Assignment 6

In this assignment, an introduction to socket programming is provided through the use of a tutorial (Section 6.1) and subsequently, an assignment (Section 6.2) is given to students for implementation using C programming platform, which tests the learning of the students through the tutorial provided.

6.1 Lab Tutorial: Socket Programming

In this section, first the concept of socket is introduced, followed by programming technique using a simple client/server communication application.

6.1.1 Objective

- Understand Sockets and some of the low-level networking concepts surrounding them.
- Learn about C socket API and the functions that are necessary to create sockets that will both act as a client and a server.

6.1.2 Pre-requisites: Knowledge of C: Functions, Loops

6.1.3 Points to note about Sockets

- Sockets are used to create connection between any two computers. It is a two-way endpoint.
- Common networking protocols like HTTP, FTP, etc rely on sockets to make connections.
- Using basic socket building block, applications like web servers, FTP clients (in fact, anything that interacts with the network) can be created.
- As it is a two-way endpoint, information can be both sent and received on any socket.

6.1.4 Socket Programming

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket (node) listens on a particular port at an IP, while another socket reaches out to the first to form a connection. Server forms the listener socket while client reaches out to the server.

6.1.5 Example: Create a simple client/server application in C using the concept of socket programming.

6.1.6 Client

In this section, the term ‘Client’ is defined and its socket workflow is explained in detail, through different functions used to implement the client.

Description

An application that runs on a personal computer. It has an extensive and appealing user interface, but it has no data that it can manipulate or present to the user. The client application 'asks' what a

user wants, then connects to the server application to obtain that data. Once the data has been obtained, it is presented to the user in a nice format. A client usually gets the data from one server at a time and interacts with one user only.

'Client' Socket Workflow

- The first part in any socket programming is to create the socket itself.
- The client socket is created with a *socket()* call. The *socket()* function returns an integer. In the socket call we specify the following parameters:

  domain: IPv4 (AF_INET)

  Type of socket: TCP/UDP (SOCK_STREAM)

  protocol: IP (0)

- Connection to a remote address is created with a *connect()* call. Here, we specify the IP address and the port that we are going to connect with. If the connection is successful, a value is returned.
- The data is retrieved with a *recv()* call. The received data can be stored in a file, or into a string.

Writing a Client Program

- Include standard libraries:
  #include <stdio.h>
  #include<stdlib.h>
- Include libraries for definition of socket functions.

  #include<sys/types.h>

  #include<sys/socket.h> //for socket APIs

  #include<netinet/in.h> //structure for storing address information

- Then, create the main() function

  int main()

  {

  return(0);

  }

- In the *main()* function, we need to do the following:
  a) Create the socket. We need an integer to hold the socket descriptor.

  int sockD;

b) Call the socket function whose result will be stored in the declared integer variable *sockD*.

sockD = socket(AF_INET, SOCK_STREAM, 0);

- This creates one of the endpoints (client) of the client/server network communication. For the other end (server), a similar function needs to be written.
- To connect to the other side (server), a *connect()* function needs to be called. But before the connection is established, an address needs to be specified to which to connect. Header file *netinet.h* contains the structure to specify the address and the port number of the remote socket.
- Declare the structure for the address:

  struct sockaddr_in servAddr;

- First we have to specify the address family (AF_INET). Now it is known that it will be an IPv4 address.

  servAddr.sin_familty = AF_INET;

- Now, specify the port to which to connect to. We can pass port number as an integer, but the structure variable accepts in network byte order only. Thus, the conversion function *htons()* will be used.

  servAddr.sin_port = htons(9001); //use some unused port number

- Next, specify the actual address. Either local address can be used (eg, 0.0.0.0) or an actual address.

  servAddr.sin_addr.s_addr = INADDR_ANY;

- Here *sin_addr* is also a structure, so we need another *'.'* to get into the fields in the structure. IN_ADDR_ANY is defined in the library and has a value 0.0.0.0.
- Now that the address is defined, *connect()* function can be called.

  a) First parameter is actual socket: *sockD*

  b) Second parameter is address: *servAddr*. This has to be cast into different structure pointer (*struct sockaddr* *). Therefore, pass the address of *servAddr* structure.

  c) The last parameter is the size of the address.

- The function *connect()* returns an integer that signifies whether the connection is successful or not. This can be used to do some primitive error handling. If the connection is successful, a value 0 is returned. Otherwise, -1 is returned.
- Thus, the *connect()* function can be implemented as

  int connetStatus = connect(sockD, (struct sockaddr) &servAddr, sizeof(servAddr);*

  if (connectStatus == -1)

{

// print some error message

}

- If the connection is successful, either send or receive data.
- Suppose, we need to receive the data from the user, *recv()* function will be used in this case.
  a) First parameter to *recv()* is the socket,
  b) Second parameter is some place to hold the data (suppose a string),
  c) Third parameter is the size of data,
  d) Lastly, there is an optional flag parameter (keep it 0).

  char strData[255];

  *recv(sockD, *strData, sizeof(strData), 0);*

  printf("%s", strData);

- Close the socket: *close(sockD);*

This is a basic TCP client that prints data, that it gets from the server. To test it, a TCP server needs to be created.

6.1.7 Server

In this section, the term 'Server' is defined and its socket workflow is explained in detail, through different functions used to implement the server.

Description

An application runs on a large computer (large meaning either fast or large storage space or both). The server application is usually written in such a way that it does not provide any method to interact with a user. Instead, it waits for other programs to connect (i.e. other programs take the place of a user) and interacts with these programs. Typically, a server application has control over large amounts of data and can access those data fast and efficiently. It can also handle requests by many clients (more or less) simultaneously.

'Server' Socket Workflow

- First a socket will be created (similar to client program).
- Next, the IP and port of the socket will be bound using *bind()* function (client used the *connect()* function to connect to the server; server will use the *bind()* function to listen for the connections).
- Then *listen()* function is called to listen to the connections (to see if any client is trying to connect to the server socket).
- The function *accept()* is called. The function returns a client socket (client that has connected).

- The functions *send()* and *recv()* are used to send and receive the data from other sockets it has connected to.

Writing a Server Program

- Include all library files as in the client program.
- Implement the *main()* function as follows:

```
int main()
{
    //create a string to hold data to be sent to client
    char serverData[255] = "Hello cleint";
    //create server socket similar to what was done in client program
    int servSockD;
    servSockD = socket(AF_INET, SOCK_STREAM, 0);
    //define server address
    struct sockaddr_in servAddr;
    servAddr.sin_family = AF_INET;
    servAddr.sin_port = htons(9001);
    servAddr.sin_addr.s_addr = INADDR_ANY;
    //bind socket to the specified IP and port
    bind(servSockD, (struct sockaddr*) &servAddr, sizeof(servAddr));
    //listen for connections
    listen(servSockD, 1);
```

**** Here, the second parameter is the number of connections waiting for this particular socket, keep it 1 for this example. For network applications that deal with lots of traffic, this number is important. ****

```
    //create an integer to hold client socket.
    int clientSocket;
    clientSocket = accept(servSockD, NULL, NULL);
    //send messages to client socket
    send(clientSocket, serverData, sizeof(serverData), 0);
    //close the socket
```

```
        close(servSockD);
}
```

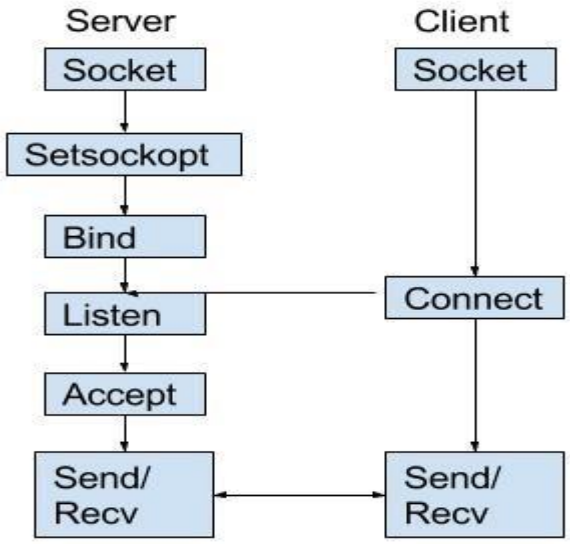
6.2 Assignment

In this section, the objective of assignment, flow diagram of client/server application, instructions to execute and instructions to submit the assignment are discussed.

6.2.1 Objective

Create a simple client/server application in C using the concept of socket programming.

6.2.2 Flow Diagram



6.2.3 Instructions to Execute

- Open two terminals on your machine and compile the server and the client programs in different terminals.
- Run the server program first, followed by running the client program.
- It can be seen that the data sent by server is printed on the terminal running the client program.
- Save the screenshot of execution as a jpeg file.

6.2.4 Instructions to Submit: Save the client program as *client.c*, server program as *server.c*, and screenshot of execution as *result.jpeg* in a folder identified with your roll number and upload on moodle in *.zip* format.

6.2.5 Submission Deadline: April 4, 2020