VU VISHWAKARMA UNIVERSITY

Maximising Human Potential

Student Name	**Samarth Abhay Kadam**
SRN No	202200739
Roll No	10
Program	AI & DS
Year	Third Year
Division	A
Subject	Computer Network Laboratory (BTECCE21506)
Assignment No	Ten

Assignment Number - 10

Title : Socket Programming for UDP Client and TCP Server.

Problem Statement : Write a C /C++ / Java /Python socket program for UDP where UDP Client communicate with UDP server.

Theory :

A **network socket** is an endpoint of an inter-process communication flow across a computer network. Today, most communication between computers is based on the Internet Protocol; therefore most network sockets are **Internet sockets**.

A **socket API** is an application programming interface (API), usually provided by the operating system, that allows application programs to control and use network sockets. Internet socket APIs are usually based on the Berkeley sockets standard.

A **socket address** is the combination of an IP address and a port number, much like one end of a telephone connection is the combination of a phone number and a particular extension. Based on this address, internet sockets deliver incoming data packets to the appropriate application process or thread.

Procedure in Client-Server Communication

There are some procedures that we have to follow to establish client-server communication. These are as follows.

Socket: With the help of a socket, we can create a new communication.

Bind: With the help of this we can, we can attach the local address with the socket.

Listen: With this help; we can accept the connection.

Accept: With this help; we can block the incoming connection until the request arrives.

Connect: With this help; we can attempt to establish the connection.

Send: With the help of this; we can send the data over the network.

Receive: With this help; we can receive the data over the network.

Close: With the help of this, we can release the connection from the network.

UDP Socket API

There are some fundamental differences between TCP and UDP sockets. UDP is a connection-less, unreliable, datagram protocol (TCP is instead connection-oriented, reliable and stream based). There are some instances when it makes to use UDP instead of TCP. Some popular applications built around UDP are DNS, NFS, SNMP and for example, some Skype services and streaming media.

Figure 4 shows the the interaction between a UDP client and server. First of all, the client does not establish a connection with the server. Instead, the client just sends a datagram to the server using the sendtofunction which requires the address of the destination as a parameter. Similarly, the server does not accept a connection from a client. Instead, the server just calls the recvfrom function, which waits until data arrives from some client. recvfrom returns the IP address of the client, along with the datagram, so the server can send a response to the client.

As shown in the Figure, the steps of establishing a UDP socket communication on the client side are as follows:

· Create a socket using the socket() function;

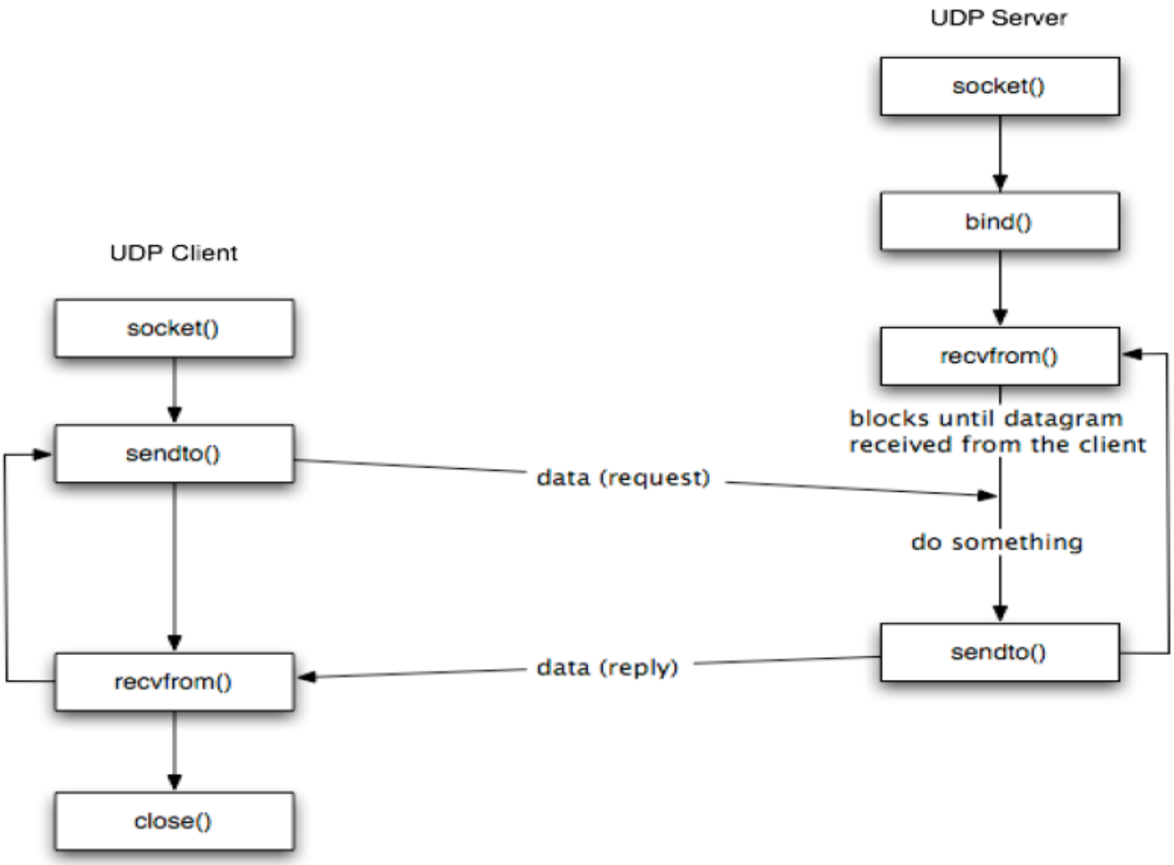
· Send and receive data by means of the recvfrom() and sendto() functions.

The steps of establishing a UDP socket communication on the server side are as follows:

· Create a socket with the socket() function;

· Bind the socket to an address using the bind() function;

· Send and receive data by means of recvfrom() and sendto().



UDP client-server.

Program :

//UDP SERVER PROGRAM

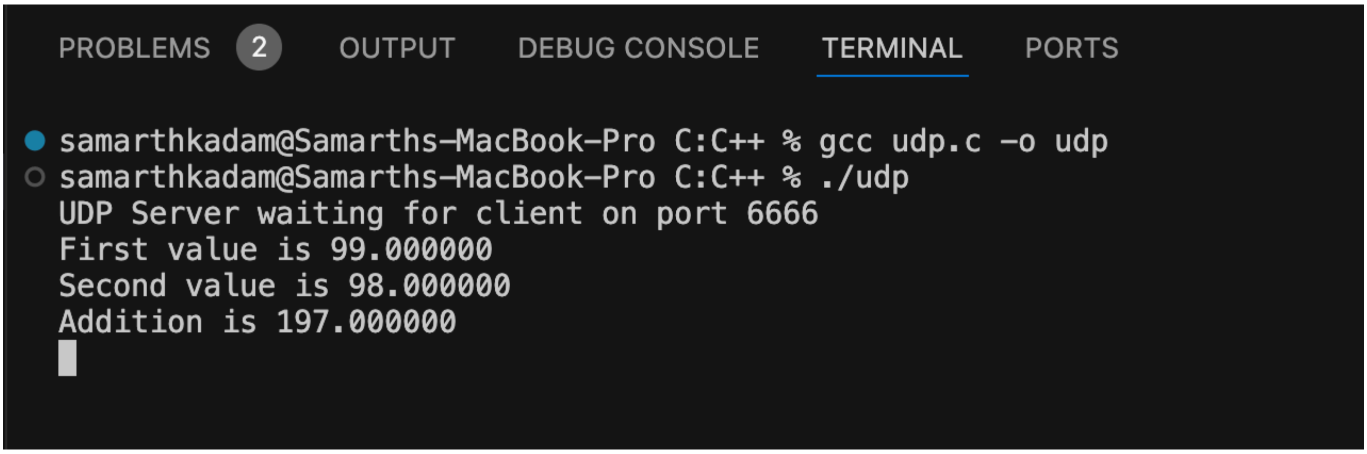
```
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<stdio.h>
#include<unistd.h>
#include<errno.h>
#include<string.h>
#include<stdlib.h>
int main()
{
        struct data
        {
                float x;
                float y;
        }gdata;
        int sock;
        int addr_len,bytes_read;
        char recv_data[1024];
        float sum ;
        struct sockaddr_in server_addr,client_addr;
        if((sock=socket(AF_INET,SOCK_DGRAM,0))==-1)
        {
                perror("Socket");
                exit(1);
        }
        server_addr.sin_family=AF_INET;
        server_addr.sin_port=htons(6666);
        server_addr.sin_addr.s_addr=INADDR_ANY;
        bzero(&(server_addr.sin_zero),8);
        if(bind(sock,(struct sockaddr *)&server_addr,sizeof(struct sockaddr))==-1)
        {
                perror("Bind");
                exit(1);
        }
        addr_len=sizeof(struct sockaddr);
        printf("\n UDPServer waiting for client on port no 6666 \n");
        fflush(stdout);
        while(1)
        {
                recvfrom(sock,&gdata,sizeof(struct data),0,(struct sockaddr *)&client_addr,&addr_len);
                printf("First value is %f \n",gdata.x);
                printf("Second value is %f \n",gdata.y);
                sum = gdata.x + gdata.y;
                printf ("Addition is %f \n",sum);
                sendto(sock,&sum,sizeof(sum),0,(struct sockaddr *)&client_addr,addr_len);
                fflush(stdout);
        }
        return 0;
}
```

//UDP CLIENT PROGRAM

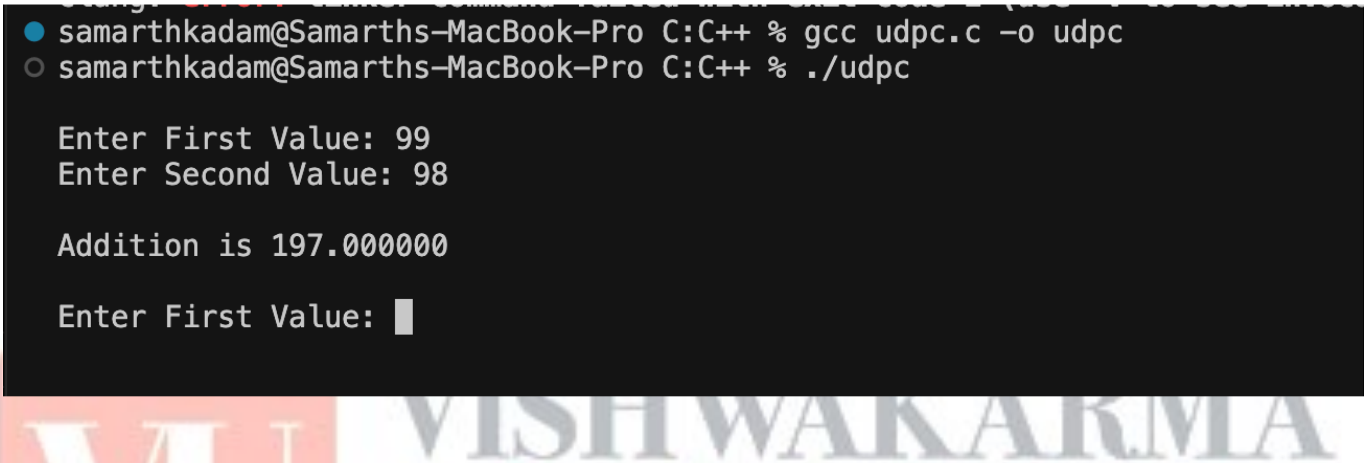
```
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<netdb.h>
#include<stdio.h>
#include<unistd.h>
```

```
#include<errno.h>
#include<string.h>
#include<stdlib.h>

int main()
{
  int sock;
  int add_len;
  struct sockaddr_in server_addr;
  struct hostent *host;
  char send_data[1024];
  struct data
  {
        float x;
        float y;
  }sdata;
  float sum;
  host=(struct hostent *)gethostbyname((char*)"127.0.0.1");
  if((sock=socket(AF_INET,SOCK_DGRAM,0))==-1)
  {
   perror("socket ");
   exit(1);
  }
  server_addr.sin_family=AF_INET;
  server_addr.sin_port=htons(6666);
  server_addr.sin_addr=*((struct in_addr*)host->h_addr);
  bzero(&(server_addr.sin_zero),8);
  add_len = sizeof(struct sockaddr);

  while(1)
  {
    printf("\n Enter First Value :");
    scanf("%f",&sdata.x);
```

```
      printf("\n Enter Second value:");
      scanf("%f",&sdata.y);
      sendto(sock,&sdata,sizeof(struct data),0,(struct sockaddr *)&server_addr,sizeof(struct sockaddr));
      recvfrom(sock,&sum,sizeof(sum),0,(struct sockaddr *)&server_addr,&add_len);
      printf("\n Addition is %f",sum);
   }
}

/***********************************
        UDP SERVER TERMINAL
************************************/
samarthkadam@Samarths-MacBook-Pro C:C++ % gcc udp.c -o udp
samarthkadam@Samarths-MacBook-Pro C:C++ % ./udp
UDP Server waiting for client on port 6666
First value is 99.000000
Second value is 98.000000
Addition is 197.000000

/***********************************
        UDP CLIENT TERMINAL
************************************/


samarthkadam@Samarths-MacBook-Pro C:C++ % gcc udpc.c -o udpc
samarthkadam@Samarths-MacBook-Pro C:C++ % ./udpc


Enter First Value: 99
Enter Second Value: 98


Addition is 197.000000
```



```
● samarthkadam@Samarths-MacBook-Pro C:C++ % gcc udp.c -o udp
○ samarthkadam@Samarths-MacBook-Pro C:C++ % ./udp
  UDP Server waiting for client on port 6666
  First value is 99.000000
  Second value is 98.000000
  Addition is 197.000000
```



```
● samarthkadam@Samarths-MacBook-Pro C:C++ % gcc udpc.c -o udpc
○ samarthkadam@Samarths-MacBook-Pro C:C++ % ./udpc

  Enter First Value: 99
  Enter Second Value: 98

  Addition is 197.000000

  Enter First Value:
```

Conclusion :

In this experiment, we implemented a UDP-based client-server model for performing basic arithmetic operations, specifically addition, using socket programming. The client program sends two float values to the server over a UDP connection, and the server responds with their sum. This exercise demonstrates fundamental aspects of networking, such as creating sockets, establishing server-client communication, and data transmission over UDP, which is a connectionless and lightweight protocol.

Through this experiment, we gained practical experience in working with UDP sockets, handling byte-ordering conversions, and managing different data structures for communication. Additionally, by porting the code from Windows to macOS, we learned about cross-platform compatibility considerations in network programming. This experiment serves as a foundational exercise for understanding network protocols, which is essential for building efficient and scalable networked applications.