You'll create a socket interface. We'll provide UDP test client/server applications. UDP clients and servers should now be able to connect with the UDP layer—separate processes! The interface will be implemented through UNIX domain sockets, but the calls will look like standard UDP socket calls.

Socket
Bind
Sendto
Recvfrom
Close

These will have standard args, but the implementation will be through UNIX domain sockets. See the textbook, web page demos, etc.

You'll need to set up something like pcbs.

We'll provide a sample UDP client and server so you can test your socket layer.

For this lab, thinking things through is vital—see the FAQ. This should do a reasonable job of mimicking real-life. For example, UDP cannot pass along data willy-nilly to the application—it must hold it until the application asks for it.

```c
mysocket = Socket(PF_INET, SOCK_DGRAM, 0) in your application
mysocket = socket(PF_INET, SOCK_DGRAM, 0) in real life.
```

Buffering!!!! NOTE: you CANNOT use the real system's buffering to do your work for you—i.e. Your UDP cannot just dump packets into the real system's buffers and then let the application grab them—your UDP must do the buffering and pass along a packet only when the application asks for it. Typically, this can be implemented via your socket layer—when the application does a Recvfrom, the socket layer will send a request to UDP, etc.

Finally! Remember where the socket code is going. Is it part of the kernel? Or do you need a #include in your applications, etc?