Programming Assignment Two: GeekOS
Preliminaries
CMSC 417 Fall 2010

1 Deadline
October 14, 2010. This assignment requires the understanding of potentially large amounts of existing code, recall of synchronization concepts presented in prerequisite courses, and setup of a development environment that permits you to run an emulator and debugger (e.g., on linuxlab).

Post general questions to the forum.

2 Objective
The goals of this project are:

- to build a mental model of the handling path of packets in an operating system
- to gain experience with the GeekOS system for implementing more advanced protocols

3 Executive Summary
You’ll fill in three TODOs in the geekos code in src/geekos/net/ethernet.c to enable the ethrecv and ethsend (message) commands to function.

The transmit path is mostly provided, except that the ethernet header must be filled in. The ethsend tool will send broadcast on eth0, since no destination mac address is provided.

The receive path for raw ethernet frames (those with type fields beyond the packet size) is mostly absent and will require some cognitive effort, since the receiving process (thread) must block waiting for an incoming frame. When a frame is received, the kernel must awaken the blocked process to receive it.

The mux.rb script can run two qemu instances and wire them together. Qemu is an emulator of an i386, much like vmware (but cheaper) and bochs (but better featured). It allows you to run an entire operating system in a window. If edited to set DebugInstance, mux.rb can cause one of the wired-up instances to wait to be run with the debugger.

Testing will be by running ethsend and ethrecv binaries.
4 Instructions

1. Determine whether you have qemu: run `which qemu`. If not, on a mac, `sudo port install qemu`. On a linux box that doesn’t suck, `sudo apt-get install qemu`. On a linux box that does, you’re on your own. On a windows box, putty to linuxlab or install vmware and knoppix.

2. Check out a copy of the geekos networking project source.

   `svn co --username=sv-geekosro https://svn.cs.umd.edu/repos/geekos/network`

   When it asks you for a password, hit return. This read-only subversion account lacks a password. Then, `cd network`.

3. Observe directories: `include`, for include files, `src`, for C source files, `scripts`, for some utilities, and `build`, where the Makefiles like and code is built.

4. To build, `cd build; make`. On a mac, you may need a cross compiler; you will be prompted to install. In the event of failure to build on linuxlab, send mail to TA and instructor. In the event of failure to build on your own machine, post to the forum.

5. To run, you may `make run`. This will start a qemu instance. Qemu developers, sadly, decide to change their argument syntax for each release. “-vga std” and “-std-vga” are alternatives, usually the wrong one is chosen. “-no-acpi” appears required, since the emulated network device conflicts with the acpi controller and qemu handles the conflict less than gracefully.

   A qemu instance started by `make run` will not communicate to anything, but the emulated network card should be present.

6. To run a pair of instances, run `ruby mux.rb`. They will be connected to each other via mux.rb, which will dump packet contents to the console. Modify this script as you see fit. To quit both qemu’s, Ctrl-C to mux.rb.

7. To debug one instance, edit mux.rb so that DebugInstance is set to (the integer) 1 or 2. Run mux.rb. Then, in another window, run `gdb geekos/kernel.exe`. Hit ‘c’, which will start the debug mode kernel, then Ctrl-C to interrupt. Type `where`, of course, to gather a stack trace.

8. In a qemu/geekos console, type `ethsend hello`. It should fail in a TODO macro. TODO macros start endless loops, which makes it a bit easy to find where you are in the debugger, if it is at all ambiguous.

9. (Start over.) In a qemu/geekos console, type `ethrecv`. It should fail in a TODO macro as well.

10. Complete the send path by editing `src/geekos/net/ethernet.c`. While running mux.rb, observe that `ethsend hello` attempts to transmit a frame.
11. Complete the receive path. `ethrecv` should print the contents of a received frame (i.e., “hello”) then exit.

5 Example Tests

`ethsend` and `ethrecv`, in combination with `mux.rb`, are your friends.

6 Overspecification

You may modify any files you see fit to modify, but `ethernet.c` should be the only one that needs modification.

7 Hints

Wait, Wake_Up, Mutex_Lock, Mutex_Unlock, list.h, `netbuf.h` must be understood before making significant progress. Useful to read include `net.c`, `ne2000.c`, `sys_net.c`.

`Malloc()` and `Free()` are the equivalents in the geekos kernel. If there is a memory leak, `Malloc` is apt to fail. If there is a memory error, `Free` is apt to throw a fault. Much of the standard library is not present; if you try to use a typical function (e.g., `sscanf`), the compiler may let it slide and it is only at the linker step that compilation will fail.

Ensure that header data are in network byte order. There’s only one field where it matters.

`Cond_Wait` and `Cond_Signal` may be useful.

You may modify `mux.rb` to send test traffic.

List elements may be on only one list (of a given type) at a time. There is only one set of pointers to next and previous elements.

8 To Turnin

Run `make submit`.

Your code MUST RUN on the submit server. You may have to test the compilation on linuxlab (which has a similar setup). Errors can occur. You may need more `#include` lines than you expect or they must be in the order on the man page. It is not sufficient that your code work on some machine you happen to know about. IF your code runs on linuxlab but not on submit, we’ll grant you a pass on this requirement... at least until we can figure out the mismatch.

Again, linuxlab’s compiler may be more picky than yours.
9 Finding Code On-line

If you find this online, don’t copy it. Cite sources in the top of your file.