**IPC: Sockets**

**Subject - Unix Operating System**

**Name – Hemant Sharma**

**PRN – 22610001 Class – TYIT**

**Assignment No – 9(c)**

**Title-** Write two programs (server and client) to show how you can establish a TCP socket connection using the above functions.

**Objectives:**

1. To learn about fundamentals of IPC through C socket programming.
2. Learn and understand the OS interaction with socket programming.
3. Use of system call and IPC mechanism to write effective application programs.
4. To know the port numbering and process relation.
5. To knows the iterative and concurrent server concept.

**Theory:**

A very basic one-way Client and Server setup where a client connects, sends messages to server and the server shows them using socket connection. Java API networking package (java.net) takes care of all of that, making network programming very easy for programmers

CLIENT-SIDE PROGRAMMING:

Establish a Socket Connection

* To connect to other machine, we need a socket connection.
* A socket connection means the two machines have information about each other’s network location (IP Address) and TCP port. The java.net.Socket class represents a Socket.
* To open a socket: Socket socket = new Socket (“127.0.0.1”, 5000)

• First argument – IP address of Server. (127.0.0.1 is the IP address of localhost, where code will run on single stand-alone machine).

• Second argument – TCP Port. (Just a number representing which

application to run on a server. For example, HTTP runs on port 80.

Port number can be from 0 to 65535) To communicate over a socket

connection, streams are used to both input and output the data. Closing

the connection. The socket connection is closed explicitly once the

message to server is sent.

SERVER-SIDE PROGRAMMING:

Establish a Socket Connection

To write a server application two sockets are needed.

* A ServerSocket which waits for the client requests (when a client makes a new Socket())
* A plain old Socket socket to use for communication with the client getOutputStream() method is used to send the output through the socket. Close the Connection After finishing, it is important to close the connection by closing the socket as well as input/output streams

**Program:**

**Server:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 12345  // Port number to use for the connection

void handle\_client(int client\_socket) {

    char buffer[1024];

    int read\_size;

    // Receive a message from client

    read\_size = read(client\_socket, buffer, sizeof(buffer) - 1);

    if (read\_size < 0) {

        perror("Error reading from client");

        return;

    }

    buffer[read\_size] = '\0';  // Null-terminate the string

    printf("Received message from client: %s\n", buffer);

    // Send a response to the client

    char response[] = "Message received successfully!";

    write(client\_socket, response, strlen(response));

    // Close the client socket

    close(client\_socket);

}

int main() {

    int server\_socket, client\_socket;

    struct sockaddr\_in server\_addr, client\_addr;

    socklen\_t client\_len = sizeof(client\_addr);

    // Create the server socket

    if ((server\_socket = socket(AF\_INET, SOCK\_STREAM, 0)) == -1) {

        perror("Socket creation failed");

        exit(1);

    }

    // Zero out the structure and set up the server address

    memset(&server\_addr, 0, sizeof(server\_addr));

    server\_addr.sin\_family = AF\_INET;

    server\_addr.sin\_addr.s\_addr = INADDR\_ANY;  // Listen on all interfaces

    server\_addr.sin\_port = htons(PORT);

    // Bind the socket to the address and port

    if (bind(server\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {

        perror("Bind failed");

        close(server\_socket);

        exit(1);

    }

    // Start listening for incoming connections

    if (listen(server\_socket, 5) == -1) {

        perror("Listen failed");

        close(server\_socket);

        exit(1);

    }

    printf("Server is listening on port %d...\n", PORT);

    while (1) {

        // Accept a new client connection

        client\_socket = accept(server\_socket, (struct sockaddr \*)&client\_addr, &client\_len);

        if (client\_socket == -1) {

            perror("Accept failed");

            continue;

        }

        // Handle the client (iterative server: one client at a time)

        handle\_client(client\_socket);

    }

    // Close the server socket

    close(server\_socket);

    return 0;

}

**Client:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define SERVER\_IP "127.0.0.1"  // Localhost

#define SERVER\_PORT 12345      // Port number to connect to

int main() {

    int client\_socket;

    struct sockaddr\_in server\_addr;

    char message[1024], server\_response[1024];

    // Create the client socket

    if ((client\_socket = socket(AF\_INET, SOCK\_STREAM, 0)) == -1) {

        perror("Socket creation failed");

        exit(1);

    }

    // Set up the server address structure

    memset(&server\_addr, 0, sizeof(server\_addr));

    server\_addr.sin\_family = AF\_INET;

    server\_addr.sin\_port = htons(SERVER\_PORT);

    if (inet\_pton(AF\_INET, SERVER\_IP, &server\_addr.sin\_addr) <= 0) {

        perror("Invalid address");

        exit(1);

    }

    // Connect to the server

    if (connect(client\_socket, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {

        perror("Connect failed");

        exit(1);

    }

    // Get input from user and send to server

    printf("Enter message to send to server: ");

    fgets(message, sizeof(message), stdin);

    send(client\_socket, message, strlen(message), 0);

    // Receive response from server

    int read\_size = recv(client\_socket, server\_response, sizeof(server\_response) - 1, 0);

    if (read\_size > 0) {

        server\_response[read\_size] = '\0';  // Null-terminate the string

        printf("Server response: %s\n", server\_response);

    } else {

        perror("Receive failed");

    }

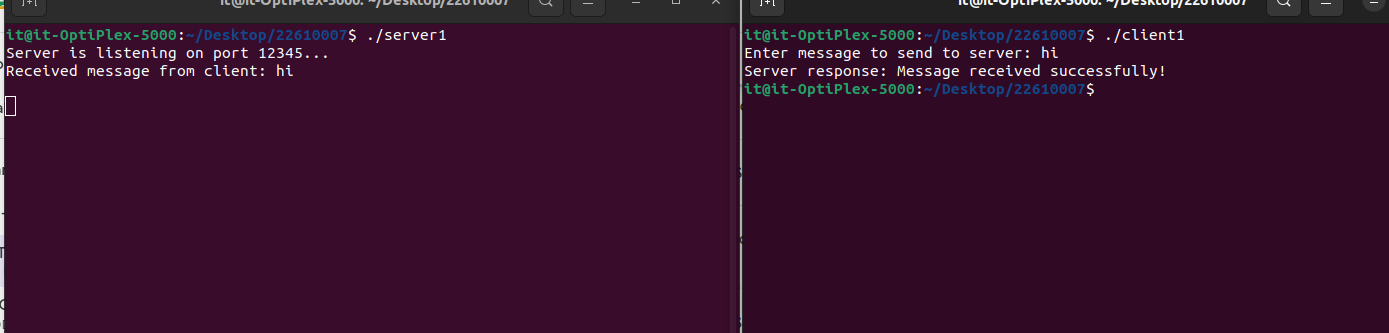
    // Close the client socket

    close(client\_socket);

    return 0;

}

**Output:**

****

**Conclusion:**

TCP socket connection using system calls in C studied and client server connection established.