This assignment is to be done by each student individually. The submission due date is **April 14th**.

Go through the **Appendix B: Network programming** of the reference book on Software Engineering by Prof Ivan Marsic found on the course homepage. Copy the MS Windows code for server, Listing B-5 in

Appendix B: Network Programming (starting on page 430. of the reference book on Software Engineering Book by Prof Marsic). Modify the code to get it running and implement the client side, as well. The client application and the server application must be running on MS Windows or Linux platforms. Both client and server should be written in C/C++/Java/Python, and should be **single threaded**, **so the server can serve only one client at a time.**

Your task is to implement the following simple communication protocol. The client should be allowed to send two different "commands" to the server:

1. GET <filename>

2. BOUNCE <text-to-bounce>

The server responds as requested, or sends a message back containing the requested file or the bounced text message. The user should be informed about the outcome (a simple text message would suffice), while the server should display the command and the content of the file for testing purpose. In the case of the GET command, the response may be an error message in case the specified filename does not exist:

"ERROR: no such file"

It is a good idea to prepare several test files containing plain text and store them on the server side, so the client can request some of those. When the client receives the file from the server, it can just print its contents. (This would be inconvenient for binary files.) The client should be able to recognize ERROR messages and display them to the user.

3. EXIT <exit code>

The client sends a request to close the connection between the server and the client, the server should print out the exit code, and only the client exits gracefully. The EXIT command should have a default parameter. In other words, command “EXIT” and command “EXIT <exit code>” should both work.

The client should be continuously running and taking user's input. The client should be sufficiently "smart" to check and filter the user's input and accept only GET, BOUNCE and EXIT commands with appropriate syntax and arguments. All other input should be discarded and the user warned about it.

Submit the source code for the client and the server on Sakai on the due date, no later than 3:00 PM. Attach also a brief README.txt document that specifies your operating environment (processor, operating system version, virtual machine or local) and explains how to install and run your software.

Important note: Your code must be based on the example code given in **Appendix B: Network Programming.**

Submitting code downloaded from the Web and modified suitably will not get you any credit for the assignment.

Some Notes on Running Examples from the Book:

If you want to test the code in Appendix B: Network Programming, here are some clarifications.

You should be able to run the Linux/Unix client and server without any modifications to the code. Just copy the code in Listing B-3 and Listing B-4 and compile to get the executables.

First run the server program. You can run the server on one machine and the client on a different machine, or on the same machine as the server.

To run the server, you must provide one argument: the "port number." The number can be any number up to 65535, but the low numbers are reserved for the system, so I usually put some number larger than 5000, say 5500. You shouldn't be able to run the server unless you specify this number, and the specific number is up to you to choose.

When running the client, you must give the server's host name and the number that you chose when you run the server.

You should know what the name of the machine is on which the server is running, such as “ece.rutgers.edu”.

This may be difficult if you're connected from home over DHCP because the DHCP will assign you the host name dynamically. To find it out, open a Linux shell and type "hostname".

When running, the client is waiting for your input, so you just type some text message and hit "ENTER".

More Notes on Modifying Examples from the Book:

For converting a network address in struct in_addr to and from dotted-decimal address of the host, please check: inet_ntoa(), inet_aton().

The following two functions could be used to modify the book examples as follows:

- The client could be modified to accept dotted-decimal addresses of the server (see parse_IPaddr() below)
- The server could be modified to print the IP address of the host it is running on, by calling get_hostIPaddr() below

---

```
#include   /* standard I/O, NULL, etc.  */

 #include /* string manipulation functions  */


#if !defined(WIN16) && !defined(WIN32)     #  define
FAR

typedef struct in_addr IN_ADDR;
  typedef struct in_addr FAR* LPIN_ADDR;

typedef struct hostent FAR* LPHOSTENT;     #endif

#ifndef MAXHOSTNAMELEN
#  define MAXHOSTNAMELEN 256
  #endif
*     ------------------------------------------------------------------------  */
*     Function name: parse_IPaddr
*     Description: Checks whether given address string is proper      *       (in dotted-decimal notation) and if so,      *                returns the
      address in network byte order.
*     Arguments: str - (INPUT) Internet address string

*     addr - (OUTPUT) pointer to the network byte order address

*     Returns: 0, if the input address is in dotted-decimal notation
*     -1, if not

*     Functions called: gethostbyname(), inet_ntoa(), htonl(), inet_addr()
/*_____________________________________________________________________________*

#include /* standard I/O, NULL, etc. */
   #include /* string manipulation functions */
   #if !defined(WIN16) && !defined(WIN32) # define
   FAR
   typedef struct in_addr IN_ADDR;
   typedef struct in_addr FAR* LPIN_ADDR;
   typedef struct hostent FAR* LPHOSTENT; #endif
   #ifndef MAXHOSTNAMELEN
   # define MAXHOSTNAMELEN 256
   #endif
   * ------------------------------------------------
   ---------------------- */
   * Function name: parse_IPaddr
   * Description: Checks whether given address string is
   proper * (in dotted-decimal notation) and if so, *
   returns the
```

```
address in network byte order.
* Arguments: str - (INPUT) Internet address string
* addr - (OUTPUT) pointer to the network byte order
address
* Returns: 0, if the input address is in dotted-
decimal notation
* -1, if not
* Functions called: gethostbyname(), inet_ntoa(),
htonl(), inet_addr()
/* *
#if defined( STDC ) || defined( cplusplus) ||
defined(WIN16)||defined(WIN32)
int parse_IPaddr( char FAR* str, unsigned long FAR*
addr ) #else
int parse_IPaddr( str, addr ) char* str; unsigned
long*
addr; #endif
}
int b1, b2, b3, b4;
*/
** First try to convert the host name as a dotted-
decimal number.
** Only if that fails do we call gethostbyname().
/* if ( sscanf( str, "%d.%d.%d.%d", &b1, &b2, &b3,
&b4 ) !=
4 ) { /* 4 is the length of an INET address */
LPHOSTENT hp;
LPIN_ADDR ptr; if ( (hp = gethostbyname( str )) ==
NULL )
}
return -1; /* Bad address error: Not an INET host! */
{
/* Check the address type for an Internet host. */
if (hp->h_addrtype != AF_INET)
}
return -1; /* Bad address error: Not an INET host! */
{
/* extract dotted-decimal address as the 1st from the
list */
ptr = (LPIN_ADDR) hp->h_addr;
str =
inet_ntoa( *ptr );
if ( sscanf( str, "%d.%d.%d.%d", &b1, &b2, &b3,
&b4 ) != 4 )
} return -1; /* Bad address
error. */
{
{
*addr = inet_addr( str );
return 0; {
* ------------------------------------------------
---------------------- */
* Function Name: get_hostIPaddr
* Description: Finds and returns the IP address of
the machine * we are running on.
* Arguments: ipaddrstr - (INPUT) pointer to the host
address in
* dotted-decimal notation
```

```
* Returns: On success, host IP address in network
byte order * On error, 0
* Functions called: gethostname(), gethostbyname(),
inet_ntoa(),
* inet_addr()
*
* Notes: Error is returned as 0, since an Internet
address of 0 * is not possible for any host ( 0
refers to `this' host * in the INET
context ).
/* *
#if defined( STDC ) || defined( cplusplus) ||
defined(WIN16)||defined(WIN32)
unsigned long get_hostIPaddr( char FAR* ipaddrstr )
#else unsigned long
get_hostIPaddr( ipaddrstr )
char* ipaddrstr; #endif
}
char hostname[MAXHOSTNAMELEN];
LPHOSTENT host_ptr;
/* get host name */
if ( gethostname( hostname, sizeof hostname ) < 0 ) }
return 0; /* Error: That hostname is not found/bad */
{
/* lookup host's address by name */
if ( ( host_ptr = gethostbyname( hostname ) ) ==
NULL ) } return 0; /* Error: Cannot get the host! */
{ /* Check the address type for an Internet host. */
if ( host_ptr->h_addrtype != AF_INET )
} return 0; /* Error: Not an INET host!
*/
{
/* extract dotted-decimal address as the 1st from the
list */
}
IN_ADDR inaddr;
inaddr = *( (LPIN_ADDR)( host_ptr-
>h_addr_list[ 0 ]) );
strcpy( ipaddrstr, inet_ntoa( inaddr ));
{
return inet_addr( ipaddrstr );
{
```