**IPC: Sockets**

**Subject - Unix Operating System**

**Name – Hemant Sharma**

**PRN – 22610001 Class – TYIT**

**Assignment No – 9(d)**

**Title-** Write two programs (server and client) to show how you can establish a UDP 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

#define MAX\_MSG\_SIZE 1024

int main() {

    int server\_socket;

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

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

    char buffer[MAX\_MSG\_SIZE];

    int recv\_len;

    // Create a UDP socket

    if ((server\_socket = socket(AF\_INET, SOCK\_DGRAM, 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);

    }

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

    while (1) {

        // Receive a message from the client

        recv\_len = recvfrom(server\_socket, buffer, sizeof(buffer), 0, (struct sockaddr \*)&client\_addr, &client\_len);

        if (recv\_len == -1) {

            perror("Receive failed");

            continue;

        }

        buffer[recv\_len] = '\0';  // Null-terminate the received string

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

        // Send a response back to the client

        char response[] = "Message received!";

        if (sendto(server\_socket, response, strlen(response), 0, (struct sockaddr \*)&client\_addr, client\_len) == -1) {

            perror("Send failed");

        }

    }

    // 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"  // IP address of the server

#define SERVER\_PORT 12345      // Port number the server is listening on

#define MAX\_MSG\_SIZE 1024

int main() {

    int client\_socket;

    struct sockaddr\_in server\_addr;

    char message[MAX\_MSG\_SIZE], server\_response[MAX\_MSG\_SIZE];

    // Create the UDP socket

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

        perror("Socket creation failed");

        exit(1);

    }

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

    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);

    }

    // Get input from the user and send it to the server

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

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

    if (sendto(client\_socket, message, strlen(message), 0, (struct sockaddr \*)&server\_addr, sizeof(server\_addr)) == -1) {

        perror("Send failed");

        close(client\_socket);

        exit(1);

    }

    // Receive the response from the server

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

    int recv\_len = recvfrom(client\_socket, server\_response, sizeof(server\_response) - 1, 0, (struct sockaddr \*)&server\_addr, &server\_len);

    if (recv\_len == -1) {

        perror("Receive failed");

        close(client\_socket);

        exit(1);

    }

    server\_response[recv\_len] = '\0';  // Null-terminate the response

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

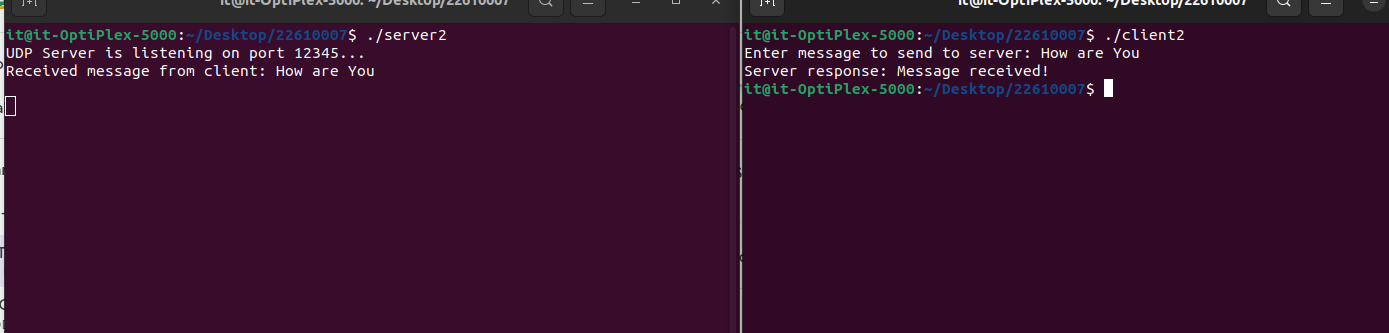
    // Close the socket

    close(client\_socket);

    return 0;

}

**Output:**

****

**Conclusion:**

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