Assignment 1
CS 425A: Computer Networks
2016-2017-- Semester I

Computer Science and Engineering Department

Indian Institute of Technology Kanpur

Due Date: Aug 9, 2016 11:55 pm via Moodle

Acknowledgement: This project is based on a homework used at Princeton University Computer Science Program

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.

Choice for programming language:C or C++.

Note: You may 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, which you will heavily using in upcoming assignments.

You will be implementing a simple Client/Server program. You have to first implement the server as a separate program, and then the client as another program. They will network communication and your job is to design this networked client/server application correctly using socket APIs.

First, you will write the **Server** program. The server will respond to a request by the client for the content of text file, by sending the content to the client. Server will be designed to be listening on a certain port waiting for connection requests from client. When a client connects, the server will send a welcome message along with a prompt for a file name. The client allows the user to type in the file name, and then sends the file name as a message to the server. If the client sends a valid file name (i.e., a file that exists in the current directory from which the server program is running), the server will transmit the content of text file to the client, which the client will be printing on standard output. If the file name is not valid, the server must send a message back to the client that 'no such file exists'. The client will then again prompt the user for another file name, and repeat. If the server sends the content of the file, even after that the client will again come back to ask for another file name from the user and repeat this process until the process is killed.

Similarly the server will also run in a loop, and after each file transfer or sending message about nonexistence of the file requested, the server should go back to listening on the socket for further file name from the same client. However, you do not need to make the server accept more connections unless the client closes the connection. If the client closes the connection, the server should go back to listening for further connection requests.

To emphasize, both the client and server are in infinite loop unless the respective processes are killed.

It's good if you handle the timeout cases where a client does not respond for a long time (but not necessary for this assignment). Server should handle the client connections sequentially and accept connections from multiple clients. After servicing one client to completion, it should proceed to the next. If multiple clients try to simultaneously connect to the server, the server should handle them one at a time (in any order). Serving one client at a time is enough.

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 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
    - socket's internal buffer is full and therefore, no data can be written, or
    - 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.

Deliverables and Due Dates:

The deliverable for this assignment is a tar file contain a directory with a client.c and server.c file along with other .h files, and a Makefile. Your makefile must have a target to build the two programs called client and server.

A Readme file should describe what other files are used in case you use any of your own .h file, or .c file etc.

Makefile is a must.Before creating the tar file, please do a make clean to remove all executables and binaries from your directory.

Grading: 40 points -- correct implementation of Client
40 points -- correct implementation of Server
10 points for structured coding and comments in the code
10 points for correct Makefile

Please note that this assignment is not a Project but an assignment which will count as 4% of your final score.

Note that
you will have a penalty of 10 points if you submit between Aug 9 11:56 pm -- Aug 10, 11:55 pm.
If you submit between Aug 10, 11:56 pm till Aug 11, 11:55 pm, you will get a penalty of 30 points.
If you submit between Aug 11, 11:56 pm till Aug 12, 11:55 pm, you will get a penalty of 50 points.
If you submit between Aug 12, 11:56 pm till Aug 13, 11:55 pm, you will get a penalty of 75 points.
At 11:55 pm on Aug 13, the submission will be automatically closed.