Computer Networks Laboratory

IT 3095

Lab instructions
On

Connection-less User Datagram Protocol (UDP) socket programming for client/server communication and analysis of traffic using Wireshark protocol analyzer

Aim of the Experiment: **Development** of client-side and server-side socket programs for the connection-less User Datagram Protocol (UDP) process-to-process communication based on 'c' language using UNIX Socket Programming APIs. **Execution** of the client/server programs and **analyze** the message/Protocol Data unit (PDU) exchange between client-server using Wireshark protocol analyzer.

Objective: Students will get hands-on experience on UNIX Socket Application Programming Interfaces (APIs) and analyze the traffic/protocols using Wireshark Protocol Analyzer for UDP client/server communications.

Software Required: UNIX/Linux OS with GNU Compiler Collection (gcc) complier, Wireshark Protocol Analyzer

N.B. *All the required software is preinstalled and configured in the Virtual Machine (VM) to run over Oracle VM VirtualBox*

Virtual Machine Name: "CN-Lab-IT-3095"

Download:
https://drive.google.com/file/d/1x8C6w8ukTTaQfhjgKh2EISvW8nXRWwCy/view?usp=sharing

Codes:

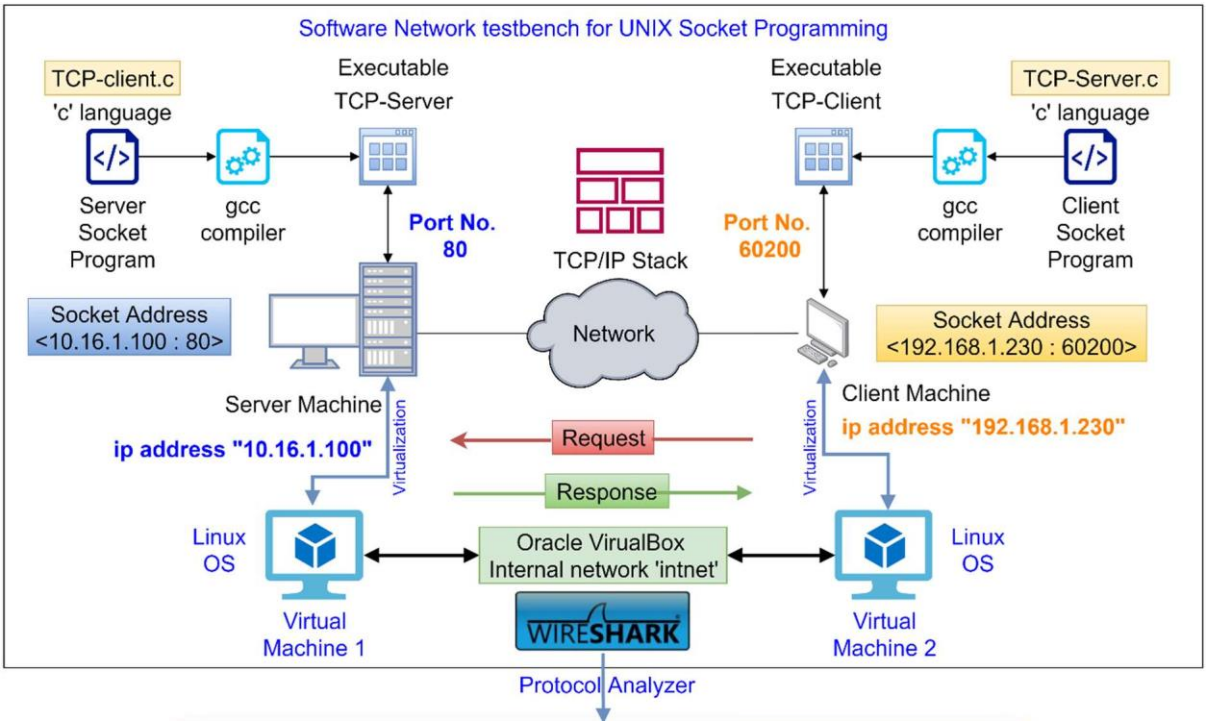
udp_server.c
udp_client.c

Network Scenario: In a network, two computers/hosts need to communicate over a connection- less framework over UDP. To enable this communication two socket programs has to be written in'c'-language. The server program is '*udp_server.c*' and the client program is '*udp_client.c*'. After writing the code it has to be complied using UNIX gcc compiler. We have to considered two machinesrunning over Linux OS for implementation. The message communication between client and server output to be visualized in Linux 'Terminal' window. Additionally, the protocol data units/packets exchanged between client and server need to be analyzed using Wireshark protocol analyzer.

Web-Socket/ internet-socket {} Programming Concept:

- Socket programming is used to develop network enabled computer programs to support TCP/IP protocol suite for communication over internet framework.
- Socket programs for client and server has to be written using the suitable libraries and APIs.
- Suitable header files and their specific APIs are used to implement socket programs.
- Socket programs are developed and implemented over two systems/machines connected over a network.
- In most of the real-world implementations Socket programs works in a pair of client/server to request/response-based message transactions over a computer network/internet.

Fundamentals of Client/Server Communication using Socket Programming



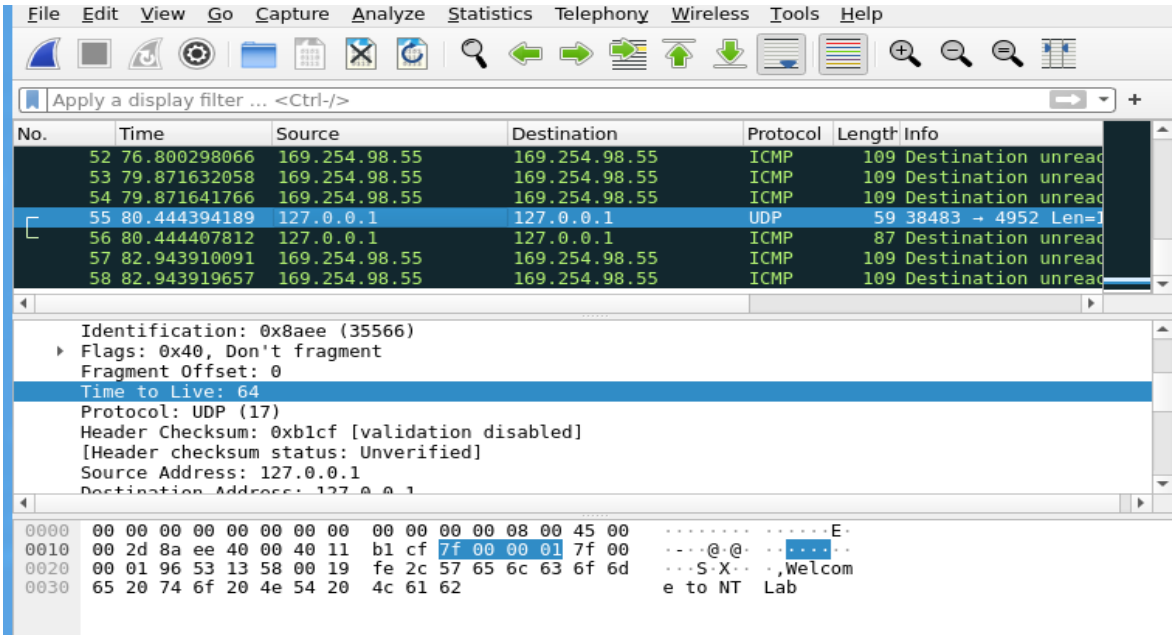


Fig. 1 UDP Client/server communication analysis using Wireshark.

udp_server.c

```
/*
** A datagram sockets "server" demo
*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#define MYPORT 4952 // the port users will be connecting to
#define MAXBUFLEN 200
int main()
{
int sockfd;
struct sockaddr_in my_addr; // my address information
struct sockaddr_in their_addr; // connector's address information
socklen_t addr_len;
int numbytes;
char buf[MAXBUFLEN];

if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
perror("socket");
exit(1);
}
my_addr.sin_family = AF_INET; // host byte order
my_addr.sin_port = htons(MYPORT); // short, network byte order
my_addr.sin_addr.s_addr = INADDR_ANY; // automatically fill with my IP
//memset(my_addr.sin_zero, '\0', sizeof my_addr.sin_zero);

if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof my_addr) == -1) {
perror("bind");
exit(1);
}

addr_len = sizeof their_addr;
if ((numbytes = recvfrom(sockfd, buf, MAXBUFLEN-1 , 0,
(struct sockaddr *)&their_addr, &addr_len)) == -1) {
perror("recvfrom");
exit(1);
}
printf("got packet from %s\n",inet_ntoa(their_addr.sin_addr));
printf("packet is %d bytes long\n",numbytes);
buf[numbytes] = '\0';
printf("packet contains \"%s\"\n",buf);
close(sockfd);
return 0;
}
```

udp_client.c

```
/*
** A datagram "client" demo
*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#define SERVERPORT 4952 // the port users will be connecting to
int main()
{
int sockfd;
struct sockaddr_in their_addr; // connector's address information
//struct hostent *he;
int numbytes;
char arg[30];

if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
perror("socket");
exit(1);
}
their_addr.sin_family = AF_INET; // host byte order
their_addr.sin_port = htons(SERVERPORT); // short, network byte order
their_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
//memset(their_addr.sin_zero, '\0', sizeof their_addr.sin_zero);

printf("Enter a message\n");
gets(arg);

if ((numbytes = sendto(sockfd, arg, strlen(arg), 0,
(struct sockaddr *)&their_addr, sizeof their_addr)) == -1) {
perror("sendto");
exit(1);
}
printf("sent %d bytes to %s\n", numbytes, inet_ntoa(their_addr.sin_addr));
close(sockfd);
return 0;
}
```

Procedure:

- Write the programs using any text editor
- Open Linux 'Terminal'
- Navigate to PATH: *cd /root/CN-LAB/EXPT3*
- Compile the programs in Linux 'Terminal'
    - Command: *gcc <filename.c> -o <filename of Executable>*
        - *gcc udp_server.c -o UDP-server*
        - *gcc udp_client.c -o UDP-client*

- Execute the program in Linux 'Terminal':
    - Command: *./ <filename of Executable>*
    - E.g. *./ UDP-server* and *./UDP-client (In separate Terminal windows)*

- Run the client and server with in the same host.
- Observations : Record Observations

Observations: (Take Screen-shorts for putting Outputs in the Record)

A. Terminal

- Observe the messages transaction between client-server and vice-versa.

B. Wireshark Protocol Analyzer

- Finally capture and analyze the packets exchanged between them. *(using Wireshark)*
- Open "Wireshark'
- Capture "*Loopback:lo"(Same Host) or"enpos3*" *(Different hosts)* Network Interface
- Filter "tcp"
- Analyze > Follow > UDP Stream (UDP Payload/ Data)
- Statistics > I/O Graphs (Study 'udp' filtered packets)
- Statistics > Flow Graph (Study Flow Type: UDP Flow)

Comments based on observations: Based on above activity, write your observations.

Conclusion: After the experiment, students will write conclusion in their own words.

Modified Task: Run the client in one host and the server in another host and repeat the experiment.

***