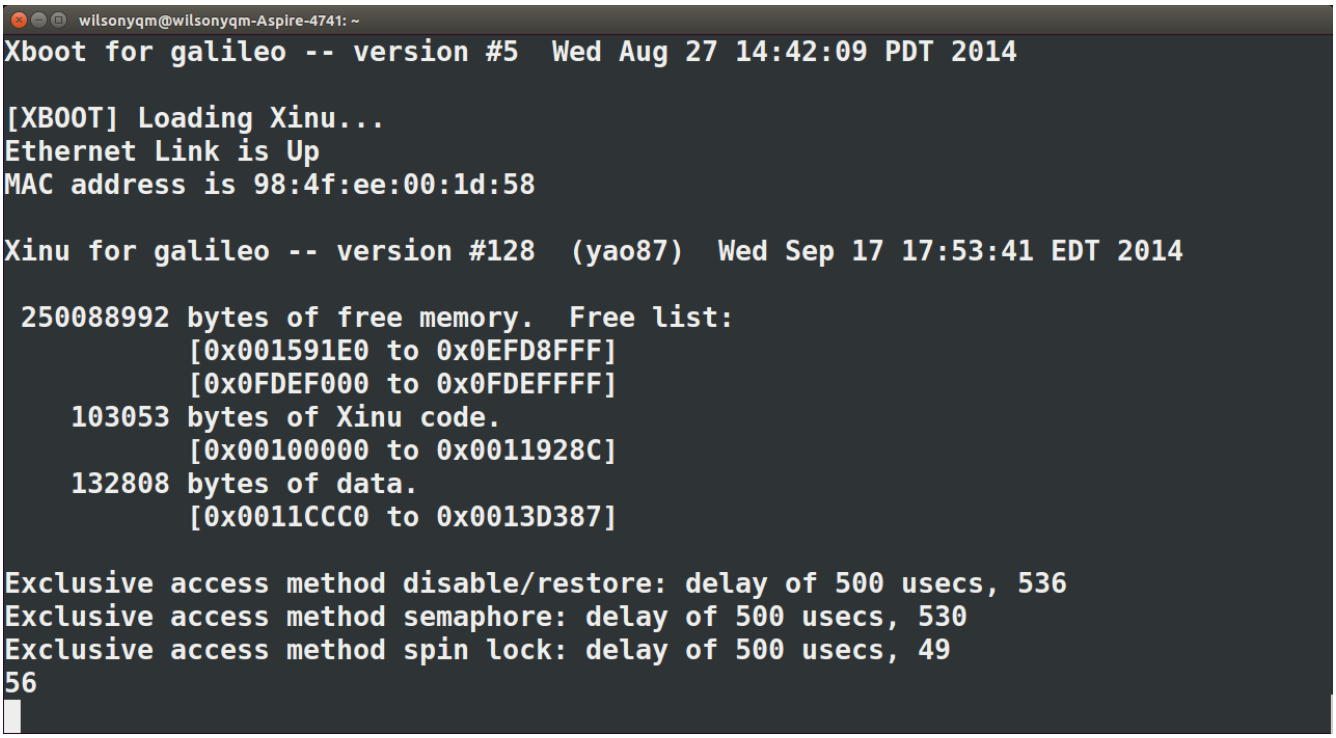
  I think this method would mostly like the performance of semaphore.

Extra Credits

How many milli-seconds are lost per second?



(34-30)/34*1000=118ms



26/56*1000 = 464ms

for 400us delay time it's (45-30)/45*1000 = 333 ms

Details about your implementation - how you managed to count number of lost milli-seconds per second.

I record the start time of the interrupt processes and end time of the them. Then I take the difference which record the time used on the interrupt process. Since the running time is 30 second and the difference time is 56 second. The interrupt loss time is 26 second. And the time loss per second is 26/56*1000 = 464 mili-seconds.

Why do you think your implementation gives the correct answer?

I recorded the real time cost in the outside of the function call and the execution time is 30 seconds, so I can briefly estimate the interrupt cost in these process.

Try different values of delay and try to get a relation between delay time and lost milli-seconds per second.

There should be a linear combination between these data.
118ms for 300
333ms for 400
464ms for 500
delay*2-480 = loss

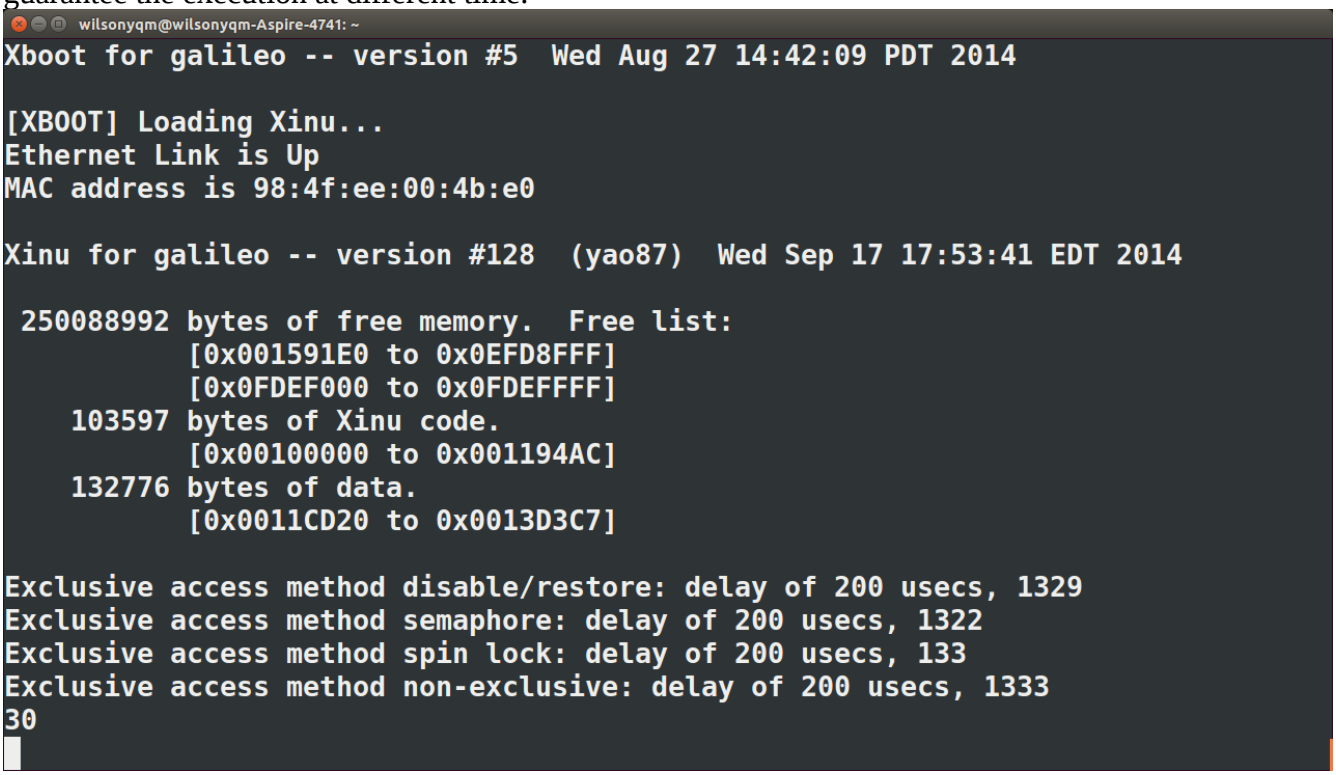
It's almost like this. But when delay time is smaller, the interrupt cost is much smaller than before and become less than 10ms.

2. In this lab implementation, we calculated the amount of useful work done by the processes and compared how fast each method of mutual exclusion is. But, to get the actual overhead, we must compare the useful computation rate with the maximum possible useful computation rate for the same period of time.
For example, if the iter_rate with no mutual exclusion is 100 and with mutual exclusion is 90, then the estimate of overhead is 100 * (1 - (90/100)) = 10% i.e. 10% of the total time is spent trying to gain exclusive access.
You have to write a code to measure the iter_rate when there is no mutual exclusion. NOTE: This does not mean, incrementing the global variable without mutual exclusion. You have to find how many iterations would be executed if mutual exclusion was not needed at all. [Hint: have each process increment it's own counter so they do not need mutual exclusion]
Calculate the overheads for each of the three mutual exclusion methods you have used in this lab.

I delay the resume time of every process.

```
for(n1=0; n1<10;n1++){
        resume(nodelay_prc[n1]);
        delayus(1);
        }
```

This is delay time is very small and it only delay at the very first period time, like an offset, which guarantee the execution at different time.



The overhead is (1333-1329)/1333 = 0.0023
(1333-1322)/1333 = 0.0083
(1333-133)/1333 = 0.9002
separately for interrupt semaphore and spin_lock.