B.M.S College of Engineering

P.O. Box No.: 1908 Bull Temple Road,

Bangalore-560 019

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

Course – Unix System Programming

Course Code –23IS3AEUSP

AY 2023-24

Report on Unix System Programming Project

Title: Socket Programming

Submitted By:

Somanath Mikali,1BM22IS196

Vishesh P Gowda, 1BM22IS232

Syed Asif Hussain Madni,1BM22IS214

Submitted To:

Dr. Shubha Rao V

B.M.S College of Engineering

P.O. Box No.: 1908 Bull Temple Road,

Bangalore-560 019

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

CERTIFICATE

Certified that the Project has been successfully presented at **B.M.S College of Engineering** by **Somanath Mikali, Vishesh P Gowda and Syed Asif Hussain Madni** bearing USN: **1BM22IS196, 1BM22IS232 and 1BM22IS214** in partial fulfillment of the requirements for the III Semester degree in **Bachelor of Engineering in Information Science & Engineering of Visvesvaraya Technological University, Belgaum** as a part of project for the Course– Unix System Programming Course Code – 23IS3AEUSP during academic year 2023-2024

Faculty Name- Dr. Shubha Rao V

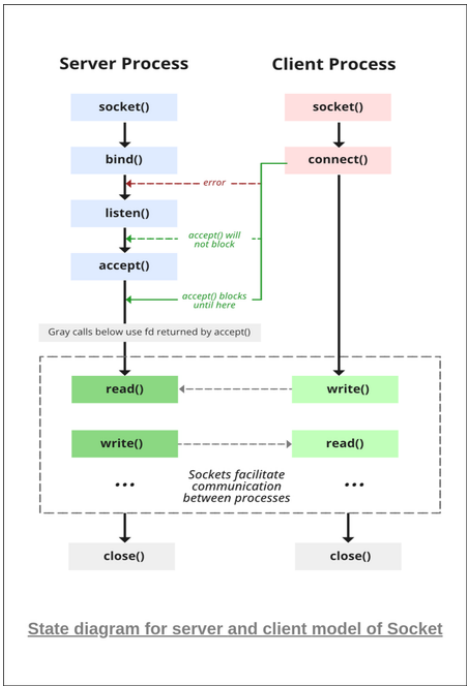
Designation- Associate Professor

Department of ISE, BMSCE

Table of Contents

1. Introduction
2. API’s used
3. Implementation
4. Application
5. Snapshots
6. References

Introduction



State diagram for server and client model of Socket

Socket programming is a fundamental concept in computer networking that enables communication between computers over a network. It involves the use of software-based communication endpoints called "sockets." These sockets allow data to be sent and received between devices, facilitating the development of various networked applications.

The process typically involves creating a socket, specifying the communication protocol, binding it to a specific address and port, and then either waiting for incoming connections (server) or initiating connections (client). Data can be exchanged between connected sockets, and the connection is eventually closed when the communication is complete.

Socket programming is crucial for building networked applications such as web servers, chat applications, and online games. It provides a flexible and powerful mechanism for devices to communicate and share data across a network. The implementation details may vary depending on the programming language and the specific application requirements, but the basic principles remain consistent across different scenarios.

API's Used in the Project

socket():Used to create a socket file descriptor.

bind():Assigns a local protocol address to a socket.

listen():Sets up a queue for incoming client connections.

accept():Creates a new connected socket for the incoming client.

fork():Used to handle multiple clients concurrently.

gethostbyname(): Retrieves host information (such as IP addresss)

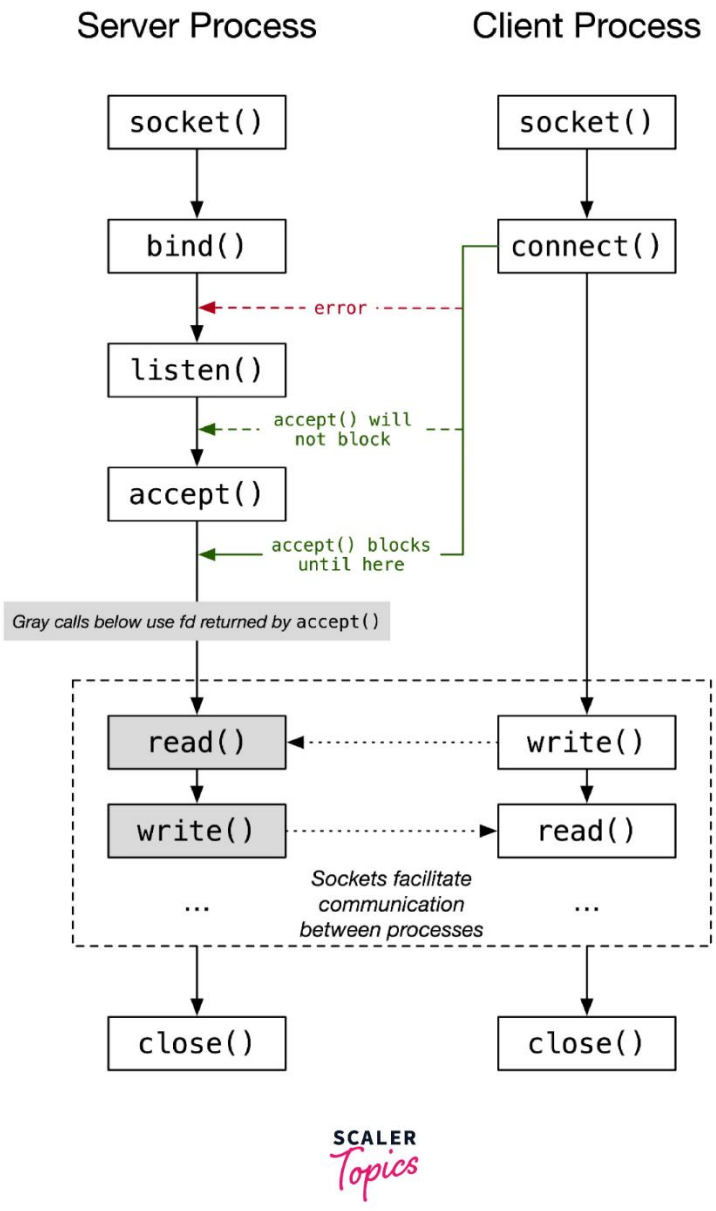
from a host name.

connect():Establishes a connection to a specified socket.

write():Sends data to the connected socket.

read():Receives data from the connected socket.

Implementation



SCALER Topics

Making Server Process

The system calls for establishing a connection are somewhat different for the client and the server, but both involve the basic construct of a socket. Both the processes establish their own sockets.

The steps involved in establishing a socket on the client side are as follows −

- Create a socket with the **socket()** system call.
- Connect the socket to the address of the server using the **bind()** system call.
- Send and receive data. There are a number of ways to do this, but the simplest way is to use the **read()** and **write**() system calls.

Making Server Process contd.

1.Create a socket with the socket() system call.

2.Bind the socket to an address using the bind() system call. For a server socket on the Internet, an address consists of a port number on the host machine.

3.Listen for connections with the listen() system call.

4.Accept a connection with the accept() system call. This call typically blocks until a client connects with the server.

5.After a connection is established, call fork() to create a new process.

6.The child process will close sockfd and call doprocessing function, passing the new socket file descriptor as an argument. When the two processes have completed their conversation, as indicated by doprocessing() returning, this process simply exits.

7.The parent process closes newsockfd. As all of this code is in an infinite loop, it will return to the accept statement to wait for the next connection.

Server Program

multiServer.c > doprocessing Format

```c
#include <stdio.h>
#include <stdlib.h>
#include<unistd.h>
#include <netdb.h>
#include <netinet/in.h>

#include <string.h>

void doprocessing(int sock);
void doprocessing(int sock) {
  int n;
  char buffer[256];
  bzero(buffer, 256);
  n = read(sock, buffer, 255);

  if (n < 0) {
    perror("ERROR reading from socket");
    exit(1);
  }

  printf("Here is the message: %s\n", buffer);
  n = write(sock, "I got your message", 18);

  if (n < 0) {
    perror("ERROR writing to socket");
    exit(1);
  }
}
```

Server Program

```c
int main(int argc, char *argv[]) {
  int sockfd, newsockfd, portno;
  socklen_t clilen;
  char buffer[256];
  struct sockaddr_in serv_addr, cli_addr;
  int n, pid;

  /* First call to socket() function */
  // End point communication creation
  sockfd = socket(AF_INET, SOCK_STREAM, 0);

  if (sockfd < 0) {
    perror("ERROR opening socket");
    exit(1);
  }

  /* Initialize socket structure */
  bzero((char *)&serv_addr, sizeof(serv_addr));
  portno = 5003;

  serv_addr.sin_family = AF_INET;
  serv_addr.sin_addr.s_addr = INADDR_ANY;
  serv_addr.sin_port = htons(portno);
```

Server Program Contd.

```
if (bind(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
  perror("ERROR on binding");
  exit(1);
}

listen(sockfd, 5);
clilen = sizeof(cli_addr);

while (1) {
  newsockfd = accept(sockfd, (struct sockaddr *)&cli_addr, &clilen);

  if (newsockfd < 0) {
    perror("ERROR on accept");
    exit(1);
  }


  pid = fork();

  if (pid < 0) {
    perror("ERROR on fork");
    exit(1);
  }
```

```
    pid = fork();

    if (pid < 0) {
      perror("ERROR on fork");
      exit(1);
    }

    if (pid == 0) {
      /* This is the client process */
      close(sockfd);
      doprocessing(newsockfd);
      exit(0);
    } else {
      close(newsockfd);
    }

  } /* end of while */
}
```

Making Client Process

Client Process is the process that is used to communicate with the service provider.

Basic Steps involved in the creation of Client Process

1.Create the Socket (Communication Endpoint for the client)

2.Now Connect the created socket with th server

3.If Successfully connected, two processes (Server-Client) can start communicating with each other, here we are using read and write api for communication

Client Program

```c
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char *argv[]) {
  int sockfd, portno, n;
  struct sockaddr_in serv_addr;
  struct hostent *server;

  char buffer[256];

  if (argc < 3) {
    printf("Hostname and the Port Number Not provided\n");
    return (0);
  }
  // Atoi converts string to integer
  portno = atoi(argv[2]);

  /* Create a socket point */
  sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

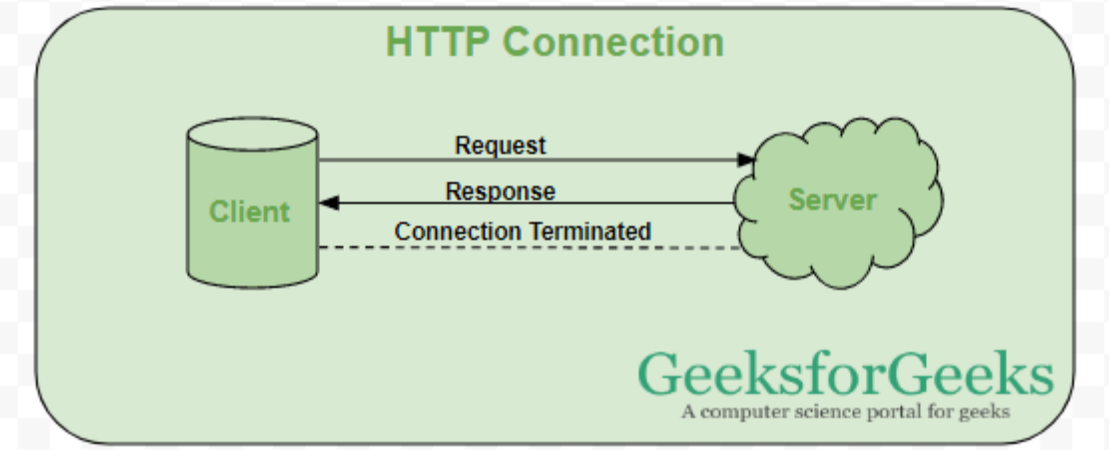
```c
  if (sockfd < 0) {
    perror("ERROR opening Client side socket");
    exit(1);
  }

  server = gethostbyname(argv[1]);

  if (server == NULL) {
    fprintf(stderr, "ERROR, no such host\n");
    exit(0);
  }

  bzero((char *)&serv_addr, sizeof(serv_addr));
  serv_addr.sin_family = AF_INET;
  bcopy((char *)server->h_addr, (char *)&serv_addr.sin_addr.s_addr,
        server->h_length);
  serv_addr.sin_port = htons(portno);

  /* Now connect to the server */
  if (connect(sockfd, (struct sockaddr *)&serv_addr,
sizeof(serv_addr)) < 0) {
    perror("ERROR connecting");
    exit(1);
  }
```

```c
  printf("Please enter the message: ");
  bzero(buffer, 256);
  fgets(buffer, 255, stdin);

  /* Send message to the server */
  n = write(sockfd, buffer, strlen(buffer));

  if (n < 0) {
    perror("ERROR writing to socket");
    exit(1);
  }

  /* Now read server response */
  bzero(buffer, 256);
  n = read(sockfd, buffer, 255);

  if (n < 0) {
    perror("ERROR reading from socket");
    exit(1);
  }

  printf("%s\n", buffer);
  return 0;
}
```

Applications

1.Web Development:

Web Servers and Clients: Socket programming is fundamental for handling HTTP requests and responses in web servers and clients. Browsers use sockets to request web pages, and servers use sockets to respond with the requested data.



2.Chat Applications:

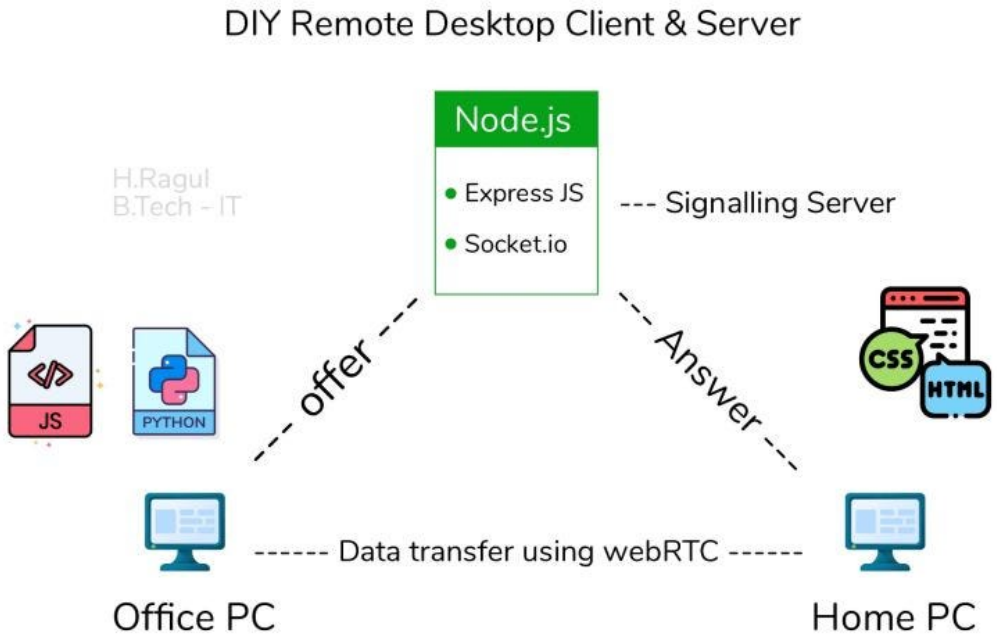
Real-time Messaging: Sockets enable instant messaging in chat applications. As users send messages, the messages are transmitted through sockets in real-time, allowing for seamless communication.

3.Remote Desktop Applications:

Screen Sharing and Control: Applications that allow remote desktop sharing and control use sockets for transmitting screen updates and user inputs between the client and server.

4.File Transfer Protocols:

FTP (File Transfer Protocol): FTP relies on sockets for transferring files between a client and a server over a network. Sockets facilitate the reliable and efficient exchange of file data.



5.Online Gaming:

Multiplayer Games: Socket programming is essential for creating multiplayer online games. It facilitates real-time communication between players, ensuring that actions and updates are quickly transmitted to all participants.

SnapShots


Shell
~/Socket-ProgrammingUNIX-PROJECT$ g++ -o server multiServer.c
~/Socket-ProgrammingUNIX-PROJECT$ ./server
Waiting for tht client to connect...
Client Connected
Here is the message: Hey there, this is from client 1
Waiting for new client to connect..
Shell
~/Socket-ProgrammingUNIX-PROJECT$ g++ -o client1 client.c
~/Socket-ProgrammingUNIX-PROJECT$ ./client1 localhost 5003
Please enter the message: Hey there, this is from client 1
I got your message
~/Socket-ProgrammingUNIX-PROJECT$
Shell
~/Socket-ProgrammingUNIX-PROJECT$


Shell
Waiting for tht client to connect...
Client Connected
Here is the message: Hey there, this is from client 1
Waiting for new client to connect..
Client Connected
Here is the message: Hey there, this is from client 2
Waiting for new client to connect..
Shell
~/Socket-ProgrammingUNIX-PROJECT$ g++ -o client1 client.c
~/Socket-ProgrammingUNIX-PROJECT$ ./client1 localhost 5003
Please enter the message: Hey there, this is from client 1
I got your message
~/Socket-ProgrammingUNIX-PROJECT$
Shell
~/Socket-ProgrammingUNIX-PROJECT$ g++ -o client2 client.c
~/Socket-ProgrammingUNIX-PROJECT$ ./client2 localhost 5003
Please enter the message: Hey there, this is from client 2
I got your message
~/Socket-ProgrammingUNIX-PROJECT$

References

1.Geeks for geek:
https://www.geeksforgeeks.org/socket-programming-cc/

2.Tutorials point:
https://www.tutorialspoint.com/unix_sockets/what_is_socket.htm