**IPC: Sockets**

**Subject - Unix Operating System**

**Name – Hemant Sharma**

**PRN – 22610001 Class – TYIT**

**Assignment No – 9(b)**

**Title-** Write programs (server and client) to implement concurrent/iterative server to connect multiple client’s requests handled through concurrent/iterative logic using UDP/TCP socket connection.

**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

**Data Dictionary:**

|  |  |  |  |
| --- | --- | --- | --- |
| **SR. NO.** | **Variable/Function** | **Data Type** | **Use** |
| 1 | ss | ServerSocket | Create a socket for server side  communication. |
| 2 | s | Socket | Socket is created |
| 3 | dos | DataOutputStream | Output Stream |
| 4 | dis | DataInputStream | Input Stream |
| 5 | str | String | String to display message from  clients. |
| 6 | br | BufferedReader | Input Data |

**Program:**

**Server:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <arpa/inet.h>

#define PORT 12345

void handle\_client(int client\_socket) {

    char buffer[1024];

    // Read the message from the client

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

    if (read\_size > 0) {

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

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

        // Send a response to the client

        char response[] = "Message received!";

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

    }

    close(client\_socket);  // Close the client connection

}

int main() {

    int server\_socket, client\_socket;

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

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

    // Create a TCP socket

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

        perror("Socket creation failed");

        exit(1);

    }

    // Set up the server address structure

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

    server\_addr.sin\_addr.s\_addr = INADDR\_ANY;

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

    // Bind the socket to the address

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

        perror("Bind failed");

        close(server\_socket);

        exit(1);

    }

    // Listen for incoming connections

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

        perror("Listen failed");

        close(server\_socket);

        exit(1);

    }

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

    while (1) {

        // Accept incoming client connections

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

        if (client\_socket == -1) {

            perror("Accept failed");

            continue;

        }

        printf("Client connected\n");

        // Fork a new process to handle the client

        pid\_t pid = fork();

        if (pid == 0) {  // Child process

            close(server\_socket);  // Child doesn't need the server socket

            handle\_client(client\_socket);  // Handle client

            exit(0);  // End child process

        } else if (pid < 0) {

            perror("Fork failed");

            close(client\_socket);

        } else {

            close(client\_socket);  // Parent doesn't need the client 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"

#define SERVER\_PORT 12345

int main() {

    int sock;

    struct sockaddr\_in server\_addr;

    char message[1024];

    char response[1024];

    // Create socket

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

        perror("Socket creation failed");

        exit(1);

    }

    // Set up the server address structure

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

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

    server\_addr.sin\_addr.s\_addr = inet\_addr(SERVER\_IP);

    // Connect to the server

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

        perror("Connect failed");

        close(sock);

        exit(1);

    }

    // Get message from the user

    printf("Enter message for the server: ");

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

    // Send message to the server

    write(sock, message, strlen(message));

    // Read the server's response

    int read\_size = read(sock, response, sizeof(response)-1);

    if (read\_size > 0) {

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

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

    }

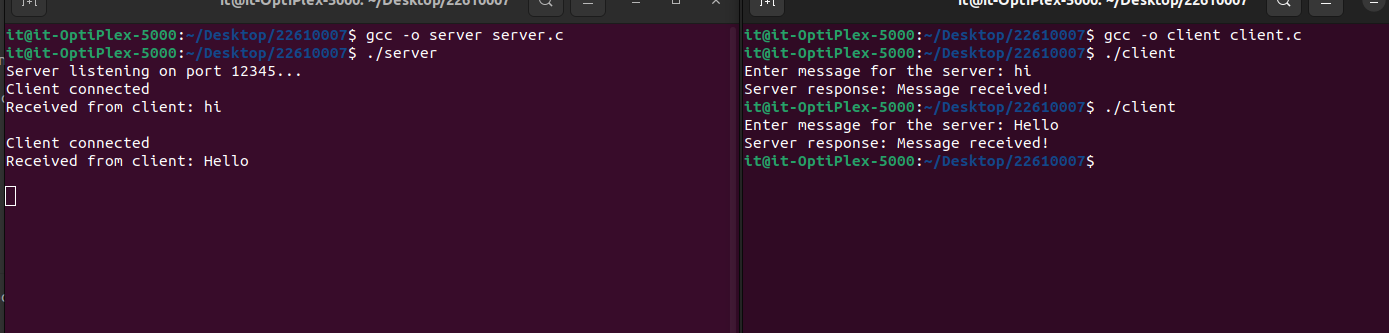
    // Close the socket

    close(sock);

    return 0;

}

**Output:**

****

**Conclusion:**

Various communication protocols like TCP/UDP can be implemented using socket programming in Java to serve requests from multiple clients.