

**MAWLANA BHASHANI SCIENCE AND TECHNOLOGY UNIVERSITY,
Santosh, Tangail -1902**



Lab Report No : 03
Lab Report Name : Socket Programming Implementation
Course Name : Computer Networks Lab

Submitted by,

Name : Md Imtyaz Ahmed

ID : IT-17017

Session : 2016-17
Dept. of ICT, MBSTU.

Submitted to,

Nazrul Islam

Assistant Professor

Dept. of ICT, MBSTU.

1. What is Socket Programming ?

Ans: Socket Programming is the fundamental technology behind communications on TCP/IP networks . A socket is one endpoint of a twoway link between two programs running on a network. The socket provides a bidirectional communication endpoint to send and receive data with another socket. Socket connections normally run between two different computers on a Local Area Network(LAN) or across the internet, but they can also be used for interprocess communication on a single computer.

2. Briefly explain the term IPC in terms of TCP/IP communication.

Ans: In computer science, inter-process communication or interprocess communication (IPC) refers specifically to the mechanisms an operating system provides to allow the processes to manage shared data. Typically, applications can use IPC, categorized as clients and servers, where the client requests data and the server responds to client requests. Many applications are both clients and servers, as commonly seen in distributed computing.

An operating system provides interprocess communication to allow processes to exchange information. Processes can communicate with each other through both:

- i. Shared Memory
- ii. Message Passing

3. What is the maximum size of a UDP datagram? What are the implications of using a packet-based protocol as opposed to a stream protocol for transfer of large files.

Ans: It depends on the underlying protocol i.e., whether you are using IPv4 or IPv6. In IPv4, the maximum length of packet size is 65,536. So, for UDP datagram you have maximum data length as:

65,535 bytes - 20 bytes(Size of IP header) = 65, 515 bytes (including 8 bytes UDP header).

In IPv6, the maximum length of packet size allowed is 64 kB, so, you can have UDP datagram of size greater than that.

TCP is connection-oriented, and a connection between client and server is established before data can be sent.

4. TCP is a reliable transport protocol, briefly explain what techniques are used to provide this reliability?

Ans: In computer networking, a reliable protocol is a communication protocol that notifies the sender whether or not the delivery of data to intended recipients was successful. Transmission Control Protocol (TCP), the main protocol used on the Internet, is a reliable unicast protocol. UDP is an unreliable protocol and is often used in computer games, streaming media or in other situations where speed is an issue and some data loss may be tolerated because of the transitory nature of the data.

5. Why are the htons() , htonl() , ntohs() , ntohl() functions used?

Ans: htons() : Host to Network Short
htonl() : Host to Network Long
ntohs(): Network to Host Short
ntohl() : Network to Host Long

6. What is the difference between a datagram socket and a stream socket? Which transport protocols do they correspond to?

Ans: A stream socket is like a phone call -- one side places the call, the other answers, you say hello to each other (SYN/ACK in TCP), and then you exchange information. Once you are done, you say goodbye (FIN/ACK in TCP). If one side doesn't hear a goodbye, they will usually call the other back since this is an unexpected event; usually the client will reconnect to the server. There is a guarantee that data will not arrive in a different order than you sent it, and there is a reasonable guarantee that data will not be damaged.

A datagram socket is like passing a note in class. Consider the case where you are not directly next to the person you are passing the note to; the note will travel from person to person. It may not reach its destination, and it may be modified by the time it gets there. If you pass two notes to the same person, they may arrive in an order you did intend, since the route the notes take through the classroom may not be the same, one person might not pass a note as fast as another, etc.

7. What is a stateful service, as opposed to a stateless service? What are the performance implications of statefulness and statelessness?

Ans: The main differences between Stateful and Stateless applications are – Applications in Stateful react by the current state, while Stateless applications act independently with taking into consideration previous/next request. If the webserver stores data in a backend manner and uses it to identify the user as an always-connected client, the service is Stateful. While in Stateless, the server does store data, but in a database to verify user/client whenever it needs to connect. In Stateful, the server thinks a client is just a dumb machine, while in Stateless, server thinks the client is an intelligent machine that doesn't need to depend on any state on the server-side. In Stateless, requests are self-contained, i.e. everything contained within the request, and handled in two distinct phases, a "request" and "response." While in Stateful, requests always dependant on the serverside state. While browsing the internet, the state generated and stored somewhere. Although the state generated in both types when the state stored on the server, it generates a session. This is called a Stateful application. When the state stored by the client, it generates some data used for further requests while technically "Stateful" as it references a state, but the state stored by the client, so call it Stateless. On the client-side, cookie stores authentication data. On the server-side, create temporary client data or store on a database (this is the typical case). While returning to the dashboard to make another payment, it's cookie stored in the browser that establishes the state with the server. Stateful is past when there were Monoliths and no dynamic user base. Stateless is future, having Microservices floating around and mostly communicate through REST interfaces and scale-like anything since there is no state stored.

8. Time Protocol : Your task in this exercise is implement Time protocol, as defined in RFC868 (<http://www.faqs.org/rfcs/rfc868.html>). You should write both a UDP and TCP server, as well as clients that access the services. The client should display the result back to the user in a "nice" way. This means that you should not just dump the raw 32-bit data you get from the server on the command line, but present it in a humanly readable fashion. The deliverables should be called TimeUdpClient.java,

TimeUdpServer.java, TimeTcpClient.java and TimeTcpServer.java, if you have chosen "Java" as a language for the exercises. Otherwise, use the appropriate file extension.

Ans: **Source code :**

TimeTcpClient.Java:

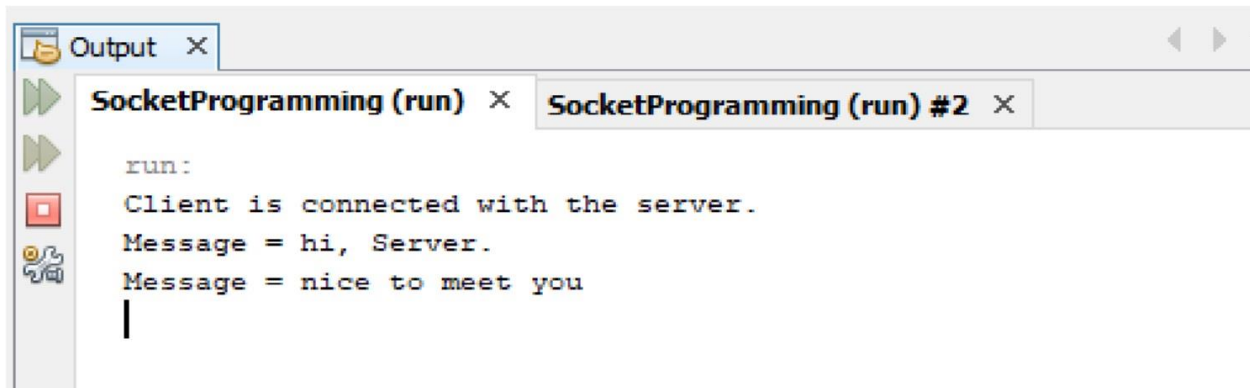
```
package TimeProtocol; import
java.io.*; import java.net.*; public
class TimeTcpClient {    public static
void main(String[] args) {    try {
        Socket socket = new Socket("localhost", 3306);
        DataOutputStream dout = new
DataOutputStream(socket.getOutputStream());
        BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
        while (true) {
            System.out.println("Say something to server: ");
            String s = br.readLine();
dout.writeUTF(s);            if
(s.equalsIgnoreCase("exit")) {
break;
        }
    }
    socket.close();
} catch (Exception e) {
    System.out.println(e);
}
}
```

Output :

TimeTcpServer.java:

```
package TimeProtocol;
import java.io.*; import
java.net.*; public class
TimeTcpServer {
    public static void main(String[] args) {
//Age server run koraba.Then client run
koraba    try {
        ServerSocket ss = new ServerSocket(6666);
        Socket s = ss.accept();//establishes connection
        System.out.println("Client is connected with the server.");
        DataInputStream dis = new DataInputStream(s.getInputStream());
while (true) {
        String str = (String) dis.readUTF();
System.out.println("Message = " + str);
        if (str.equalsIgnoreCase("exit")) {
break;
        } }
        ss.close();
    } catch (Exception e) {
        System.out.println(e);
    }
}
}
```

Output :



TimeUdpClient.java :

```
package TimeProtocol; import java.net.*; public class
TimeUdpClient {    public static void main(String[] args)
throws Exception {
    DatagramSocket ds = new DatagramSocket();
    String str = "Hi!";
    InetAddress ip = InetAddress.getByName("127.0.0.1");
    DatagramPacket dp = new DatagramPacket(str.getBytes(), str.length(),
ip,
3000);
    ds.send(dp);
    System.out.println("The message has been succesfully sent.");

    ds.close();
}
}
```

Output:



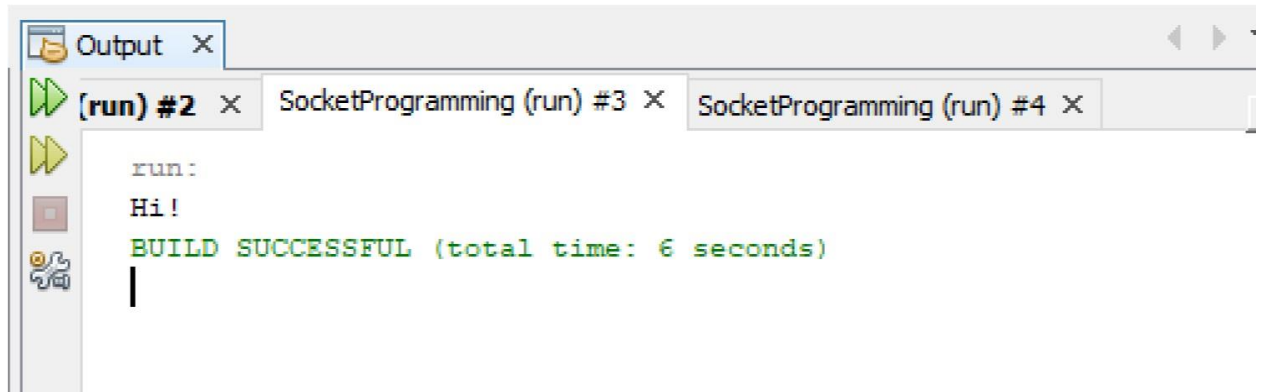
TimeUdpServer.java :

```
package TimeProtocol;
import java.net.*;

public class TimeUdpServer {

    public static void main(String[] args) throws Exception {
        DatagramSocket ds = new DatagramSocket(3000);
        byte[] buf = new byte[1024];
        DatagramPacket dp = new DatagramPacket(buf, 1024);
        ds.receive(dp);
        String str = new String(dp.getData(), 0, dp.getLength());
        System.out.println(str);
        ds.close();
    }
}
```

Output:



9. Write socket programming code and display the output?

Ans: Client.py :

```
import socket
import select
import errno

HEADER_LENGTH = 10

IP = "127.0.0.1"
PORT = 1234
my_username = input("Username: ")
client_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM) client_socket.connect((IP, PORT))
client_socket.setblocking(False) username =
my_username.encode('utf-8')
username_header = f"{len(username):<{HEADER_LENGTH}}".encode('utf-
8') client_socket.send(username_header + username)
while
True:
message = input(f'{my_username} > ')
if message:
    message = message.encode('utf-8')
    message_header =
f"{len(message):<{HEADER_LENGTH}}".encode('utf-
8')
    client_socket.send(message_header + message)

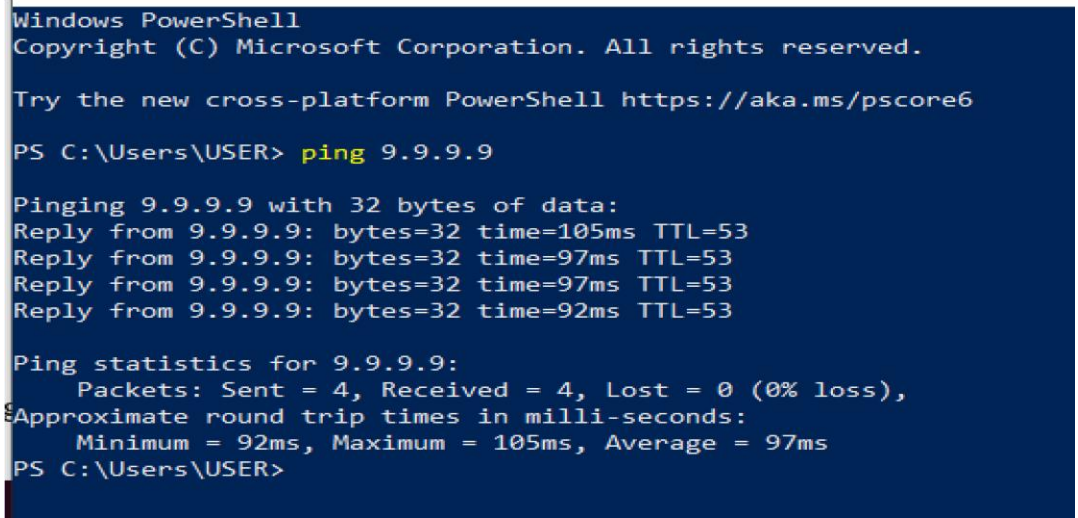
tr
y:
while True:
```

```

username_header =
client_socket.recv(HEADER_LENGTH) if not
len(username_header): print('Connection
closed by the server') sys.exit()
username_length = int(username_header.decode('utf-8').strip())
username = client_socket.recv(username_length).decode('utf-8')
message_header = client_socket.recv(HEADER_LENGTH)
message_length = int(message_header.decode('utf-8').strip())
message = client_socket.recv(message_length).decode('utf-8')
print(f'{username} > {message}') except IOError as e: if
e.errno != errno.EAGAIN and e.errno != errno.EWOULDBLOCK:
print('Reading error: {}'.format(str(e))) sys.exit()
continue except Exception as e: print('Reading error:
'.format(str(e))) sys.exit()

```

Output :



```

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\USER> ping 9.9.9.9

Pinging 9.9.9.9 with 32 bytes of data:
Reply from 9.9.9.9: bytes=32 time=105ms TTL=53
Reply from 9.9.9.9: bytes=32 time=97ms TTL=53
Reply from 9.9.9.9: bytes=32 time=97ms TTL=53
Reply from 9.9.9.9: bytes=32 time=92ms TTL=53

Ping statistics for 9.9.9.9:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 92ms, Maximum = 105ms, Average = 97ms
PS C:\Users\USER>

```

Server.py :

```

import socket
import select
HEADER_LENGTH =
10
IP = "127.0.0.1"
PORT = 1234

```

```

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR,
1) server_socket.bind((IP, PORT)) server_socket.listen()
sockets_list = [server_socket]
clients = {}
print(f'Listening for connections on
{IP}:{PORT}...') def
receive_message(client_socket): try:
    message_header =
client_socket.recv(HEADER_LENGTH) if not
len(message_header): return False
    message_length = int(message_header.decode('utf-8').strip())
    return {'header': message_header,
'data': client_socket.recv(message_length)}
except: return False while True: read_sockets, _,
exception_sockets = select.select(sockets_list, [], sockets_list)
for notified_socket in read_sockets: if notified_socket ==
server_socket: client_socket, client_address =
server_socket.accept() user =
receive_message(client_socket) if user is False: continue
    sockets_list.append(client_socket)

    clients[client_socket] = user
    print('Accepted new connection from {}:{}'.format(*client_address,
user['data'].decode('utf-8'))) else:
    message =
receive_message(notified_socket) if message
is False: print('Closed connection from:
{}'.format(clients[notified_socket]['data'].decode('utf-
8')))) sockets_list.remove(notified_socket) del
clients[notified_socket] continue
user = clients[notified_socket] print(f'Received message
from {user["data"].decode("utf-8")}'):
{message["data"].decode("utf-8")}')
    for client_socket in clients: if
client_socket != notified_socket:
        client_socket.send(user['header'] + user['data'] +
message['header'] + message['data'])
    for notified_socket in
exception_sockets:
sockets_list.remove(notified_socket)

del clients[notified_socket]

```

Output:



Conclusion : The socket programming will provide the ability of the implement in analytics, streaming in binary, document collaboration and so on. On the positive side, The IO can control the connection in sockets. For this reason, both the server and also client side is consists of IO libraries. It makes the real-time application are feasible in the mobile devices in Embedded Operating System.