Assignment

Aim : Write a program using UDP Sockets to enable file transfer (Script, Text, Audio and Video one file each) between two machines. Demonstrate the packets captured traces using Wireshark Packet Analyzer Tool for peer to peer mode.

Theory:
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:
<sys/socket.h>
Definitions for the most basic of socket structures with the BSD
socket API <sys/types.h>
Basic data types associated with structures within the BSD
socket API <netinet/in.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 two hosts, 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_DGRAM,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 new connections 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:

- socket
- pointer to buffer for data
- size of buffer
- flags (same as in read or other receive socket function)l address struct of sending peer
- length of address struct of sending peer.

Client:

A simple demo to send an UDP packet containing "Hello World!" to address 127.0.0.1, port 7654 might look like this:

```
#include <stdio.h>
#include <errno.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
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 the size of the buffer contents.

Typical UDP client code

1. Create UDP socket to contact server (with a given hostname and service port number)
2. Create UDP packet.
3. Call send(packet), sending request to the server.
4. Possibly call receive(packet) (if we need a reply).

Typical UDP Server code

1. Create UDP socket listening to a well known port number.
2. Create UDP packet buffer
3. Call receive(packet) to get a request, noting the address of the client.
4. 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.