Computer Networks Laboratory

IT 3095

Lab instructions
On

Connection-oriented Transmission Control Protocol (TCP) 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-oriented Transmission Control Protocol (TCP) 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 TCP 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:

TCP-server.c
TCP-client.c

Network Scenario: In a network, two computers/hosts need to communicate over a connection-oriented framework over TCP. To enable this communication two socket programs has to be written in 'c'-language. The server program is '*TCP-server.c*' and the client program is '*TCP-client.c*'. After writing the code it has to be complied using UNIX gcc compiler. We have to considered two machines running 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

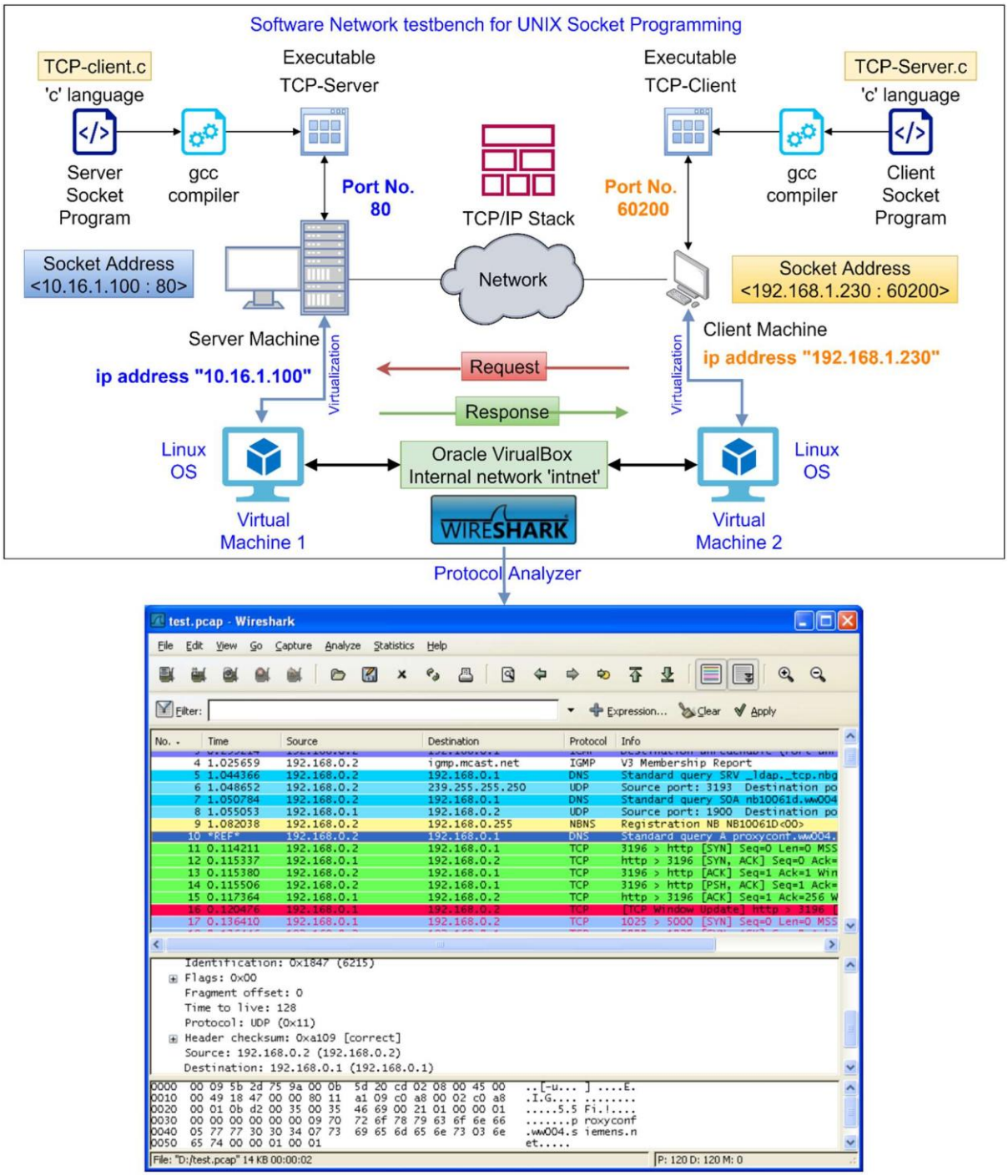

Fig. 1 TCP Client/server communication analysis using Wireshark.

TCP-server.c

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<fcntl.h>
#include<string.h>

main()
{
int sockfd,fd1, length,i,x,y,sum;
char buf[100],p[3]; /* We will use this buffer for communication */struct
sockaddr_in sa,ta1;

/* The following system call opens a socket. The first parameter indicates the family of the protocol to be followed. For internet protocols we use AF_INET. For TCP sockets thesecond parameter is SOCK_STREAM. The third parameter is set to 0 for user applications.*/
sockfd=socket(AF_INET,SOCK_STREAM,0);

/* The structure "sockaddr_in" is defined in <netinet/in.h> for the internet family of protocols. This has three main fields. The field "sin_family" specifies the family and is therefore AF_INET for the internet family. The field "sin_addr" specifies the internet address of the server. This field is set to INADDR_ANY for machines having a single IP address. The field "sin_port" specifies the port number of the server.*/

 sa.sin_family=AF_INET;
 sa.sin_addr.s_addr=INADDR_ANY;
 sa.sin_port=60018;

/* With the information provided in serv_addr, we associate the server with its port usingthe bind() system call. */

 i=bind(sockfd,(struct sockaddr *)&sa,sizeof(sa));
 printf("test %d%d\n",sockfd,i);
```

```
/* This specifies that up to 5 concurrent client requests will be queued up while the systemis executing the "accept" system call below.*/

listen(sockfd,5);

/* The accept() system call accepts a client connection. It blocks the server until a client request comes.
The accept() system call fills up the client's details in a struct sockaddr which is passed asa parameter. The length of the structure is noted in clilen. Note that the new socket descriptor returned by the accept() system call is stored in "fd1".*/

 length=sizeof(sa);
 fd1=accept(sockfd, (struct sockaddr *) &ta1,&length);

/* We initialize the buffer, copy the message to it, and send the message to the client. */for(i=0; i

 < 100; i++) buf[i] = '\0';
                              strcpy(buf,"Message from server");
                              send(fd1, buf, 100, 0);

/* We again initialize the buffer, and receive a message from the client. */

for(i=0; i < 100; i++) buf[i] = '\0';
                              recv(fd1, buf, 100, 0);
                              printf("%s\n", buf);

close(fd1);

 }
```

TCP-client.c

```
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<arpa/inet.h>
#include<fcntl.h>
#include<string.h>

main()
{
  int i,sockfd,a,b,p=6,q=7,r;
  char buf[100];
  struct sockaddr_in sa;

/* Opening a socket is exactly similar to the server process */
  sockfd=socket(AF_INET,SOCK_STREAM,0);

/* Recall that we specified INADDR_ANY when we specified the server address in the server. Since the client can run on a different machine, we must specify the IP address ofthe server.

TO RUN THIS CLIENT, YOU MUST CHANGE THE IP ADDRESS SPECIFIED BELOW TO THEIP ADDRESS OF THE MACHINE WHERE YOU ARE RUNNING THE SERVER.*/

  sa.sin_family=AF_INET;
  sa.sin_addr.s_addr=inet_addr("127.0.0.1"); //Loop back IP address
  sa.sin_port=60018;

/* With the information specified in serv_addr, the connect() system call establishes a connection with the server process.*/

  i=connect(sockfd,(struct sockaddr *)&sa,sizeof(sa));
```

```
/* After connection, the client can send or receive messages. However, please note that recv() will block when the server is not sending and vice versa. Similarly send() will blockwhen the server is not receiving and vice versa. For non-blocking modes, refer to the online man pages.*/

for(i=0; i < 100; i++) buf[i] = '\0';
        recv(sockfd, buf, 100, 0);
        printf("%s\n", buf);

        for(i=0; i < 100; i++) buf[i] = '\0';
        strcpy(buf,"Message from client");
        send(sockfd, buf, 100, 0);

close(sockfd);

 }
```

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 TCP-server.c -o TCP-server*
        - *gcc TCP-client.c -o TCP-client*

- Execute the program in Linux 'Terminal':
    - Command: *./ <filename of Executable>*
    - E.g. *./ TCP-server* and *./TCP-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 > TCP Stream (TCP Payload/ Data)
- Analyze > Expert Information (Check TCP SYN, SYN+ACK, FIN etc.)
- Statistics > I/O Graphs (Study 'tcp' filtered packets)
- Statistics > Flow Graph (Study Flow Type: TCP 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.

***