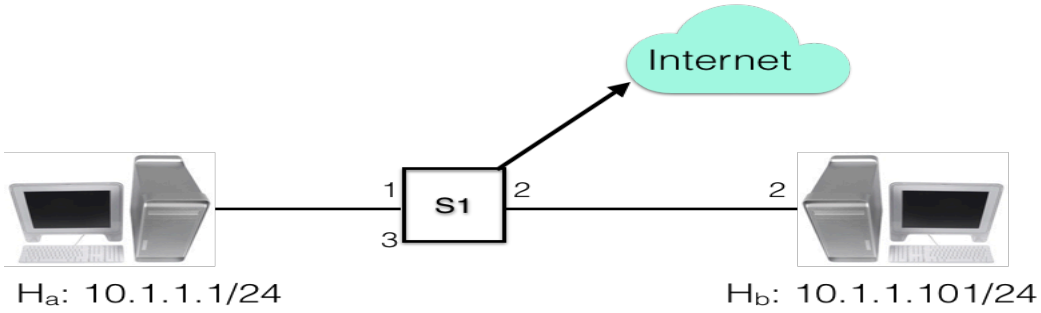
Exercises – Basics of Network Programming

Experiential Learning Workshop

1 General Guidelines

1. Make a team of two unless stated otherwise.
2. The TCP server would run on machine A and client would run on machine B or vice versa.



3. For each exercise, use **wireshark** capture to analyze the packets exchanged.
4. The default client for connecting to TCP server is assumed to be `tcpclient01`, an executable of program `tcpclient01.c`. For simple client program or simple server for a basic communication, use **nc**. The IP address used in the example below are for illustrative purposes.
5. The exercises below assume that client's IP address is 10.1.1.1 and other machine in your team has the IP address 10.1.1.101. Please use appropriate IP address in your setup. Replace these with the values applicable to your setup.
6. To kill any program in the linux terminal, please press **`Ctrl-C`** and not **`Ctrl-Z`**. The latter will suspend the program and not stop it.
7. Use **gcc** to compile a program and use option `-o` to name the executable file, e.g. `gcc -o tcpclient -g tcpclient01.c`
8. Both the client and server programs require command line parameters and same should be specified when running.
9. The max buffer size supported is 8192 bytes.

2 Description of programs

When the program is invoked without any command line arguments or with -h option, it will display its usage syntax with mandatory and options arguments.

2.1 Client programs

1. The client program (`tcpclient01`) supports following command line arguments.
   a. -s: IP address of server. Mandatory
   b. -p: port number of server. Mandatory
   c. -c: count of messages sent by client. Mandatory.
   d. -d: Initial delay in milli seconds before the first message is sent. Optional with default value of 0ms.
   e. -i: Average time gap in milli seconds between two messages. Optional with default value of 2s (or 2000 ms).

2. Client program waits for a response (even empty) before sending the next message. The average gap value (option -i) is applicable after it has received response from the server.
3. To connect to a server 10.1.1.101 and send 10 messages with initial delay of 3 seconds, and average gap of 4 seconds, invoke the following command in the terminal window.
    a. `./tcpclient01 -s 10.1.1.101 -p 2222 -c 10 -d 3000 -i 4000`
    b. The option `-d` and `-i` are optional and can be skipped to use their default values.

2.2 Description of server programs.

There are several server programs (`tcpserver01`, …, `tcpserver12`) providing an insight into evolution of server programming. Basic description and features of these server programs are described below. The common command line arguments for server programs are given below.

-s : server IP Address. Mandatory
-p : server port number. Mandatory.

Note: A server can have multiple IP Addresses and thus to specify which IP address to be used, the option -s is used. To specify all the IP addresses on all the interface use the IP address 0.0.0.0.

2.2.1 tcpserver01.c

Receives one message, waits for a second and echoes back the received message to the client and then exits. Thus, just a simple server to process one message.

2.2.2 tcpserver02.c

Before it accepts a message, it waits for about 300s (5 minutes) after `listen` call. This to indicate TCP connection management that client connection can be still be established after server has issues `listen`() though not issued accept() call. Further, it issues 2nd `recv`() call as after first send.

2.2.3 tcpserver03.c

Compared to tcpserver01.c, it has extra `recv`() calls before it exits. This is to indicate what happens when a client closes connections and server still reads the data.

2.2.4 tcpserver04.c

When server program does not use `htons`(port) to ensure Network byte order, then this program indicated what is the port number value to which server program is bound to.

2.2.5 tcpserver05.c

Server binds to all the IP Addresses belong to the machine i.e. it binds to IP address 0.0.0.0. Further, server keeps processing all the messages from a client. When the client closes the connection, then waits for the next client. However, server program accepts connections from one client at a time and communicated with that client till the client closes the connection. So, far (tcpserver01.c to

tcpserver04.c), these were just accepting only one client connection and then exiting.

2.2.6 tcpserver06.c

Whenever a client program connects to a server, it forks a new child process and that child process deals with communication from the client. The parent process waits for the new client to connect.

2.2.7 tcpserver07.c

This server program fixes the bug of tcpserver06.c where it clears all the zombie processes that were created when child process exited. The clearance of zombie process occurs before the parent process waits for a new client connection.

2.2.8 tcpserver08.c

This program is further improvement on tcpserver07, where it defines a signal handler for child exit. Thus, whenever a child process exits, the parent process gets the signal and it handles the signal and clear the zombie process right away rather than waiting for next new client to cnnect. Thus is it efficient on system resources.

2.2.9 tcpserver09.c

This server program does not create any child for a client. It is a single threaded program that deals with multiple client concurrently. This is achieved by making use of `select`() socket call.

2.2.10 tcpserver10.c

This server program is an efficient improvement over tcpserver09. This program makes use of `poll`() socket call instead of `select`. Thus, there is no need to maintain separate master and working set of fds for each socket connection.

2.2.11 tcpserver12.c

This server program is fully an efficient implementation of a server program that handles any number of concurrent clients. This program makes use of `epoll`() socket call instead of `select/poll`. Today efficient TCP server makes use of this socket call to provide efficiency.

3 Hands-on 1: Simple client server communications

3.1 Single Server communication

3.1.1 Single Server Single Client communication

1. Run the single server (`tcpserver01`) on your chosen port e.g. `2222`.
2. Use `nc` as a client (on a different machine) to connect to this server program.
3. After the communication, close the connection
4. Analyze the TCP communication using wireshark.
5. Analyze the TCP server states before client connection, during the conversation and after the connection is closed by client. Use the command `netstat -natp` to look at the TCP Server states.

6. Connect multiple `nc` clients to this single server. Analyze the connection state. All connection should be in ESTABLISHED state

3.1.2 Single Server with delay before starting

1. Run the single server (`tcpserver02`) on your chosen port e.g. 2222.
2. Connect a client to it, Is the connection successful? Analyze the TCP state.

3.1.3 Server reading on a closed connection

1. Run the single server (`tcpserver03`) on your chosen port e.g. 2222.
2. Connect a client to it, Is the connection successful? Analyze the TCP state.
3. When the connection is closed by the client (Ctrl-C), what is the value returned by `read`() call. What is the value returned by next `read()` call.

3.1.4 UDP server communication

1. Run the the udp server using (`nc -u`) on your chosen port e.g. 2222 e.g. use the command `nc -u -l 2222`.
2. Use multiple concurrent udp clients to communicate with this server. Is the coimmunication successful
3. Analyse the difference between TCP server and UDP server

3.2 Exercise Expectations

1. Understanding of basic TCP communication and TCP Connection state.
2. Understanding EOF (End Of File) on a proper connection close.
3. Understand the difference of TCP Sockets and UDP Sockets.

4 Hands-on 2: Handling basic errors in TCP server

4.1 TCP Server ignore Network Byte order

1. Run the TCP Server (`tcpserver04`) which does not use `htons`(port) function to convert port number (e.g. 2222) to network byte order.
2. Which port number is used by TCP server.
3. What happens when a client connects to server on original port 2222.
4. What happens when a client connects to the server equivalent to network byte order port number.

4.2 Concurrent invocation of TCP Server on same port

1. Run two TCP Server (`tcpserver01`/02/03/04) on same port numbers concurrently (e.g. 2222)
2. Which server program is able to run. What kind of errors other server program encounters on invocation.

4.3 Connecting to UDP Server on non-existent port

1. Connect a client to UDP server which is not running on the specified port.
2. What kind of error is returned by UDP server machine.
3. Analyze the error in wireshark.

4.4 Exercise Expectations

1. Understand occurrence of basic errors in communication with TCP/UDP servers.

5 Hands-on 3: TCP server with concurrent clients.

5.1 A Single TCP server handling concurrent clients

1. Run the TCP server (`tcpserver06`) that spawns a child for each new client.
2. Connects at least 5 clients and exchange communication on each of these few times.
3. Analyze the communication in wireshark.
4. Identify the difference in socket tuples for these connections.
5. Terminate few clients and launch new clients
6. Identify the zombie processes

5.2 A Single TCP server handling concurrent clients clearing zombie process

1. Run the TCP server (`tcpserver07`) that spawns a child for each new client. This server also clears up zombie processes before it starts a new connection.
2. Connect few clients (at least 5) and exchange data on each.
3. Terminate some clients (but do not launch any client)
4. Can you identify the zombie processes?
5. Start a new client connecting to this server,
6. Do you still see the zombie processes?
7. Analyze the working of zombie processes.

5.3 Single TCP server handling concurrent clients without zombie processes

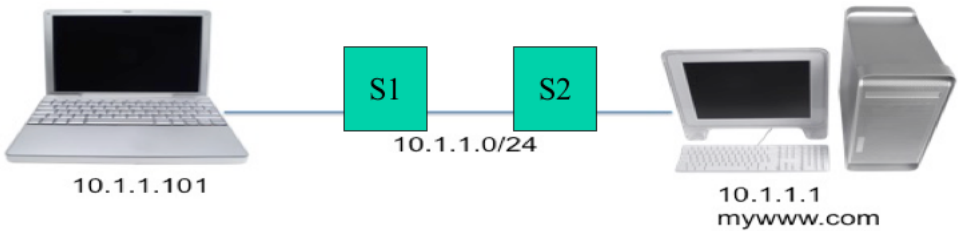
1. Run the TCP server (`tcpserver08`) that spawns a child for each new client. This server also clears up zombie processes before it starts a new connection.
2. Connect few clients (at least 5) and exchange data on each.
3. Terminate some clients (but do not launch any client)
4. Do you see any zombie processes?
5. Understand the working signals and SIGCHLD in particular.

5.4 Exercise Expectations

i. Understanding of dealing with concurrent clients by a single process.
ii. Understand the working of zombie processes and their clean up.

6 Hands-on 4: TCP Streaming and UDP message boundary

The setup for this exercise requires two switches to be used as follows:



Note:

i. The switch S1 can be your lab network switch, and thus connect 2nd machine via a switch which in turn is connected to wall jack (lab network switch).

ii. We will use following simple python programs. You don't really need to know the python language syntax (reading of the program itself should explain what is being done).
   a. `udp_client.py`
   b. `udp_server.py`
   c. `tcp_client.py`
   d. `tcp_server.py`

iii. Use the option -h to understand the usage syntax. By default, server listens on all the IP address and port number 9999 and receive in the buffer size of 10 bytes and reads data from socket every 1 second and displays the same. The client program by default connects to server on port 9999, but requires server IP to be specified, the client uses a default buffer size of 100 bytes and sends 10 messages at interval of 5 seconds each. The options for these programs are
   a. –s <server IP address>
   b. –p <server port number>
   c. –b <buffer size>
   d. –c <count of packets/messages to be sent>. The first message contains sequence of A… characters, 2nd message contains sequence of B… characters, and so on.
   e. –d delay (in sending by client or receiving by server)

6.1 UDP message Orientation

1. On H2, run the server `./udp_server.py` with buffer size of 20.
2. On H1, run the client `./udp_client.py` sending the message to server with buffer size of 50 bytes.
3. Note down what is displayed by the server. Explain why the server does not receive full packet. Change the buffer size of server to study further.
4. Analyze wireshark capture on what is being transmitted by client.

6.2 UDP Packet Loss

1. Repeat the above exercise with client sending 10 packets at interval of 5 seconds.
2. Break the link between switch S1 and S2 at 12th seconds and restore this link at 26th second. To break the link, just remove the wire between 2 switches or between switch and the wall jack.
3. Using wireshark, analyze the packet sent by client as well as those received by server. Explain which packets are received and which are lost.

6.3 TCP Stream based transmission

1. Repeat the above exercise but with tcp client and servers.
2. Analyze if the server receives full data.
3. Explain and understand the difference between TCP and UDP behavior.

6.4 TCP Packet Loss and Recovery

1. Repeat the above exercise.
2. Analyze if the server receives full data i.e. data transmitted during the time when link between S1 and S2 is broken/disconnected.
3. Analyze how many TCP segments were transmitted vs how many received.
4. Analyze if retransmitted segment contained any new data?

5. Analyze the time difference between retransmitted segments.
6. Explain and understand the difference between TCP and UDP behavior.

6.5 Exercise Expectations

1. Ability to understand UDP message delivery being boundary oriented.
2. Ability to understand TCP based message streaming i.e. no message boundaries.
3. Ability to under packet loss recovery in TCP whereas there is no recovery in UDP packet loss.

← end of exercises handout →