60-256 Laboratory - 9
Winter 2013

Attendance = 2 point
Completion = 8 points
Due date: To be completed by the end of the lab period.

Sockets:

In layman's term, a Socket is an end point of communication between two systems on a network. To be a bit precise, a socket is a combination of IP address and port on one system. On each system a socket exists for a process interacting with the socket on other system over the network. A combination of local socket and the socket at the remote system is also known a 'Four tuple' or '4-tuple'. Each connection between two processes running at different systems can be uniquely identified through their 4-tuple.

In this Lab you will create a **server** that continuously runs and sends the date and time as soon as a **client** connects to it. The server program (**NetServer.c**) is provided for you to compile and run.

A partially completed client program (**NetClient.c**) has also been provided, with the indications of the codes you have write. (Look for "**TODO**" flag). Complete the client program, compile it and the run it while the server is running.

Note that the client needs to know the IP address and port number of the server. When both the applications are running in the same machine, you can use the loop back IP address 127.0.0.1. We have used port number 7777 for the server to be used by the client as well.

NOTICE the following points for the server application:

- The call to the function 'socket()' creates an UN-named socket inside the kernel and returns an integer known as socket descriptor.
- This function takes domain/family as its first argument. For Internet family of IPv4 addresses we use AF_INET.
- The second argument 'SOCK_STREAM' specifies that the transport layer protocol that we want should be reliable i.e. it should have acknowledgement techniques. For example : TCP
- The third argument is generally left zero to let the kernel decide the default protocol to use for this connection. For connection oriented reliable connections, the default protocol used is TCP.

- The call to the function 'bind()' assigns the details specified in the structure 'serv_addr' to the socket created in the step above. The details include the family/domain, the interface to listen on (in case the system has multiple interfaces to network) and the port on which the server will wait for the client requests to come.

- The call to the function 'listen()' with second argument as '10' specifies maximum number of client connections that server will queue for this listening socket.

- After the call to listen(), this socket becomes a fully functional listening socket.

- In the call to accept(), the server is put to sleep and for an incoming client request, the function accept() wakes up and returns the socket descriptor representing the client socket.

- The call to accept() is run in an infinite loop so that the server is always running and the delay or sleep of 1 sec ensures that this server does not eat up all of your CPU processing.

- As soon as server gets a request from client, it prepares the date and time and writes on the client socket through the descriptor returned by accept().

Your client application should follow the following points:

- A socket should be created through call to socket() function.

- Information like IP address of the remote host and its port is bundled up in a structure and a call to function connect() is made which tries to connect this socket with the socket (IP address and port) of the remote host.

- You do not have to bind your client socket on a particular port as client generally use port assigned by kernel as client can have its socket associated with any port. But in case of server it has to be a well-known socket. Well-known servers bind to a specific port, like HTTP server runs on port 80 etc. There is no such restriction on clients.

- Once the sockets are connected, the server sends the data (date + time) on your clients socket through clients socket descriptor and your client should read it through normal read call on the its socket descriptor.