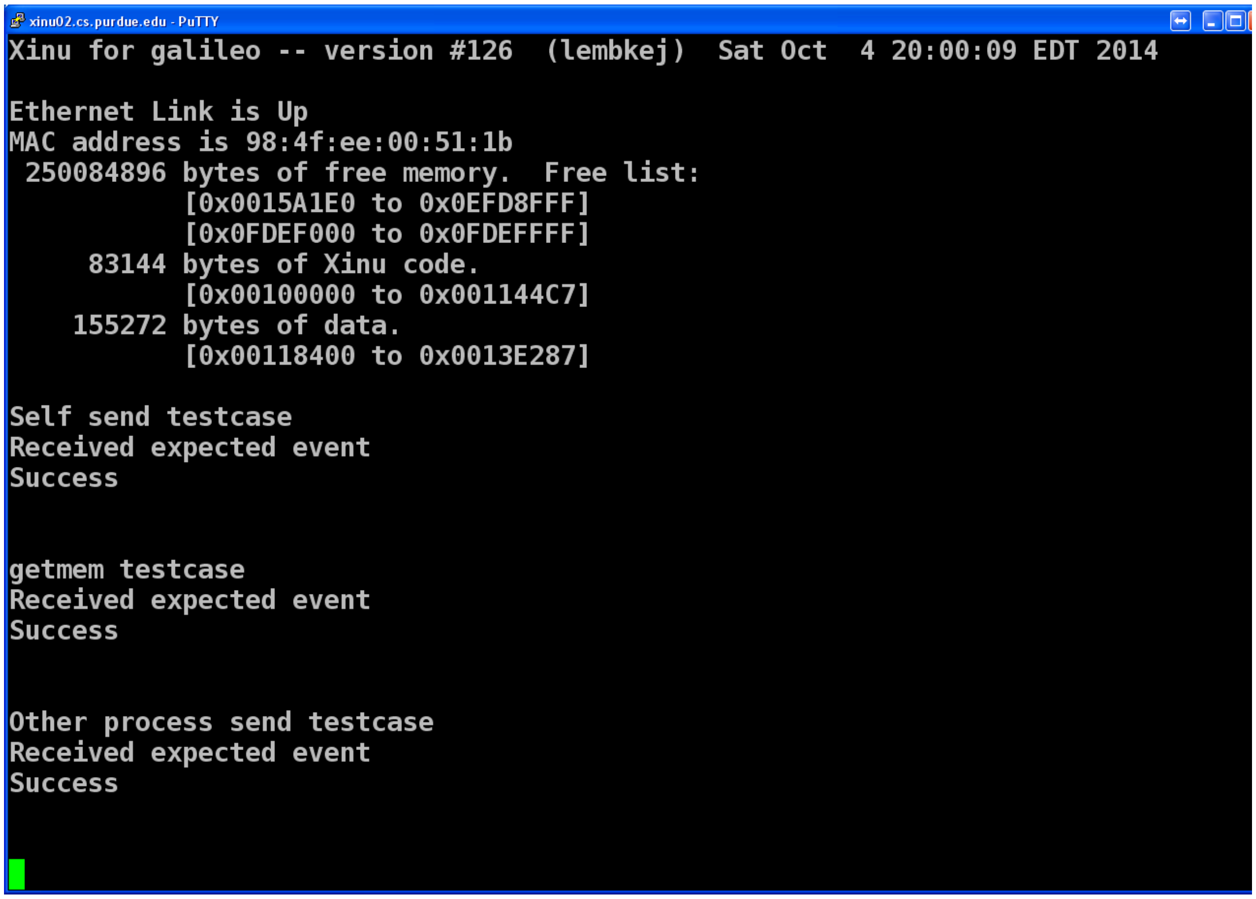
This is the final result



```
Xinu for galileo -- version #126  (lembkej)  Sat Oct  4 20:00:09 EDT 2014

Ethernet Link is Up
MAC address is 98:4f:ee:00:51:1b
 250084896 bytes of free memory.  Free list:
           [0x0015A1E0 to 0x0EFD8FFF]
           [0x0FDEF000 to 0x0FDEFFFF]
     83144 bytes of Xinu code.
           [0x00100000 to 0x001144C7]
    155272 bytes of data.
           [0x00118400 to 0x0013E287]

Self send testcase
Received expected event
Success


getmem testcase
Received expected event
Success


Other process send testcase
Received expected event
Success
```

-
- The details behind your implementation. As part of this discussion write answers to the following questions:
    - Describe your test cases. How to they ensure that your code correctly meets the requirements?

      I used the given test cases. First we have to tell the difference from system events and normal events. For system event the handler will directly call the registered event handler after registered, and for normal event we have to receive the event handler and call the event handler after sending the event.

      Second, we have to test the process send process event handler. We have to register an event in a process and create another process then switch to this new process to send an event to the main process which will help to call the event handler.

Third, we also need to test send event handler to the process itself. This is much easier, just register the event and send event to itself.

- Consider an additional requirement (you do not have to implement it) where the event handler must be run in the context of the process that registered it. How would the code have to modified to handle that case? What process context is executing when sendevent is called?

  I have already implemented it in the code for the system event. I have two different send functions: send_process_event, sendevent. Sendevent will call send_process_event with eventprocesspid sending the eventhandler to the eventprocess, if we need the event handler must be run in the context of the process that registered it, just call send_process_event directly.

- This lab only uses a couple operating system generated events. Come up with four (4) additional operating system events that you think would make sense to add (you do not need to implement these, just name them).

  We will need getstack and freestack, semcreate and signal semaphore.

  Since we already have wait semaphore, we still need semcreate and signal semaphore to work together with wait semaphore.

  Sometimes we will need to have a stack for memory allocation, so we need getstack event to create a stack in memory. It's also needed to free them.

  Also, we can have a basic create event with fixed size and parameters. It would be useful since we usually would like to create a tiny process to test some cases. Even though we don't know exactly the function would be, we can implement in the create process.

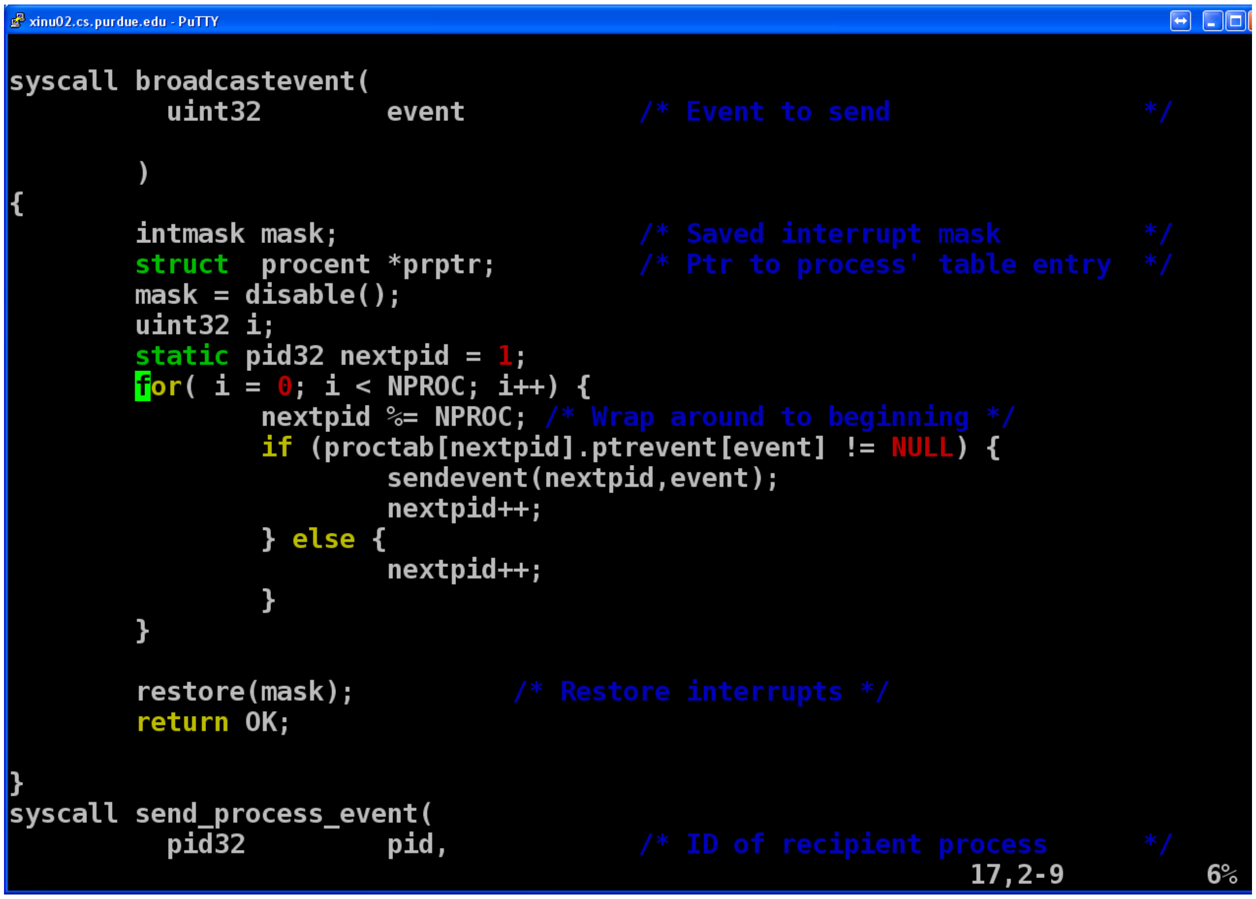
Extra Credit:

Add another system call with the following prototype:

```
        syscall broadcastevent(uint32 event);
```

Which sends an event to all processes that have an event handler registered for the input event. The order in which the event handlers are invoked is not important.

I just search all the process table. From 1 to NPROC. If the event table is not null it will call sendevent with this event and pid. Then continue to search until search all the NPROC process.



```
syscall broadcastevent(
          uint32        event           /* Event to send                */

        )
{
        intmask mask;                   /* Saved interrupt mask         */
        struct  procent *prptr;         /* Ptr to process' table entry  */
        mask = disable();
        uint32 i;
        static pid32 nextpid = 1;
        for( i = 0; i < NPROC; i++) {
                nextpid %= NPROC; /* Wrap around to beginning */
                if (proctab[nextpid].ptrevent[event] != NULL) {
                        sendevent(nextpid,event);
                        nextpid++;
                } else {
                        nextpid++;
                }
        }

        restore(mask);          /* Restore interrupts */
        return OK;

}
syscall send_process_event(
          pid32         pid,            /* ID of recipient process      */
                                                              17,2-9        6%
```