## **National University of Computer and Emerging Sciences**



## **Laboratory Exercise 03**

*for*

# Computer Networks

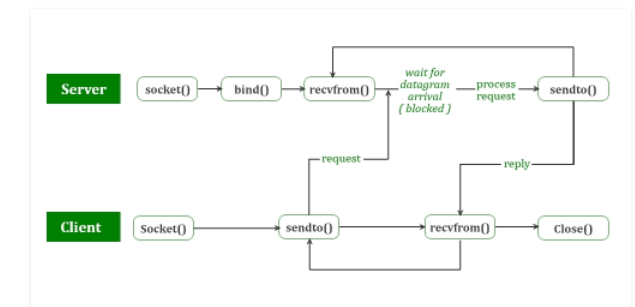
|  |
| --- |
| Objectives:  * Socket Programming * Client Server connectivity * UDP-model |

**Note: Carefully read the following instructions (***Each instruction contains a weightage***)**

1. Make a Microsoft Word file and paste all of your code with all possible screenshots of every **task output and submit on Google classroom within given time.**
2. Please submit your file in this format rollno\_Name.
3. Do not submit your assignment after deadline.
4. Do not copy code from any source otherwise you will be penalized with negative marks.

**FAST School of Computer Science**

## **UDP Client Server Communication**



**UDP Server:**

1. Create a UDP socket.
2. Bind the socket to the server address.
3. Wait until the datagram packet arrives from the client.
4. Process the datagram packet and send a reply to the client.
5. Go back to Step 3.

**UDP Client:**

1. Create a UDP socket.
2. Send a message to the server.
3. Wait until response from the server is received.
4. Process reply and go back to step 2, if necessary.
5. Close socket descriptor and exit.

**It is called connection less protocol of Network Communication.**

## **Lab Task 1:**

Implement the following UDP Client Server architecture into your Ubuntu Linux and create a **UDP chat server** then add the screenshot of the output.

1. Comment Each Construct in your Code (You are allowed to use internet)
2. Describe the purpose of each line using single line comment i.e.

// Comments…

Note: Comments font Color should be green.

1. Compare both TCP (use your previous knowledge) and UDP server and client-side code implementation in details.

## **Lab Task 2:**

Develop a Client-Server application using UDP where the client will send two file names to the server and the server will copy the content of the first file into the second file. After copying is done, the server will send appropriate message to the client. The second file should be initially empty. Assume that both files are present at the server.

## **Lab Task 3:**

Develop a Client-Server application using UDP where the client will send a decimal integer to the server and the server will calculate the sum of its even positioned digits and send back the result to the client. The client will display the result. [Example: Input: 1248, Output: 2+8=10]

## **Lab Task 4:**

Given a text file with 10 sentences as input, write a client server C program where client encrypts the text file using bitwise operations (as a key) at the client side and sends the encrypted file to the server. Server program will take the key as the input and will decrypt the original file.

**Source codes:**

**server.c**

// Server side implementation of UDP client-server model

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <netinet/in.h>

#define PORT     8080

#define MAXLINE 1024

// Driver code

int main() {

    int sockfd;

    char buffer[MAXLINE];

    char \*hello = "Hello from server";

    struct sockaddr\_in servaddr, cliaddr;

    // Creating socket file descriptor

    if ( (sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0 ) {

        perror("socket creation failed");

        exit(EXIT\_FAILURE);

    }

    memset(&servaddr, 0, sizeof(servaddr));

    memset(&cliaddr, 0, sizeof(cliaddr));

    // Filling server information

    servaddr.sin\_family    = AF\_INET; // IPv4

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

    servaddr.sin\_port = htons(PORT);

    // Bind the socket with the server address

    if ( bind(sockfd, (const struct sockaddr \*)&servaddr,

            sizeof(servaddr)) < 0 )

    {

        perror("bind failed");

        exit(EXIT\_FAILURE);

    }

    int len, n;

    len = sizeof(cliaddr);  //len is value/result

    n = recvfrom(sockfd, (char \*)buffer, MAXLINE,

                MSG\_WAITALL, ( struct sockaddr \*) &cliaddr,

                &len);

    buffer[n] = '\0';

    printf("Client : %s\n", buffer);

    sendto(sockfd, (const char \*)hello, strlen(hello),

        MSG\_CONFIRM, (const struct sockaddr \*) &cliaddr,

            len);

    printf("Hello message sent.\n");

    return 0;

}

**Source Codes:**

**client.c**

// Client side implementation of UDP client-server model

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <netinet/in.h>

#define PORT     8080

#define MAXLINE 1024

// Driver code

int main() {

    int sockfd;

    char buffer[MAXLINE];

    char \*hello = "Hello from client";

    struct sockaddr\_in     servaddr;

    // Creating socket file descriptor

    if ( (sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0 ) {

        perror("socket creation failed");

        exit(EXIT\_FAILURE);

    }

    memset(&servaddr, 0, sizeof(servaddr));

    // Filling server information

    servaddr.sin\_family = AF\_INET;

    servaddr.sin\_port = htons(PORT);

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

    int n, len;

    sendto(sockfd, (const char \*)hello, strlen(hello),

        MSG\_CONFIRM, (const struct sockaddr \*) &servaddr,

            sizeof(servaddr));

    printf("Hello message sent.\n");

    n = recvfrom(sockfd, (char \*)buffer, MAXLINE,

                MSG\_WAITALL, (struct sockaddr \*) &servaddr,

                &len);

    buffer[n] = '\0';

    printf("Server : %s\n", buffer);

    close(sockfd);

    return 0;

}

Clearly perform each step on your word document from scratch to end and attached the proper screenshot of each performed step (If needed).

## **BEST WISHES**