Network Programming Laboratory (UCS413)
Assignment:2

Introduction to basics of socket Programming:

a) Learn about socket header files and basic system calls.

Socket header files contain data definitions, structures, constants, macros, and options used by socket subroutines. An application program must include the appropriate header file to make use of structures or other information a particular socket subroutine requires.

Commonly used socket header files are:

- **/usr/include/netinet/in.h** Defines Internet constants and structures.
- **/usr/include/netdb.h** Contains data definitions for socket subroutines.
- **/usr/include/sys/socket.h** Contains data definitions and socket structures.
- **/usr/include/sys/types.h** Contains data type definitions.
- **/usr/include/arpa.h** Contains definitions for internet operations.
- **/usr/include/sys/errno.h** Defines the error no values that are returned by drivers and other kernel-level code.

Elementary socket system calls:

- **socket() System Call:** Creates an end point for communication and returns a descriptor:
  int socket (int AddressFamily, int Type, int Protocol);
- **Bind( ) System call:** Binds a name to a socket. The bind subroutine assigns a Name parameter to an unnamed socket. It assigns a local protocol address to a socket.
  int bind (int sockfd, struct sockaddr *myaddr, int addrlen);
- **connect()** System call: The connect function is used by a TCP client to establish a connection with a TCP server.
  int connect(int sockfd, struct sockaddr *servaddr, int addrlen);
- **listen() System call:** This system call is used by a connection-oriented server to indicate that it is willing to receive connections.
  int listen (int sockfd, int backlog);
- **accept() System call:** The actual connection from some client process is waited for by having the server execute the accept system call.
  int accept (int sockfd, struct sockaddr *cliaddr, int *addrlen);
- **send( )**, **sendto( )**, **recv( )** and **recvfrom( ) System calls:** These system calls are similar to the standard read and write functions.
- **close( ) system call**: The normal Unix close function is also used to close a socket and terminate a TCP connection.
  int close (int sockfd);

b) Create an Echo-Server using TCP socket programming in connection- oriented Scenario. Echo Server echo back (return back) the message sent by the client. Also, write the Client side program. *While creating program focus on the use of following socket programming functions.*
socket(), sockaddr_in, bind(), listen(), accept(), ntohs(), ntohl(), read()/recv(), write()/send() , connect().

<u>Steps to be followed:</u>

Server Side:

- Include appropriate header files.
- Create a TCP Socket.
- Bind the address and port using bind() system call.
- Server executes listen() system call to indicate its willingness to receive connections.
- Accept the next completed connection from the client process by using an accept() system call. At this point, connection is established between client and server, and they are ready to transfer data.
- Receive a message from the Client using recv()/read() system call.
- Send the received message back(echo) to the client using send()/write() system call.
- Close the socket using close() system call.

Client Side:

- Include appropriate header files
- Create a TCP Socket.
- Establish connection to the Server using connect() system call.
- Send and recieve messages using send() and recv() system call respectively.
- Close the socket using close() system call

Execution Steps:

- Save client and server program into two separate file with .c extension.
- Open two terminal and execute .c files by following commands
- gcc filename.c –o filename (compilation)
- ./filename (run)