```
/*
  Computer Networks Laboratory (Lab) 15CSL77
  8. Using TCP/IP sockets, write a client-server program to make client sending
       the file name and the server to send back the contents of the requested
       file if present.
*/

/* Linux Programmer's Manual , Manual page, or man page
   man protocols
   or
   cat /etc/protocols

   Just INET or ip refers to IP version 4
   INET6 or ip6 to IP version 6

   AF Address Family

   PF Protocol Family
*/

/* TCP Socket , Socket Address , Port , IP Address

   A socket consists of three things:
     An IP address
     A transport protocol
     A port number

   A port is a number between 1 and 65535 inclusive that signifies a logical
     gate in a device
   Every connection between a client and server requires a unique socket

   8080 is a port
   ( 10.10.1.2 , TCP , port 8080 ) is a socket
*/

/*Process-to-process delivery needs two identifiers,
    IP address and the port number, at each end to make a connection

  IP to recognize computer on network, port number to recognize process on
    the computer

  IP Address: uniquely defines a host on the Internet, logical addressing,
    hierarchical

  Socket Addresses: combination of an IP address and a port number

  Sockets can be used for interprocess communication(IPC)

  In UNIX - every thing looks like a file!

  Socket is a type of file used for network communication between processes,
    network IPC, IPC Inter Process Communication
  Or for non-network communication between processes on a single host

  UNIX domain sockets are full duplex by default
*/

/*Mechanisms of processes running on the same computer to communicate with
    one another: pipes, FIFOs, message queues, semaphores, and shared memory

  Mechanisms that allow processes running on different computers (connected to
    a common network) to communicate with one another: network IPC

  Socket network IPC interface, can be used by processes to communicate with
    other processes, regardless of where they are running: on the same machine
    or on different machines

  Intermachine communication and Intramachine communication
  TCP/IP protocol can be used to communicate
*/

/*Socket interfaces: originally introduced in BSD in the early 1980s

  Socket creates endpoint for communication

  socket is an abstraction of a communication endpoint

  Like file descriptors used to access a file, socket descriptors used to access
    sockets

  Functions that deal with file descriptors, such as read and write , will work
    with a socket descriptor also

  To create a socket, use the socket function

    #include <sys/socket.h>
    int socket(int domain, int type, int protocol);

    Returns: file (socket) descriptor if OK, −1 on error

    domain:  nature of the communication, like AF_ (for address family)
      AF_INET IPv4 Internet domain  , or AF_INET6 for IPv6 protocol

    type: type of the socket, communication characteristics, like SOCK_STREAM
      meaning sequenced, reliable, bidirectional, connection-oriented byte streams

    protocol: usually zero, selects default protocol for given domain, socket type
              This is the same number which appears on protocol field in the IP
              header of a packet.(man protocols for more details

    When done using the file descriptor, call close to relinquish access to the
      file or socket and free up the file descriptor for reuse

    int serverFd = socket(AF_INET, SOCK_STREAM, 0)
*/

// Server side

/*
   setsockopt - set options on sockets
   getsockopt()  and setsockopt() manipulate options for the socket referred to
     by the file descriptor sockfd

   Options may exist at multiple protocol levels; they are always present at the
     uppermost socket level

      #include <sys/types.h>
      #include <sys/socket.h>

      int setsockopt(int sockfd, int level, int optname,
                     const void *optval, socklen_t optlen);

   Helps in reuse of address and port , prevents error "address already in use"



   When manipulating socket options, the level at which the option resides and
     the name of the option must  be specified

   To  manipulate options at the sockets API level
     level is specified as SOL_SOCKET

   Optname and any specified options are passed uninterpreted to the appropriate
     protocol module for interpretation
   File <sys/socket.h> contains definitions for socket level options
   Most  socket-level  options  utilize  an int argument for optval
   For setsockopt(), the argument should be nonzero to enable a boolean option,
     or zero if the option is to be disabled

   Option name: SO_REUSEADDR
     Indicates  that the rules used in validating addresses supplied in a bind
       call should allow reuse of local addresses
     For AF_INET sockets this means that a socket may bind, except when there is
       an active listening socket bound to the address
     When the listening socket is bound to INADDR_ANY with a specific port then
       it is not possible to bind to this port for any local address

   Option name: SO_REUSEPORT
     Permits multiple AF_INET or AF_INET6 sockets to be bound to an identical
       socket address

   SO_REUSEPORT is not available on older POSIX systems

   The arguments optval and optlen are used to access option values for setsockopt
   If no option value is to be supplied or returned, optval may be NULL

   int option = 1

   If SO_REUSEPORT is available on your systems, then setsockopt can be

   setsockopt ( serverFd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT,
                                                 &option , sizeof ( option ) )


   else if SO_REUSEPORT is not available on your systems, setsockopt can be
   setsockopt ( serverFd, SOL_SOCKET, SO_REUSEADDR ,
                                                 &option , sizeof ( option ) )


   man 7 socket
*/

/*
   Address identifies a socket endpoint in a particular communication domain
   IPv4 Internet domain ( AF_INET ), a socket address is represented by
     a sockaddr_in structure:

     struct in_addr { in_addr_t     s_addr; };         // IPv4 address

     struct sockaddr_in {
                          sa_family_t     sin_family;  // address family
                          in_port_t       sin_port;    // port number
                          struct in_addr sin_addr;     // IPv4 address
                        };

   #define PORT 8080

   struct sockaddr_in address

   address.sin_family = AF_INET
   address.sin_addr.s_addr = INADDR_ANY
                    // socket accepts connections to all the IPs of the machine
   address.sin_port = htons( PORT )
                    // convert values between host and network byte order
*/

/*
   bind - bind a name to a socket

   When  a  socket  is  created  with socket, it exists in a name space
     (address family) but has no address assigned to it
   bind() assigns the address specified by sockaddr addr to the socket referred
     to by the file descriptor sockfd
   addrlen specifies the size, in bytes, of the address structure
     pointed to by addr
```

```
   This operation is called "assigning a name to a socket"

      #include <sys/types.h>
      #include <sys/socket.h>

      int bind(int sockfd, const struct sockaddr *addr, socklen_t addrlen);

   bind( serverFd, (struct sockaddr *) &address ,  sizeof(address) );

*/

/*
   listen - listen for connections on a socket

      #include <sys/types.h>
      #include <sys/socket.h>

      int listen(int sockfd, int backlog);

   listen marks the socket referred to by sockfd as a passive socket
   socket that will be used to accept incoming connection requests using accept

   backlog argument defines the maximum length to which the queue of pending
     connections for sockfd may grow
   If a connection request arrives when the queue is full, the client may
     receive an ECONNREFUSED error

   listen(serverFd, 2)
*/

/*
   accept - accept a connection on a socket

      #include <sys/types.h>
      #include <sys/socket.h>

      int accept(int sockfd, struct sockaddr *addr, socklen_t *addrlen);

   accept is used with connection-based socket types, extracts the first
     connection request on the queue of pending connections for the listening
     socket, sockfd
     creates a new connected socket, and returns a new file descriptor
       referring to that socket

   argument sockfd is a socket that has been created with socket, bound to a
     local address with bind, and is listening for connections after a listen

   int newSocket

   newSocket = accept ( serverFd , (struct sockaddr *)&address,
                                                   (socklen_t*) & addrlen) )
*/

/*
   Now read from new file descriptor newSocket into buffer using read

      #include <unistd.h>

      ssize_t read(int fd, void *buf, size_t count)

   char fileName[256] = {'\0'}
   int numberOfBytesRead

   As the client is requesting for content of file, read the file name
   numberOfBytesRead = read( newSocket , fileName, 1024)
*/

/*
   open - open and possibly create a file
```

```
      #include <sys/types.h>
      #include <sys/stat.h>
      #include <fcntl.h>

      int open(const char *pathname, int flags);
      int open(const char *pathname, int flags, mode_t mode);

   open returns a file descriptor, a small, nonnegative integer for use in
     subsequent system calls like read, write, lseek, fcntl

   flags must include one of the following access modes:O_RDONLY, O_WRONLY,
     or O_RDWR. These request opening the file read-only, write-only, or
     read/write, respectively
*/

/*
   send - send a message on a socket, transmit a message to another socket

      #include <sys/types.h>
      #include <sys/socket.h>

      ssize_t send(int sockfd, const void *buf, size_t len, int flags);

   send may be used only when the socket is in a connected state, so that the
     intended recipient is known
   message is found in buf and has length len

   Can write API be used instead?

   while content can be read from file into buffer

      send ( newSocket , buffer , strlen(buffer) , 0 )

*/

/*
   At       Server                     Client

       create   Socket            create   Socket
                 ↑ ↓                        ↑ ↓
                 ↑ ↓                        ↑ ↓
              setsockopt                    ↑ ↓
                 ↑ ↓                        ↑ ↓
                 ↑ ↓                        ↑ ↓
                bind                        ↑ ↓
                 ↑ ↓                        ↑ ↓
                 ↑ ↓                        ↑ ↓
                listen ⇄ ⇄ ⇄ ⇄ ⇄ connect ⇄  ↑ ↓
                 ↑ ↓                        ↑ ↓
                 ↑ ↓                        ↑ ↓
                accept                      ↑ ↓
                 ↑ ↓                        ↑ ↓
                 ↑ ↓                        ↑ ↓
            send /receive ⇄ ⇄ ⇄ ⇄ ⇄ ⇄ ⇄ send /receive
*/

/* g++ server.c -o server
   g++ client.c -o client
*/

/*
   Assume: Like Mark Watney escaping Mars gravity, assume no errors would occur
           The Martian; by Andy Weir
           else
           Test for errors on making each function call
*/

// Client Side
```

```
/*
   create a socket , save socket descriptor

   int socketFd

   socketFd = socket( AF_INET, SOCK_STREAM, 0)

*/

/* Where to connect - server

   Set values of server address to connect to


   struct sockaddr_in serv_addr

   Initialize using memory set memset - fill memory with a constant byte

      #include <string.h>

      void *memset(void *s, int c, size_t n);


   memset( &serv_addr, '0', sizeof(serv_addr))

   #define PORT 8080

   serv_addr.sin_family = AF_INET       // Connect using IVP4 protocol
   serv_addr.sin_port = htons(PORT)     // To the same port 8080

   Port is done, how about IP address
*/

/*
  Local server: when using the same system as server, address 127.0.0.1
  Like in XAMPP , WAMPP

  But, IPv4 addresses has to converted from text to binary form

      #include <arpa/inet.h>

      int inet_pton(int af, const char *src, void *dst);

  inet_pton function converts the character string src into a network address
    structure in the af address family,
    then copies the network address structure to dst

  inet_pton( AF_INET, "127.0.0.1", &serv_addr.sin_addr)

*/

/* Now connect to server

   connect - initiate a connection on a socket

      #include <sys/types.h>
      #include <sys/socket.h>

      int connect(int sockfd, const struct sockaddr *addr,
                  socklen_t addrlen);

   connect() system call connects the socket referred to by the file descriptor
     sockfd to the address specified by addr
   addrlen argument specifies the size of addr

   Connection-based protocol sockets may successfully connect() only once
   Connectionless  protocol sockets  may use connect() multiple times to change
     their association
   connect( socketFd, (struct sockaddr *)&serv_addr, sizeof(serv_addr))

*/

/* If connection is successful, then send/read to/from socket

   send - send a message on a socket, transmit a message to another socket

      #include <sys/types.h>
      #include <sys/socket.h>

      ssize_t send(int sockfd, const void *buf, size_t len, int flags);

   char clientMessage[18] = "Hello from Client"

   send( socketFd , clientMessage , strlen(clientMessage) , 0 )


   Now read from new file descriptor newSocket into buffer using read

      #include <unistd.h>

      ssize_t read(int fd, void *buf, size_t count)


   int numberOfBytesRead

   char buffer[1024] = {'\0'}

   numberOfBytesRead = read( socketFd , buffer, 1024)

*/

/* Addressing: machine's network address helps us identify the computer on the
     network we wish to contact, and the service helps us identify the particular
     process on the computer

   Byte Ordering: big and little endian

   APIs for converting between the processor byte order and the network byte order
*/

/* Textbook: W. Richard Stevens, Advanced Programming in the UNIX Environment,
     Pearson Education
   And
   Stack overflow
*/
/*
         RIP - Richard Stevens, Rajeev Motwani
*/
```