7) Write a client-server application for chat using UDP

AIM: To write a client-server application for chat using UDP

ALGORITHM: CLIENT

1. Include necessary package in java

2. The client establishes a connection to the server.

3. The client accept the connection and to send the data from client to server and vice versa.

4. The client communicate the server to send the end of the message.

5. Stop the program.

ALGORITHM: SERVER

1. Include necessary package in java

2. The server establishes a connection to the client.

3. The server accept the connection and to send the data from server to client and vice versa.

4. The server communicate the client to send the end of the message

5. Stop the program.

// 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;

}

Outcome:

To implement a client server application using connection less service i.e. UDP.

8) Implement programs using raw sockets (like packet capturing and filtering)

ALGORITHM :

1. Start the program and to include the necessary header files

2. To define the packet length

3. To declare the IP header structure using TCPheader

4. Using simple checksum process to check the process

5. Using TCP \IP communication protocol to execute the program

6. And using TCP\IP communication to enter the Source IP and port number and Target IP address and port number.

7. The Raw socket () is created and accept the Socket ( ) and Send to ( ), ACK

8. Stop the program

//---cat rawtcp.c---

// Run as root or SUID 0, just datagram no data/payload

Program:

#include <unistd.h> #include <stdio.h> #include <sys/socket.h> #include <netinet/ip.h> #include <netinet/tcp.h>

// Packet length

#define PCKT\_LEN 8192

// May create separate header file (.h) for all

// headers' structures

// IP header's structure struct ipheader {

unsigned char iph\_ihl:5, /\* Little-endian \*/ iph\_ver:4;

unsigned char iph\_tos; unsigned short int iph\_len; unsigned short int iph\_ident; unsigned char iph\_flags; unsigned short int iph\_offset; unsigned char iph\_ttl; unsigned char iph\_protocol; unsigned short int iph\_chksum; unsigned int iph\_sourceip; unsigned int iph\_destip;

};

/\* Structure of a TCP header \*/ struct tcpheader {

unsigned short int tcph\_srcport; unsigned short int tcph\_destport; unsigned int tcph\_seqnum; unsigned int tcph\_acknum;

unsigned char tcph\_reserved:4, tcph\_offset:4;

// unsigned char tcph\_flags; unsigned int

tcp\_res1:4, /\*little-endian\*/

tcph\_hlen:4, /\*length of tcp header in 32-bit words\*/

tcph\_fin:1, /\*Finish flag "fin"\*/

tcph\_syn:1, /\*Synchronize sequence numbers to

start a connection\*/ tcph\_rst:1, /\*Reset flag \*/

tcph\_psh:1, /\*Push, sends data to the application\*/

tcph\_ack:1, /\*acknowledge\*/ tcph\_urg:1, /\*urgent pointer\*/ tcph\_res2:2;

unsigned short int tcph\_win; unsigned short int tcph\_chksum; unsigned short int tcph\_urgptr;

};

// Simple checksum function, may use others such as Cyclic Redundancy Check, CRC

unsigned short csum(unsigned short \*buf, int len)

{

unsigned long sum; for(sum=0; len>0; len--) sum += \*buf++;

sum = (sum >> 16) + (sum &0xffff); sum += (sum >> 16);

return (unsigned short)(~sum);

}

int main(int argc, char \*argv[])

{

int sd;

// No data, just datagram char buffer[PCKT\_LEN];

// The size of the headers

struct ipheader \*ip = (struct ipheader \*) buffer; struct tcpheader \*tcp = (struct tcpheader \*) (buffer + sizeof(struct ipheader));

struct sockaddr\_in sin, din; int one = 1;

const int \*val = &one; memset(buffer, 0, PCKT\_LEN);

if(argc != 5)

{

printf("- Invalid parameters!!!\n");

printf("- Usage: %s <source hostname/IP> <source port>

<target hostname/IP> <target port>\n", argv[0]); exit(-1);

}

sd = socket(PF\_INET, SOCK\_RAW, IPPROTO\_TCP);

if(sd < 0)

{

perror("socket() error"); exit(-1);

}

else

printf("socket()-SOCK\_RAW and tcp protocol is OK.\n");

// The source is redundant, may be used later if needed

// Address family sin.sin\_family = AF\_INET; din.sin\_family = AF\_INET;

// Source port, can be any, modify as needed sin.sin\_port = htons(atoi(argv[2])); din.sin\_port = htons(atoi(argv[4]));

// Source IP, can be any, modify as needed sin.sin\_addr.s\_addr = inet\_addr(argv[1]); din.sin\_addr.s\_addr = inet\_addr(argv[3]);

// IP structure ip->iph\_ihl = 5;

ip->iph\_ver = 4;

ip->iph\_tos = 16;

ip->iph\_len = sizeof(struct ipheader) + sizeof(struct tcpheader);

ip->iph\_ident = htons(54321); ip->iph\_offset = 0;

ip->iph\_ttl = 64;

ip->iph\_protocol = 6; // TCP

ip->iph\_chksum = 0; // Done by kernel

// Source IP, modify as needed, spoofed, we accept through command line argument

ip->iph\_sourceip = inet\_addr(argv[1]);

// Destination IP, modify as needed, but here we accept through command line argument

ip->iph\_destip = inet\_addr(argv[3]);

// The TCP structure. The source port, spoofed, we accept through the command line

tcp->tcph\_srcport = htons(atoi(argv[2]));

// The destination port, we accept through command line tcp->tcph\_destport = htons(atoi(argv[4]));

tcp->tcph\_seqnum = htonl(1); tcp->tcph\_acknum = 0;

tcp->tcph\_offset = 5;

tcp->tcph\_syn = 1;

tcp->tcph\_ack = 0;

tcp->tcph\_win = htons(32767);

tcp->tcph\_chksum = 0; // Done by kernel tcp->tcph\_urgptr = 0;

// IP checksum calculation

ip->iph\_chksum = csum((unsigned short \*) buffer, (sizeof(struct ipheader) + sizeof(struct tcpheader)));

// Inform the kernel do not fill up the headers' structure, we fabricated our own

if(setsockopt(sd, IPPROTO\_IP, IP\_HDRINCL, val, sizeof(one))

< 0)

{

perror("setsockopt() error"); exit(-1);

}

else

printf("setsockopt() is OK\n");

printf("Using:::::Source IP: %s port: %u, Target IP: %s port: %u.\n", argv[1], atoi(argv[2]), argv[3], atoi(argv[4]));

// sendto() loop, send every 2 second for 50 counts unsigned int count;

for(count = 0; count < 20; count++)

{

if(sendto(sd, buffer, ip->iph\_len, 0, (struct sockaddr

\*)&sin, sizeof(sin)) < 0)

// Verify

{

perror("sendto() error"); exit(-1);

}

else

printf("Count #%u - sendto() is OK\n", count); sleep(2);

}

close(sd); return 0;

}

RESULT :

Thus the Above programs using raw sockets TCP \IP (like packet capturing and filtering) was executed and successfully.