**Introduction to Socket Programming**

This assignment introduces you to basic socket programming. The goal is to get you comfortable with (a)Unix socket programming and (b) simple client-server interaction.

You are required to submit code, which compiles and runs on the servers Jason and Volta. Both programs should be written in C. You might want to find out about the architecture and OS of these two servers to facilitate programming.

**Note: You will be able to find copies of this program on many places on the web, as well as in some form in the textbook. We strongly recommend that you implement this program without consulting other references, as you will gain a much better understanding of these system calls.**

We want you to write code to enable a machine M1 to receive text messages from a machine M2 over TCP sockets. M1 should print these text messages on the standard output. M1 should then reverse the string and send it back to M2. M1 should be listening for text messages on a port known to M2. M2 should take a text message from stdin – transmit the message, receive the reverse message, print the reversed message on standard output and then exit.

You can assume that the client is run as

$ ./client server-IP-address port-number

where "server-IP-address" is the IP address of the server, and "port-number" is the TCP port the server listens on. The server is run as "server port-number". If the server cannot bind on a port, print a message to standard error.

You should use fread and fwrite calls for reading/writing data to and from sockets and files. Do not use special "string" versions of these calls (e.g., fgets and fputs as they are not designed for binary data).

Make sure you handle the following correctly:

1. **Local and remote operation**: Your program should be able to operate when connection over both localhost (127.0.0.1) or to between machines. When testing, make sure you use an appropriate port to avoid firewall in the lab cluster. You can get the machine's routing IP address via ifconfig (in Linux/Mac) or your GUI Network Preferences. Note online websites tools like http://whatismyipaddress.com/ may **not** work because your machine may be behind a [NAT](http://en.wikipedia.org/wiki/Network_address_translation) and use a different IP address for wide-area communication than from communication between the client and server in the local cluster.
2. **Buffer management**: Assume that the file contents can be arbitrarily large (assume a typical size of 20KB), but the buffers you use to read/write to the file or socket must be small and of fixed size (e.g., 4096 bytes).
3. **Handling return values**: By default, sockets are blocking, and for this assignment we will use only blocking sockets. "Blocking" means that when we issue a socket call that cannot be done immediately (including not being able to read or write), our process is waits until it can perform the action. This includes the case when
   * a socket's internal buffer is full and therefore, no data can be written, or
   * a socket's buffer is empty, and no data is available to be read.

However, if there is some data available to be read or some can be written, the call will return the number of bytes read or written respectively. NOTE: This returned value can be less than specified in the length argument to the call or indicate an error. You must handle this.