EXPERIMENT13

INTRODUCTIONTOSOCKETPROGRAMMING

OBJECTIVE:

- Learn and Understand Communication using socket programming

BACKGROUND:

For C++:
Command: g++ *source_files…* -o *output_file* For
C:
Command: gcc*source_files… -o output_file*
Source files need not be cpp or c files. They can be preprocessed files, assembly files, or object files.

The whole compilation file works in the following way:
Cpp/c file(s) >> Preprocessed file(s) >>Assembly File(s) Generation>> Object file(s) Generation >> Final Executable

Every c/cpp file has its own preprocessed file, assembly file, and object file.

1. For running only the preprocessor, we use -E option.
2. For running the compilation process till assembly file generation, we use –S option.
3. For running the compilation process till object file creation, we use –c option.
4. If no option is specified, the whole compilation process till the generation of executable will run.

A file generated using any option can be used to create the final executable. For example, let's suppose that we have two source files: math.cpp and main.cpp, and we create object files:
g++ main.cpp –c –o main.o g++
math.cpp –c –o math.o

The object files created using above two commands can be used to generate the final executable. *g++ main.omath.o –o my_executable*
The file named "my_executable" is the final exe file. There is specific extension for executable files in Linux.

Socket:

A socket is defined as an endpoint for communication. A pair of processes communicating over a network employs a pair of sockets—one for each process. A socket is identified by an IP address concatenated with a port number. In general, sockets use a client–server architecture. The server waits for incoming client requests by listening to a specified port. Once a request is received, the server accepts a connection from the client socket to complete the connection. Servers implementing specific services (such as telnet, FTP, and HTTP) listen to well-known ports (a telnet server listens to port 23; an FTP server listens to port 21;

and a web, or HTTP, server listens to port 80). All ports below 1024 are considered well known; we can use them to implement standard services.

The communication over the network in TCP/IP model takes place in form of a client server architecture. ie, the client begins the communication and server follows up and a connection is established. Sockets can be used in many languages like Java, C++ , Python etc but here in this lab, we will understand the socket communication in its purest form (i.e in C programming language)

Functions used in Socket Programming:

socket() Endpoint for communication
bind() Assign a unique telephone number listen()
Wait for a caller connect() Dial a
number accept() Receive a call
send(), recv() Talk
close() Hang up

Servercode:

```
#include <sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<stdio.h>
#include <unistd.h> int
main()
{ intsd; char* msg="connected";
          char
          msg1[100];
          structsockaddr_inmy_addr, client_addr;
          my_addr.sin_family=AF_INET;
          my_addr.sin_port=htons(4999);
          my_addr.sin_addr.s_addr=inet_addr("127.0.0.1");
          sd=socket(AF_INET,SOCK_STREAM,0);
          bind(sd,(structsockaddr *) &my_addr,sizeof(struct
sockaddr_in)); listen(sd,5); inti=0; while(i<2)
      { int size=sizeof(structsockaddr);
            intnew_sd=accept(sd,(structsockaddr*)&client_addr,&size);
            send(new_sd, msg,100,0); recv(new_sd,msg1,100,0);
            printf("%s\n",msg1);
            ++i;
      } return
      0;
}
```

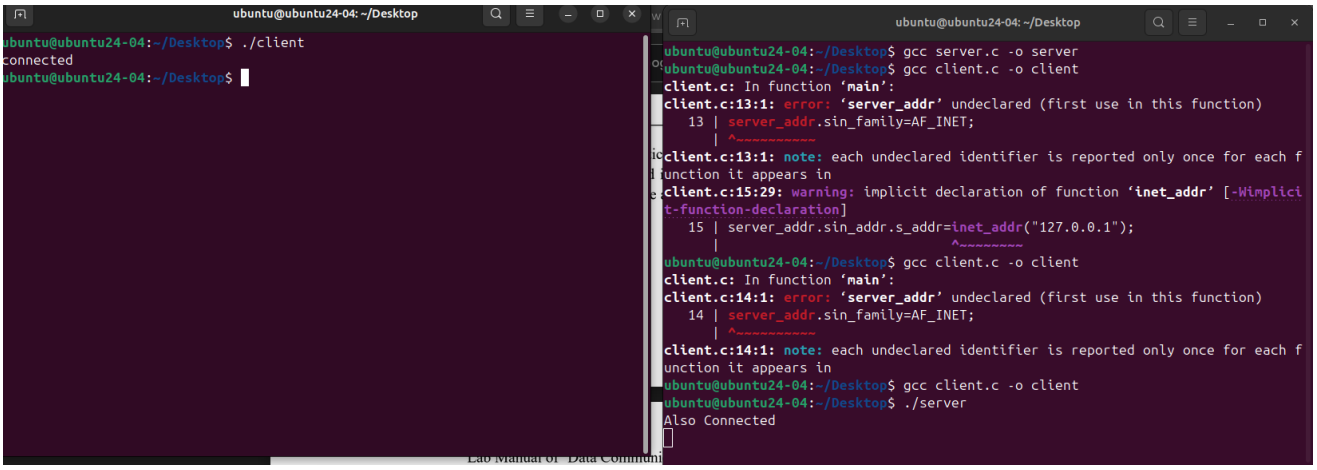
Client Code

```
#include <sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<stdio.h>
#include <unistd.h>
```

```
int main()
{
        Intsd; char
        msg[100];
        char* msg1="Also Connected";
        structsockaddr_inserver_addr;
        server_addr.sin_family=AF_INET;
        server_addr.sin_port=htons(4999);
        server_addr.sin_addr.s_addr=inet_addr("127.0.0.1");
        sd=socket(AF_INET,SOCK_STREAM,0);
        connect(sd,(structsockaddr*)
        &server_addr,sizeof(structsockaddr));
        recv(sd,msg,100,0); send(sd,
        msg1,100,0); printf("%s\n",msg);
        return 0;
}
```

TASK#1

Run the code and then comment the whole code in your own words in the space given below. Compile both client and server using GCC in Ubuntu terminal. First run the server. Open a new terminal window and run the clients in the second terminal.



TASK#2

Modify the server and client code to make an authentication method to establish the connection with client. The client will send a username and password if both are correct the further connection will continue else the connection will stop. Paste your code and output in the space provided below.

SERVER

Mudassar Hussain BSEE 2023 FAST NU LHR <l236006@lhr.nu.edu.pk>


```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>

int main() {
    int sd;
    char *welcome_msg = "Connected";
    char username[50], password[50];
    char auth_success[] = "Authentication Successful";
    char auth_fail[] = "Authentication Failed. Connection Closed.";

    struct sockaddr_in my_addr, client_addr;

    // Server address setup
    my_addr.sin_family = AF_INET;
    my_addr.sin_port = htons(4999);
    my_addr.sin_addr.s_addr = inet_addr("127.0.0.1");

    // Create socket
    sd = socket(AF_INET, SOCK_STREAM, 0);

    // Bind socket
    bind(sd, (struct sockaddr *)&my_addr, sizeof(my_addr));

    // Listen for connections
    listen(sd, 5);

    printf("Server is running and waiting for client...\n");

    while (1) {
        int size = sizeof(client_addr);
        int new_sd = accept(sd, (struct sockaddr *)&client_addr, &size);

        // Receive username and password
        recv(new_sd, username, sizeof(username), 0);
        recv(new_sd, password, sizeof(password), 0);

        // Check credentials
        if (strcmp(username, "admin") == 0 && strcmp(password, "1234") == 0) {
            send(new_sd, auth_success, sizeof(auth_success), 0);
```

```
            // Further communication
            char client_msg[100];
            send(new_sd, welcome_msg, sizeof(welcome_msg), 0);
            recv(new_sd, client_msg, sizeof(client_msg), 0);
            printf("Client says: %s\n", client_msg);
        } else {
            send(new_sd, auth_fail, sizeof(auth_fail), 0);
        }

        close(new_sd); // Close connection
    }

    close(sd); // Close server socket
    return 0;
}
```

Client

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <string.h>

int main() {
    int sd;
    struct sockaddr_in server_addr;
    char username[50], password[50];
    char server_msg[100];

    // Input username and password
    printf("Enter username: ");
    scanf("%s", username);
    printf("Enter password: ");
    scanf("%s", password);

    // Server address setup
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = htons(4999);
    server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");

    // Create socket
    sd = socket(AF_INET, SOCK_STREAM, 0);

    // Connect to server
    if (connect(sd, (struct sockaddr *)&server_addr, sizeof(server_addr)) < 0) {
        printf("Connection failed.\n");
```

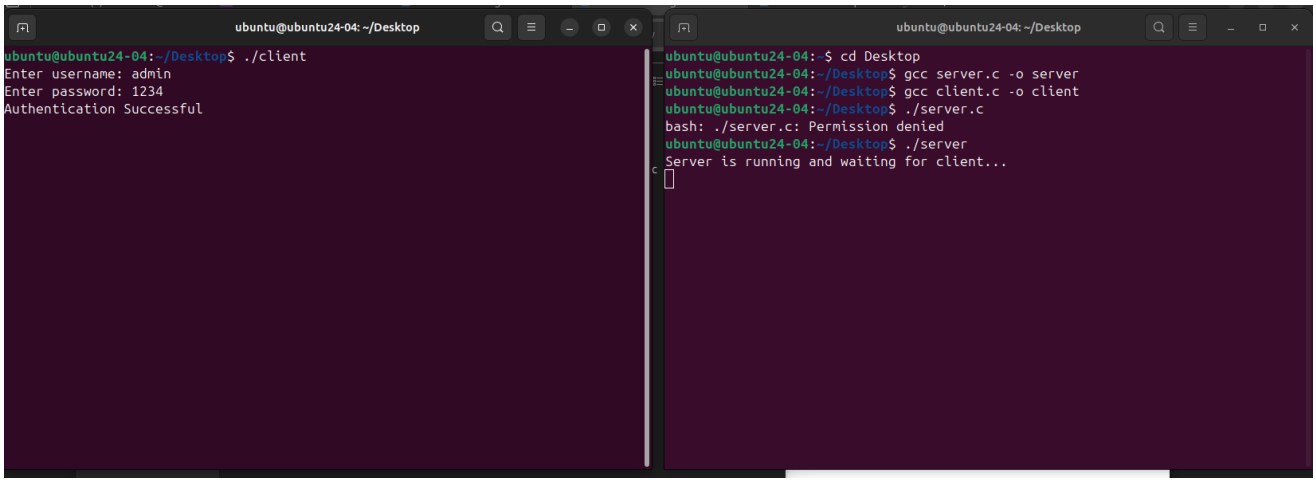
```
        return 1;
    }

    // Send username and password
    send(sd, username, sizeof(username), 0);
    send(sd, password, sizeof(password), 0);

    // Receive authentication response
    recv(sd, server_msg, sizeof(server_msg), 0);
    printf("%s\n", server_msg);

    // If authentication successful, continue communication
    if (strstr(server_msg, "Successful") != NULL) {
        char msg_to_server[] = "Hello Server!";
        recv(sd, server_msg, sizeof(server_msg), 0);
        printf("Server says: %s\n", server_msg);
        send(sd, msg_to_server, sizeof(msg_to_server), 0);
    }

    close(sd);
    return 0;
}
```



POST LAB QUESTIONS:

- The above code is making a TCP Connection convert it into UDP and paste your output in provided space.