CSCI 415 Computer Networks
Homework 1
Due 02/04/08

Saad Mneimneh
Computer Science
Hunter College

Problem 1
Listen to the first movement of Beethoven's fifth symphony.

Problem 2
Write the following sentence in Morse code:

"I am in the CSCI 415 class."

Problem 3
Compile the following server and client C++ code and run it. You will be asked to modify the code in a later homework.

server.c

```
#include <iostream>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>

using std::cout;
using std::cin;
const unsigned short int port = 5432;
const int max_pending = 10;
const int max_len = 256;

int main() {

  sockaddr_in address; //address
  sockaddr_in client_address; //client address
  char message[max_len];

  int s;
  int new_s;
  int len;
  //build address
  memset(&address, 0, sizeof(address));
  address.sin_family = AF_INET;
  address.sin_addr.s_addr = htonl(INADDR_ANY);
  address.sin_port = htons(port);

  //setup passive open
  if ((s=socket(PF_INET, SOCK_STREAM, 0)) < 0) {
    cout<<"error in socket";
    return 0;
  }

  //bind socket to address
  if (bind(s, (sockaddr *)&address, sizeof(address)) < 0) {
    cout<<"error in bind";
    return 0;
  }

  if (listen(s, max_pending) < 0) {
    cout<<"error in listen";
    return 0;
  }

  //wait for connection, then receive message
  socklen_t size = sizeof(sockaddr_in);
  while (1) {
    if ((new_s = accept(s, (sockaddr *)&client_address, &size)) < 0) {
      cout<<"error in accept";
      return 0;
    }

    while (len = recv(new_s, message, sizeof(message), 0)) {
      cout<<message<<"\n";
    }
    close(new_s);
  }
}
```

client.c

```
#include <iostream>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>

using std::cout;
using std::cin;
const unsigned short int port = 5432;

int main() {
  int s;
  sockaddr_in address; //address to connect to

  memset(&address, 0, sizeof(address));
  address.sin_family = AF_INET;
  address.sin_port = htons(port);
  address.sin_addr.s_addr = htonl(INADDR_ANY); //my IP address

  //active open
  if ((s=socket(PF_INET, SOCK_STREAM, 0)) < 0) {
    cout<<"error in socket";
    return 0;
  }

  //connect
  if (connect(s, (sockaddr *)&address, sizeof(address)) < 0) {
    cout<<"error in connect";
    close(s);
    return 0;
  }

  char message[256];
  while(cin.getline(message, 256, '\n')) {
    if (strlen(message) == 0)
      break;
    send(s,message,strlen(message)+1,0);
  }
  close(s);
  return 0;
}
```

Start one server and one client in separate windows. Start the server first; otherwise, the client will not be able to connect to the server and will report an error and exit. The client should accept messages from the keyboard (you) and sends them to the server. The server just echoes what you send. To exit the client just type an empty message (hit Enter).

Notes:

If you are using Windows, you only need to include the following header files:

```
#include <iostream>
#include <winsock.h>
```

Also, you need the following Windows specific initialization of sockets in both the server and the client:

```
WORD wVersionRequested;
WSADATA wsaData;
wVersionRequested = MAKEWORD( 1, 1 );
WSAStartup( wVersionRequested, &wsaData );
```

Also replace this line in the client:

```
address.sin_addr.s_addr = htonl(INADDR_ANY); //my IP address
```

by the following line:

```
address.sin_addr.s_addr = inet_addr("127.0.0.1");
//127.0.0.1 designates the local host
```

Problem 4

Calculate the total time required to transfer a 1000 KB file in the following cases assuming an RTT of 100 ms, a packet size of 1 KB, and an initial 2xRTT of “handshaking” before data is sent:

(a) The bandwidth is 1.5 Mbps, and data packets can be sent continuously.

(b) The bandwidth is 1.5 Mbps, but after we finish sending each data packet we must wait for one RTT before sending the next.

(c) The bandwidth is “infinite”, meaning that we take transmit time to be zero, and up to 20 packets can be sent per RTT.

(d) The bandwidth is infinite, and during the first RTT we can send one packet (2^0), during the second RTT we can send two packets (2^1), during the third RTT we can send four packets (2^2), and so on. (We will look at this technique later on.)