Name: Pratik Kashinath Darange

Division: A(A3)

Roll No: 60

COMPUTER NETWORKS PRACTICAL NO. VII

(SOCKET PROGRAMMING)

1. Concept of Socket :

- A socket is one endpoint of a two-way communication channel between two networked programmes. By creating named contact points between which the communication takes place, the socket mechanism offers a form of inter-process communication (IPC).

- Similar to how pipes are generated using the "Pipe" system call and sockets using the "socket" system call. The socket offers network-based bidirectional FIFO communication. At each end of the communication, a socket is established that connects to the network. Each socket is assigned a unique address. This address is made up of a port number and an IP address.

- Typically, client-server programmes use sockets. The server establishes a socket, connects it to a network port, and then watches for a client request to connect. After creating a socket, the client tries to connect it to the server socket. Data transfer starts as soon as the connection is made.

Significance :

- Sockets are useful for both stand-alone and network applications. Sockets allow you to exchange information between processes on the same machine or across a network, distribute work to the most efficient machine, and they easily allow access to centralized data.
- Socket application program interfaces (APIs) are the network standard for TCP/IP.
- A wide range of operating systems support socket APIs. i5/OS sockets support multiple transport and networking protocols.

2. Transport layer protocols - TCP and UDP:

TCP :

- Transmission Control Protocol is referred to as TCP.
- Applications can access all transport layer services using it.
- Since the link between the two ends of the transmission must be established, it is a connection-oriented protocol. TCP creates a virtual circuit between the sender and receiver for the duration of a transmission in order to establish the connection.

UDP (User Datagram Protocol):

- UDP is a simple protocol and it provides non-sequenced transport functionality.
- UDP is a connectionless protocol.
- This type of protocol is used when reliability and security are less important than speed and size.
- UDP is an end-to-end transport level protocol that adds transport-level addresses, checksum error control, and length information to the data from the upper layer.
- The packet produced by the UDP protocol is known as a user datagram.

3. Application of TCP and UDP :

TCP

Stream data transfer: TCP protocol transfers data in the form of a stream of bytes. Bytes are grouped into TCP segments by TCP, which is subsequently given to the IP layer for transmission to the destination. TCP divides the data and sends it to the IP.

Reliability: TCP gives each byte it transmits a sequence number and anticipates a positive acknowledgement from the receiving TCP. The data is sent again to the destination if ACK is not received within a timeout period.
The receiving TCP uses the sequence number to either remove duplicate segments or to reassemble the segments if they arrive out of order.

Flow Control: Upon receipt, TCP notifies the sender in an acknowledgement how many bytes it may receive before exceeding its internal buffer. The number of bytes is transmitted in the ACK in the form of the largest sequence number it is capable of receiving without encountering any issues. This device is additionally known as a window mechanism.

Multiplexing is the technique of receiving data from various programmes and sending it to those applications running on various computers. The right application receives the data at the other end. The action in question is called demultiplexing. TCP uses the logical channels known as ports to transmit the packet to the appropriate application.

Logical Connections: A logical connection is made up of a set of sockets, sequence numbers, and window sizes. The pair of sockets each connection uses for sending and receiving data serves as its unique identification.

UDP

- It uses small packet size with small header (8 bytes). This fewer bytes in the overhead makes UDP protocol need less time in processing the packet and need less memory.
- It does not require connection to be established and maintained. Also absence of acknowledgement field in UDP makes it faster as it need not have to wait on ACK or need not have to hold data in memory until they are ACKed.
- It uses checksum with all the packets for error detection.
  It can be used in events where a single packet of data needs to be exchanged between the hosts.

CODE:

Server.c code (server program)

```
#include <arpa/inet.h>
#include <errno.h>
#include <netinet/in.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
#define PORT 5000
#define MAXLINE 1024
int max(int x, int y)
{
    if (x > y)
        return x;
    else
        return y;
}
int main()
{
    int listenfd, connfd, udpfd, nready, maxfdp1;
    char buffer[MAXLINE];
    pid_t childpid;
    fd_set rset;
    ssize_t n;
    socklen_t len;
    const int on = 1;
    struct sockaddr_in cliaddr, servaddr;
    char* message = "Hello Client";
    void sig_chld(int);

    /* create listening TCP socket */
    listenfd = socket(AF_INET, SOCK_STREAM, 0);
    bzero(&servaddr, sizeof(servaddr));
    servaddr.sin_family = AF_INET;
    servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
    servaddr.sin_port = htons(PORT);

    // binding server addr structure to listenfd
    bind(listenfd, (struct sockaddr*)&servaddr, sizeof(servaddr));
    listen(listenfd, 10);

    /* create UDP socket */
```

```c
    udpfd = socket(AF_INET, SOCK_DGRAM, 0);
    // binding server addr structure to udp sockfd
    bind(udpfd, (struct sockaddr*)&servaddr, sizeof(servaddr));

    // clear the descriptor set
    FD_ZERO(&rset);

    // get maxfd
    maxfdp1 = max(listenfd, udpfd) + 1;
    for (;;) {

        // set listenfd and udpfd in readset
        FD_SET(listenfd, &rset);
        FD_SET(udpfd, &rset);

        // select the ready descriptor
        nready = select(maxfdp1, &rset, NULL, NULL, NULL);

        // if tcp socket is readable then handle
        // it by accepting the connection
        if (FD_ISSET(listenfd, &rset)) {
            len = sizeof(cliaddr);
            connfd = accept(listenfd, (struct sockaddr*)&cliaddr, &len);
            if ((childpid = fork()) == 0) {
                close(listenfd);
                bzero(buffer, sizeof(buffer));
                printf("Message From TCP client: ");
                read(connfd, buffer, sizeof(buffer));
                puts(buffer);
                write(connfd, (const char*)message, sizeof(buffer));
                close(connfd);
                exit(0);
            }
            close(connfd);
        }
        // if udp socket is readable receive the message.
        if (FD_ISSET(udpfd, &rset)) {
            len = sizeof(cliaddr);
            bzero(buffer, sizeof(buffer));
            printf("\nMessage from UDP client: ");
            n = recvfrom(udpfd, buffer, sizeof(buffer), 0,
                         (struct sockaddr*)&cliaddr, &len);
            puts(buffer);
            sendto(udpfd, (const char*)message, sizeof(buffer), 0,
                (struct sockaddr*)&cliaddr, sizeof(cliaddr));
        }
    }
}
```

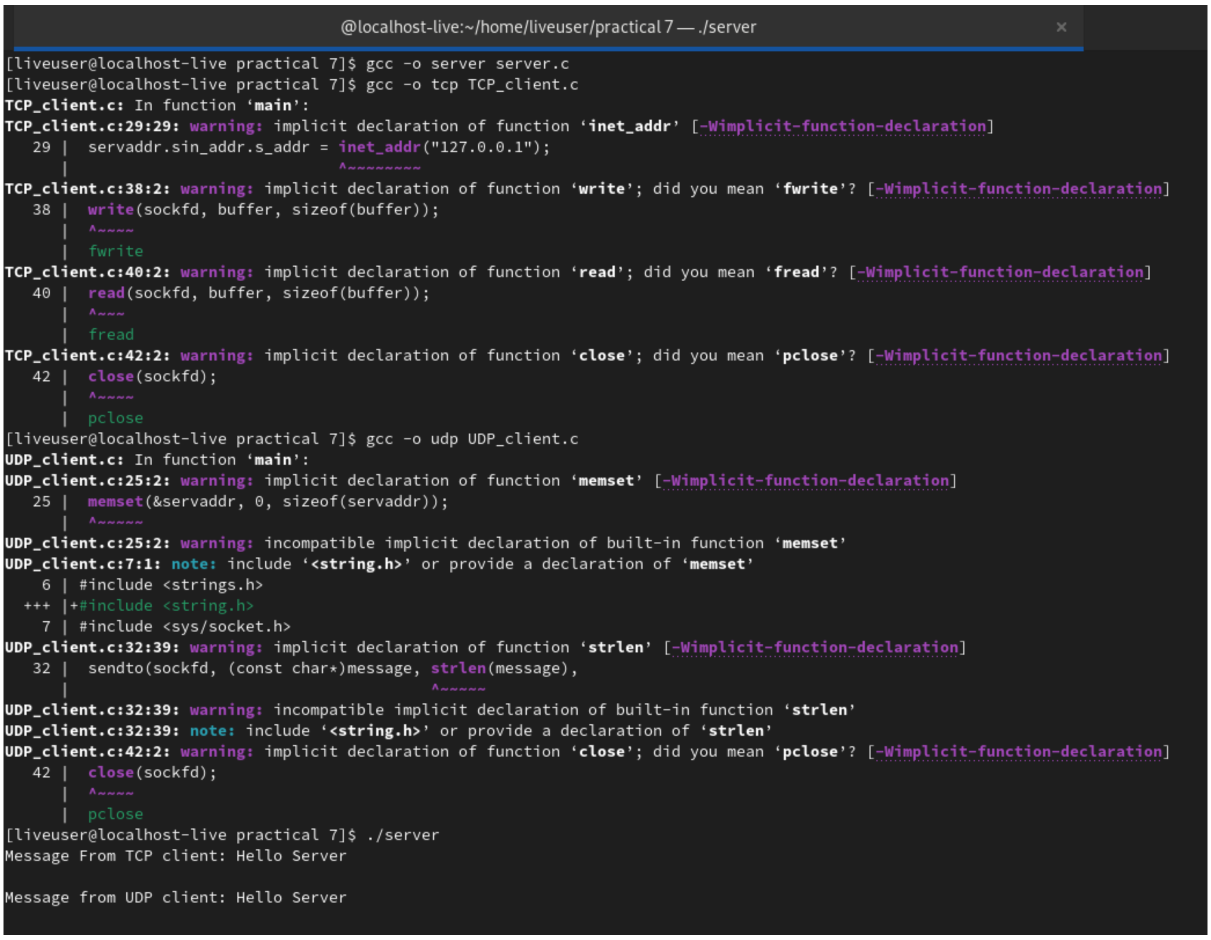
TCP_Client.c

```c
// TCP Client program
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#define PORT 5000
#define MAXLINE 1024
int main()
{
    int sockfd;
    char buffer[MAXLINE];
    char* message = "Hello Server";
    struct sockaddr_in servaddr;

    int n, len;
    // Creating socket file descriptor
    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("socket creation failed");
        exit(0);
    }

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

    // Filling server information
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(PORT);
    servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");

    if (connect(sockfd, (struct sockaddr*)&servaddr,
                              sizeof(servaddr)) < 0) {
        printf("\n Error : Connect Failed \n");
    }

    memset(buffer, 0, sizeof(buffer));
    strcpy(buffer, "Hello Server");
    write(sockfd, buffer, sizeof(buffer));
    printf("Message from server: ");
    read(sockfd, buffer, sizeof(buffer));
    puts(buffer);
    close(sockfd);
}
```

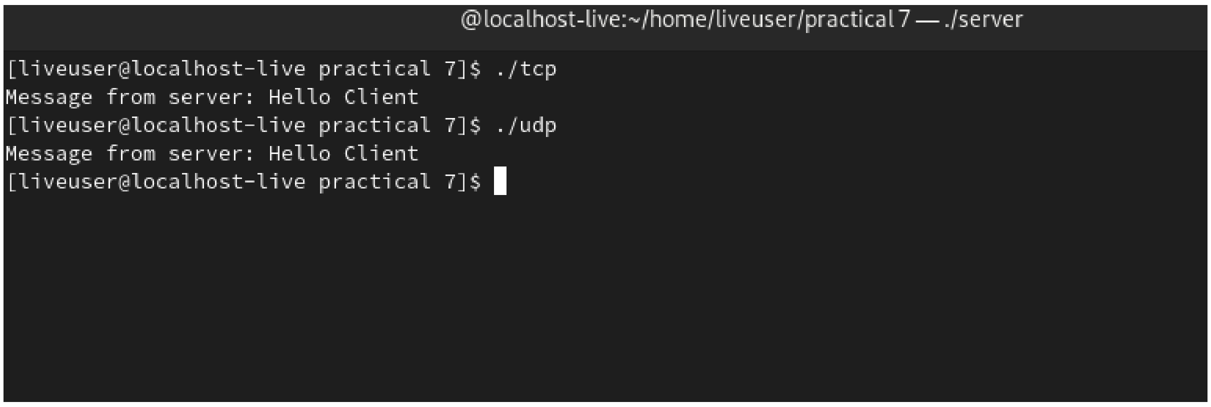
UDP_client.c

```c
// UDP client program
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <sys/socket.h>
#include <sys/types.h>
#define PORT 5000
#define MAXLINE 1024
int main()
{
    int sockfd;
    char buffer[MAXLINE];
    char* message = "Hello Server";
    struct sockaddr_in servaddr;

    int n, len;
    // Creating socket file descriptor
    if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0) {
        printf("socket creation failed");
        exit(0);
    }

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

    // Filling server information
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(PORT);
    servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
    // send hello message to server
    sendto(sockfd, (const char*)message, strlen(message),
        0, (const struct sockaddr*)&servaddr,
        sizeof(servaddr));

    // receive server's response
    printf("Message from server: ");
    n = recvfrom(sockfd, (char*)buffer, MAXLINE,
                0, (struct sockaddr*)&servaddr,
                &len);
    puts(buffer);
    close(sockfd);
    return 0;
}
```

OUTPUT:



```
[liveuser@localhost-live practical 7]$ gcc -o server server.c
[liveuser@localhost-live practical 7]$ gcc -o tcp TCP_client.c
TCP_client.c: In function ‘main’:
TCP_client.c:29:29: warning: implicit declaration of function ‘inet_addr’ [-Wimplicit-function-declaration]
   29 |   servaddr.sin_addr.s_addr = inet_addr("127.0.0.1");
      |                              ^~~~~~~~~
TCP_client.c:38:2: warning: implicit declaration of function ‘write’; did you mean ‘fwrite’? [-Wimplicit-function-declaration]
   38 |   write(sockfd, buffer, sizeof(buffer));
      |   ^~~~~
      |   fwrite
TCP_client.c:40:2: warning: implicit declaration of function ‘read’; did you mean ‘fread’? [-Wimplicit-function-declaration]
   40 |   read(sockfd, buffer, sizeof(buffer));
      |   ^~~~
      |   fread
TCP_client.c:42:2: warning: implicit declaration of function ‘close’; did you mean ‘pclose’? [-Wimplicit-function-declaration]
   42 |   close(sockfd);
      |   ^~~~~
      |   pclose
[liveuser@localhost-live practical 7]$ gcc -o udp UDP_client.c
UDP_client.c: In function ‘main’:
UDP_client.c:25:2: warning: implicit declaration of function ‘memset’ [-Wimplicit-function-declaration]
   25 |   memset(&servaddr, 0, sizeof(servaddr));
      |   ^~~~~~
UDP_client.c:25:2: warning: incompatible implicit declaration of built-in function ‘memset’
UDP_client.c:7:1: note: include ‘<string.h>’ or provide a declaration of ‘memset’
    6 | #include <strings.h>
  +++ |+#include <string.h>
    7 | #include <sys/socket.h>
UDP_client.c:32:39: warning: implicit declaration of function ‘strlen’ [-Wimplicit-function-declaration]
   32 |   sendto(sockfd, (const char*)message, strlen(message),
      |                                        ^~~~~~
UDP_client.c:32:39: warning: incompatible implicit declaration of built-in function ‘strlen’
UDP_client.c:32:39: note: include ‘<string.h>’ or provide a declaration of ‘strlen’
UDP_client.c:42:2: warning: implicit declaration of function ‘close’; did you mean ‘pclose’? [-Wimplicit-function-declaration]
   42 |   close(sockfd);
      |   ^~~~~
      |   pclose
[liveuser@localhost-live practical 7]$ ./server
Message From TCP client: Hello Server

Message from UDP client: Hello Server
```



```
[liveuser@localhost-live practical 7]$ ./tcp
Message from server: Hello Client
[liveuser@localhost-live practical 7]$ ./udp
Message from server: Hello Client
[liveuser@localhost-live practical 7]$
```