**Assignment 10 :**

**Problem Definition:**

**Problem Definition: Write a program using UDP Sockets to enable file transfer (Script,**

**Text, Audio and Video one file each) between two machines**

**1.1Prerequisite:**

a) Socket Header

b) Network Programming

c) Ports

*Learning Objectives:*

1

2

. To understand Work of Socket

. Different methods associated with Client & Server Socket

*New Concepts:*

1

2

. Client Server Communication

. Port Address

**1.3** *Theory:*

**1.3.1 Introduction**

What is UDP?

UDP is a connectionless and unreliable transport protocol.The two ports serve to

identify the end points within the source and destination machines. User Datagram

Protocol is used, in place of TCP, when a reliable delivery is not required.However,

UDP is never used to send important data such as web- pages, database information,

etc. Streaming media such as video, audio and others use UDP because it offers

speed.

*Why UDP is faster than TCP?*

The reason UDP is faster than TCP is because there is no form of flow control. No

error checking, error correction, or acknowledgment is done by UDP.UDP

is only concerned with speed. So when, the data sent over the Internet is affected by

collisions, and errors will be present. UDP packet's called as user datagrams with 8

bytes header. A format of userdatagrams is shown in figure

3. In the user datagrams first 8 bytes contains header information and the remaining

bytes contain data.

*LINUX SOCKET PROGRAMMING:*

The Berkeley socket interface, an API, allows communications between hosts or

between processes on one computer, using the concept of a socket. It can work with

many different I/O devices and drivers, although support for these depends on the

operating-system implementation. This interface implementation is implicit for

TCP/IP, and it is therefore one of the fundamental technologies underlying the

Internet. It was first developed at the University of California, Berkeley for use on

Unix systems. All modern operating systems now have some implementation of the

Berkeley socket interface, as it has become the standard interface for connecting to

the Internet. Programmers can make the socket interfaces accessible at three different

levels, most powerfully and fundamentally at the RAW socket level. Very few

applications need the degree of control over outgoing communications that this

provides, so RAW sockets support was intended to be available only on computers

used for developing Internet-related technologies. In recent years, most operating

systems have implemented support for it anyway, including Windows XP. The

header files: The Berkeley socket development library has many associated header

files. They include: Definitions for the most basic of socket structures with the BSD

socket API Basic data types associated with structures within the BSD socket API

Definitions for the stockading {} and other base data structures.

**The header files**:

The Berkeley socket development library has many associated

header files.They include: ***<sys/socket.h>***

Definitions for the most basic of socket structures with the BSD

**socket API *<sys/socket.h>***

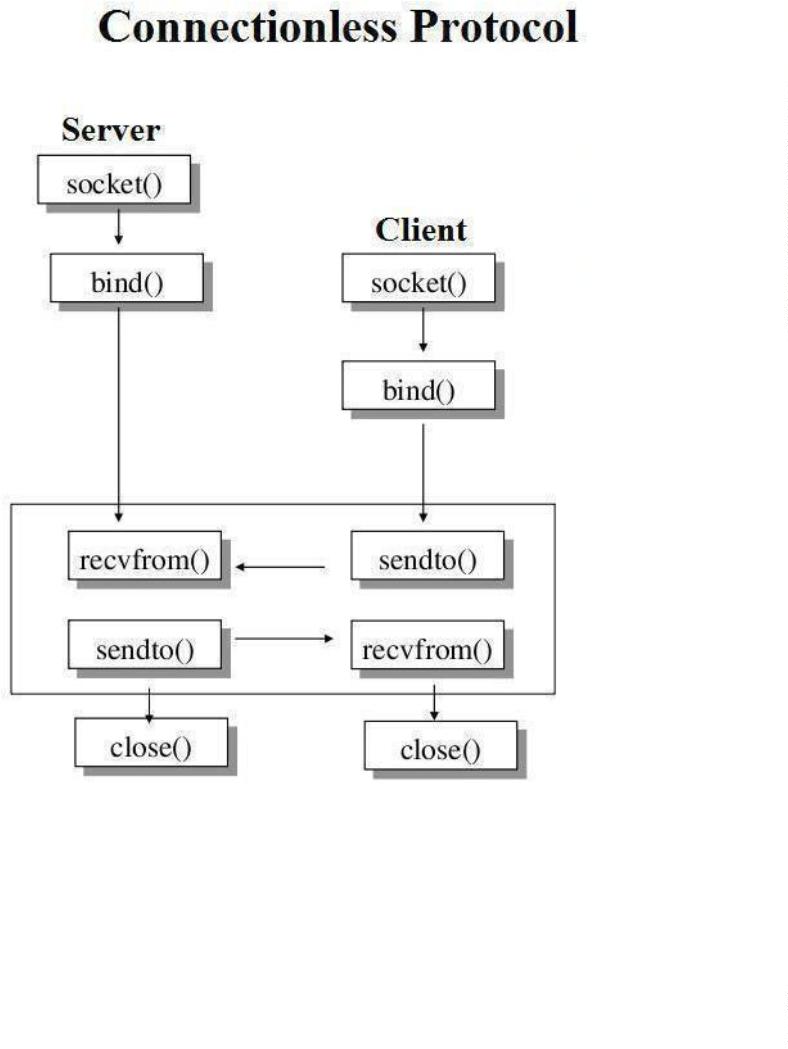
Basic data types associated with structures within the BSD socket API <sys/types.h>

***Socket API<sys/types.h>***

Definitions for the socketaddr\_in{} and other base data structures

***<sys/un.h>***

Definitions and data type declarations for SOCK\_UNIX streams



**UDP:** UDP consists of a connectionless protocol with no guarantee of delivery. UDP

packets may arrive out of order, become duplicated and arrive more than once, or

even not arrive at all. Due to the minimal guarantees involved, UDP has considerably

less overhead than TCP. Being connectionless means that there is no concept of a

stream or connection between twohosts, instead, data arrives in datagrams. UDP

address space, the space of UDP port numbers (in ISO terminology, the TSAPs), is

completely disjoint from that of TCP ports. Server: Code may set up a UDP server

on port 7654 as follows:

***sock =***

***socket(PF\_INET,SOCK\_DGR***

***AM,0);sa.sin\_addr.s\_addr***

***=***

***INADDR\_ANY;***

***sa.sin\_port = htons(7654);***

***bound = bind(sock,(struct sockaddr \*)&sa, sizeof(struct sockaddr));***

***if (bound < 0) fprintf(stderr, "bind(): %s\n",strerror(errno));***

***listen(sock,3);*** bind() binds the socket to an address/port pair. listen()

sets the length of the newconnections queue.

***while (1)***

***{***

***printf ("recv test ...\n");***

***recsize = recvfrom(sock, (void \*)hz, 100, 0, (struct sockaddr \*)&sa,***

***fromlen);printf ("recsize: %d\n ",recsize);***

***if (recsize < 0)***

***fprintf(stderr, "%s\n",***

***strerror(errno));sleep(1);***

***printf("datagram: %s\n",hz);***

***}***

This infinite loop receives any UDP datagrams to port 7654 using recvfrom(). It uses

the parameters: l socket l pointer to buffer for data l size of buffer l flags (same as in

read or other receive socket function)

Client: A simple demo to send an UDP packet containing "Hello World!" to address127.0.0.1,

port 7654 might look like this:

#include #include #include #include

#include#include int main(int argc, char

\*argv[])

***{***

***int sock; struct***

***sockaddr\_in sa;***

***int bytes\_sent,***

***buffer\_length;***

***char buffer[200];***

***sprintf(buffer,***

***"***

***Hello World!");***

***buffer\_length = strlen(buffer) +1;***

***sock = socket(PF\_INET,***

***SOCK\_DGRAM, 0);***

***sa.sin\_family = AF\_INET;***

***sa.sin\_addr.s\_addr =***

***htonl(0x7F000001);***

***sa.sin\_port = htons(7654); bytes\_sent = sendto(sock, buffer, buffer\_length,***

***0, &sa,sizeof(struct sockaddr\_in) );***

***if(bytes\_sent < 0) printf("Error sending packet: %s\n",***

***strerror(errno) );return 0;***

***}***

In this code, buffer provides a pointer to the data to send, and

buffer\_length specifies thesize of the buffer contents. Typical UDP client

code

●

Create UDP socket to contact server (with a given

hostname and service portnumber)

Create UDP packet.

●

●

●

Call send(packet), sending request to the server.

Possibly call receive(packet)

(ifwe need a reply).Typical UDP

Server code

●

●

Create UDP socket listening to a well known port number.

Create UDP packet buffer Call receive(packet) to get a request,

noting the address ofthe client.

●

Process request and send reply back with send(packet).

*APPLICATION :*

Socket programming is essential in developing any application over a network.

**Conclusion:** Thus we have studied Working of UDP Socket.

CONCLUSION Thus we have successfully implemented the socket programming for TCP

