# Experiment 6: Socket Programming-I

**Aim:** To use TCP Sockets for Inter Process Communication

**Objective:** After carrying out this experiment, students will be able to:

* Apply TCP Socket programming technique to establish IPC between remote processes
* Analyse the difference between sockets and other enabling techniques for IPC such as Pipes and Message Queues

**Problem statement:** You are required to write programs to implement a two TCP based servers and clients.

(a) The functionality of this server (echo server) is that it should echo any data it receives from a client back to it (the client).

(b) The client will send two integers/floats the server. The server will add them and sent it back to the client. Display the sum in client.

**Analysis:** While analyzing your program, you are required to address the following points:

* How is socket programming different from other techniques for IPC such as Pipes and Message Queues?
* What happens if the number of incoming client requests exceeds the second argument of the listen() function in the server?

**MARKS DISTRIBUTION**

|  |  |  |
| --- | --- | --- |
| **Component** | **Maximum Marks** | **Marks Obtained** |
| Preparation of Document | 7 |  |
| Results | 7 |  |
| Viva | 6 |  |
| **Total** | **20** |  |

Submitted by: PRACHI PODDAR

Register No: 17ETCS002122

1. **Algorithm/Flowchart**

Client code description:

* Include all the standard libraries and the libraries required for socket programming like <netb.h> which is responsible for reserving highest port internet number, <sys/socket.h>,
* Create a structure of socket

struct sockaddr\_in server

listen\_fd = socket(AF\_INET, SOCK\_STREAM, 0)

bzero( &servaddr, sizeof(servaddr))

servaddr.sin\_family = AF\_INET

servaddr.sin\_addr.s\_addr = htons(INADDR\_ANY)

servaddr.sin\_port = htons(8080)

bind(listen\_fd, (struct sockaddr \*) &servaddr, sizeof(servaddr))

listen(listen\_fd, 10)

* Echo the message back to the server using commands like accept(), with appropriate arguments.

Server code description:

* Include all the standard and necessary files required for socket programming.
* Create a structure for the server as well by using struct data-type.
* Pass the main string that needs to be echoed back by the client.

while(1)

bzero(sendline,100)

bzero(recvline,100)

fgets(sendline,100,stdin)

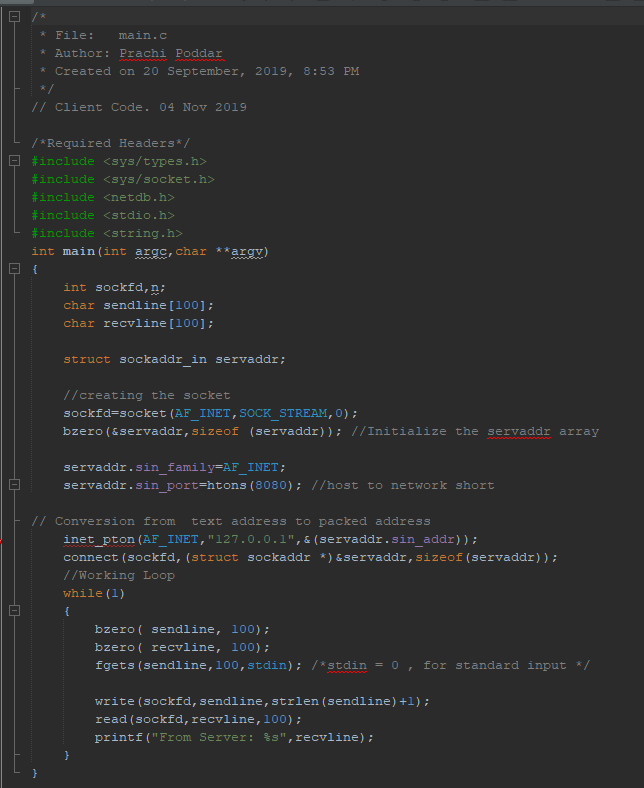
write(sockfd,sendline,strlen(sendline)+1)

read(sockfd,recvline,100)

printf("From Server: %s",recvline)

1. **Program**
2. **CODE**

Code for client:



Code for server:

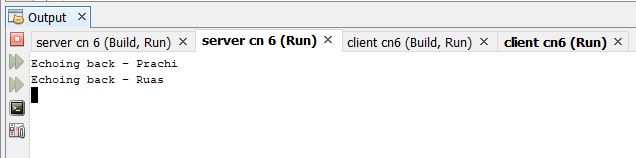


OUTPUT:

Client:

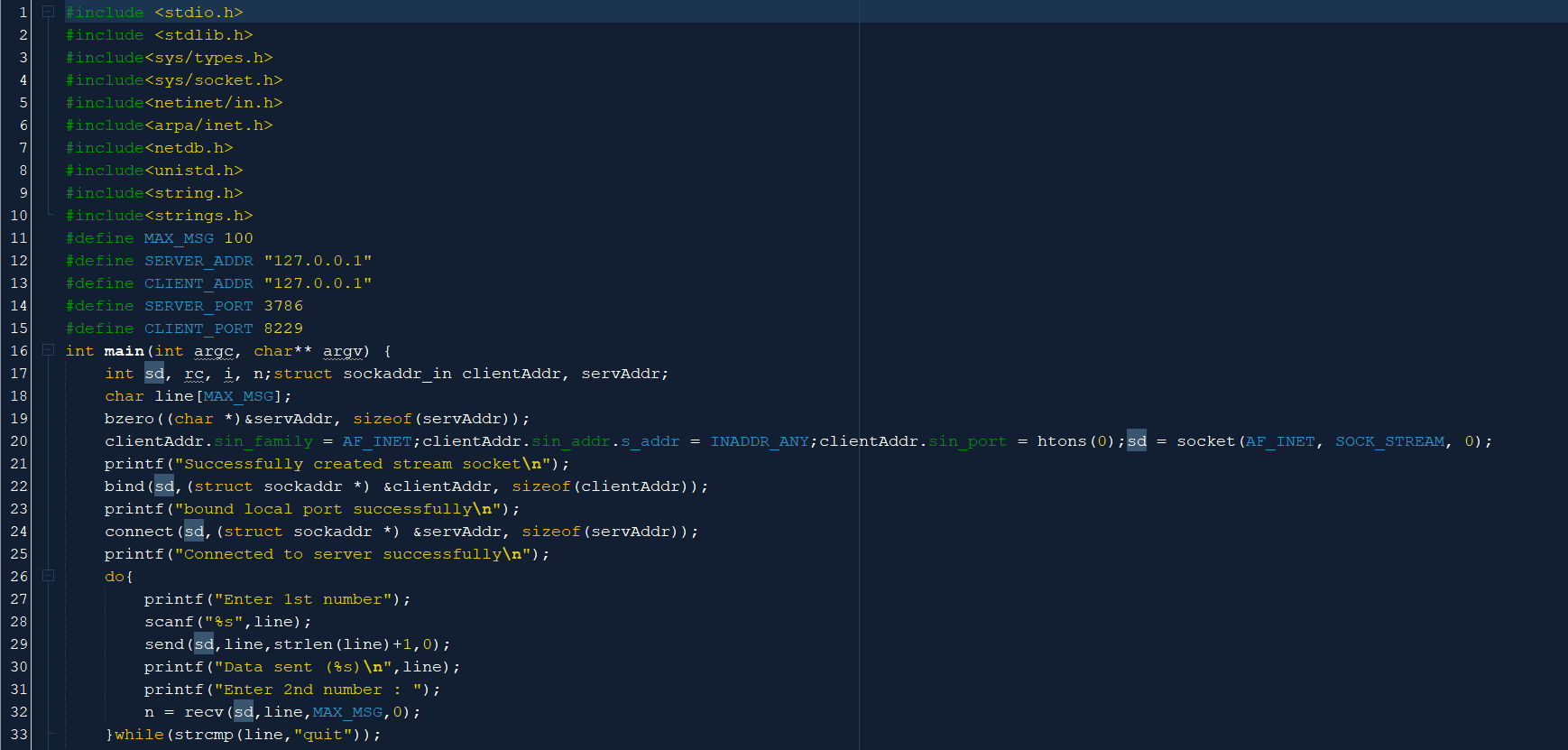


Server:



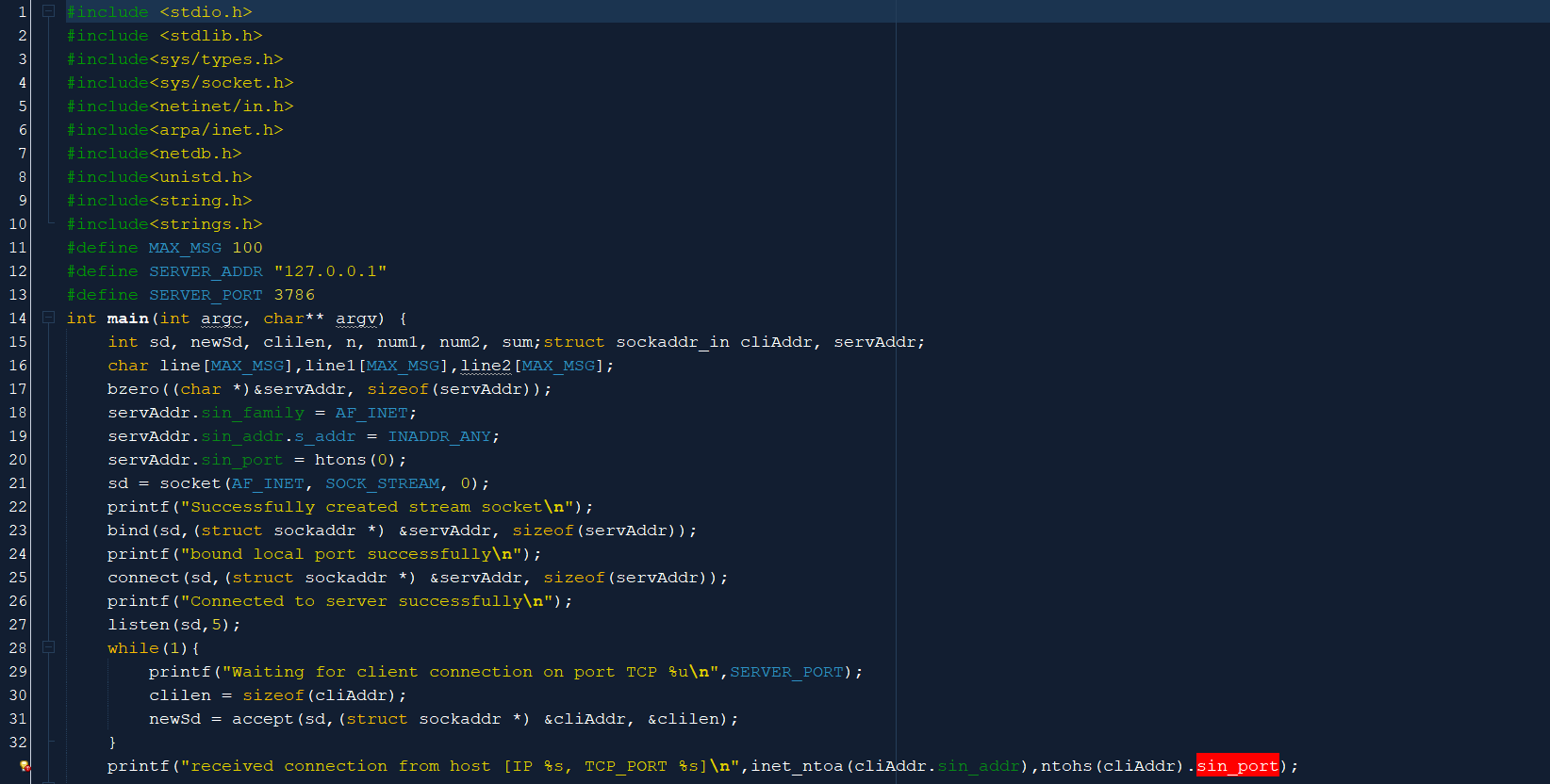
1. **CODE**:

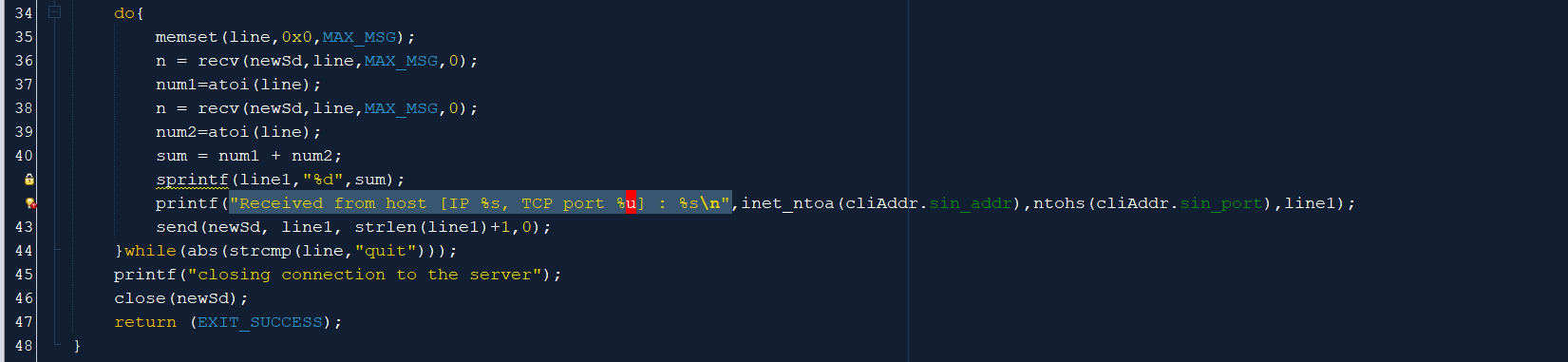
Client:





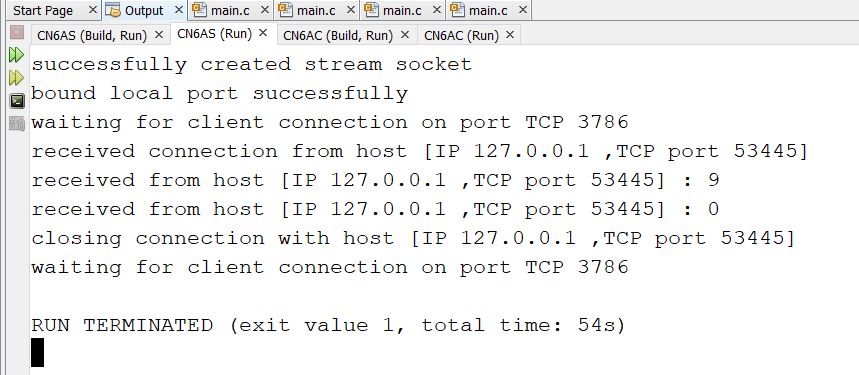
Server:



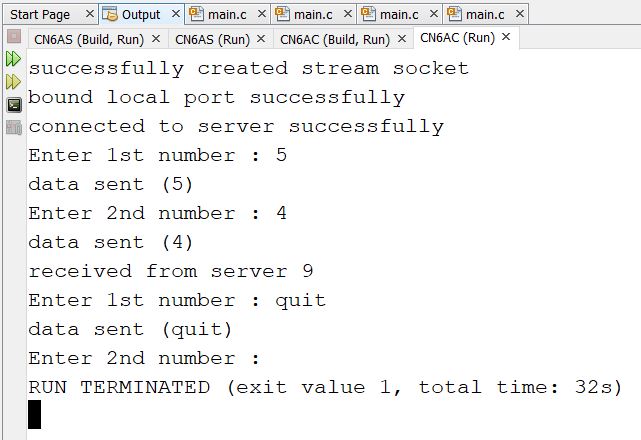


OUTPUT:

Server output:



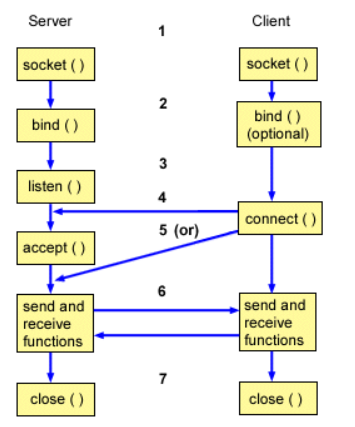
Client output:



1. **Analysis and discussion**

**Socket programming –** A socket is a communications connection point (endpoint) that you can name and address in a network. Socket programming shows how to use socket APIs to establish communication links between remote and local processes.

The processes that use a socket can reside on the same system or different systems on different networks. Sockets are useful for both stand-alone and network applications. Sockets allow you to exchange information between processes on the same machine or across a network, distribute work to the most efficient machine, and they easily allow access to centralized data. Socket application program interfaces (APIs) are the network standard for TCP/IP. A wide range of operating systems support socket APIs. IBM® i sockets support multiple transport and networking protocols. Socket system functions and the socket network functions are threadsafe.



**Pipes** are used as a medium to simply exchange information between sender and receiver. The messages that are exchanged are not analysed or processed. Pipes are used in inter-process communication.

**Message queues** – In this, a queue is used to store the messages which are being exchanged between the sender and the receiver. A process inputs a message in the queue which allows another process to read it. An interface is provided to the processes and these processes use this interface to access the message queue to either put one message in the queue for other multiple processes to read it or read one message from the queue.

1. **Conclusion**

A socket has a typical flow of events. In a connection-oriented client-to-server model, the socket on the server process waits for requests from a client. To do this, the server first establishes (binds) an address that clients can use to find the server. When the address is established, the server waits for clients to request a service. The client-to-server data exchange takes place when a client connects to the server through a socket. The server performs the client's request and sends the reply back to the client.