Experiential Learning Workshop on Basics of Socket Programming

June 28, 2018

Dr. Ram P Rustagi
Professor, CSE Dept
KSIT, Bangalore
rprustagi@ksit.edu.in

Resources

- https://rprustagi.com/ELNT/Experiential-Learning.html
  – Articles in ACCS Journal https://acc.digital
- www.github.com/rprustagi
  – Source code and examples for articles
- https://www.rprustagi.com/ieee/drait
  – Slides for this talks, Example programs, web pages
- Program source
  – rprustagi.com/workshops/programs

Day 1: Basics of Networking

- Overview
- Introduction to basic networking Tools
- Hands-on 1: using networking tools
- TCP/IP Stack 4 layer model
- Analysis of layers in IP, TCP/UDP and HTTP
- Handson-2: layers in ping, nc, http
- IP addressing, subnetting and routing
- Hierchical addressing
- Handson-3: Subnet mismatch and reachability
- Supernetting, longest prefix match
- Handson-4: Overlapping subnets, longest match
- Summary

Day 2: Basics of Socket Programming

- Overview: sockets
- Simple client server programs
- Handson-1: Writing TCP and UDP server
- Errors in socket programming
- Network byte order, buffer mgmt, socket close
- Handson-2: Network byte order, socket close
- Multiple concurrent client communication
- Socket call return value and reliability
- Handson-3: Handling concurrent clients, listen
- TCP Streaming and UDP Message boundary
- Handson-4: TCP streaming & UDP msg boundary
- Summary

Day 3: Basics of Web Security

- Overview: HTTPS protocol
- Server certificate and server authentication
- Handson-1: Deploying a SSL certificate
- Mixed content and browser warnings
- Locks icons and HTTP Status
- Handson-2: Creating website with mixed content
- MITM attack and ARP spoofing
- MITM with browser and information stealing
- Handson-3: Implementing MITM with arpspoofing
- Understanding HSTS, CSP
- Handson-4: Implementing ARP Spoofing
- Summary

Day 2: Basics of Socket Programming

- Overview: sockets
- Simple client server programs
- Handson-1: Writing TCP and UDP server
- Errors in socket programming
- Network byte order, buffer mgmt, socket close
- Handson-2: Network byte order, socket close
- Multiple concurrent client communication
- Socket call return value and reliability
- Handson-3: Handling concurrent clients, listen
- TCP Streaming and UDP Message boundary
- Handson-4: TCP streaming & UDP msg boundary
- Summary

Inter Process Communication

- IPC - within same system
  - Files
  - Shared memory
  - Pipes
  - Messages
  - Unix sockets
- IPC - systems across network
  - Network sockets
  - Messages (Queues)

TCP/UDP Communication

- Client-Server Network Programming
- Server Address (IP + Port)
  – What to do when server has multiple IP addresses
- Client Address (IP + Port)
  – What to do when client has multiple IP addresses?
- Communication protocol supports two types
  – In order, loss less, no duplicate delivery
  – out of order, loss of data, duplicate data
- Understand each of these

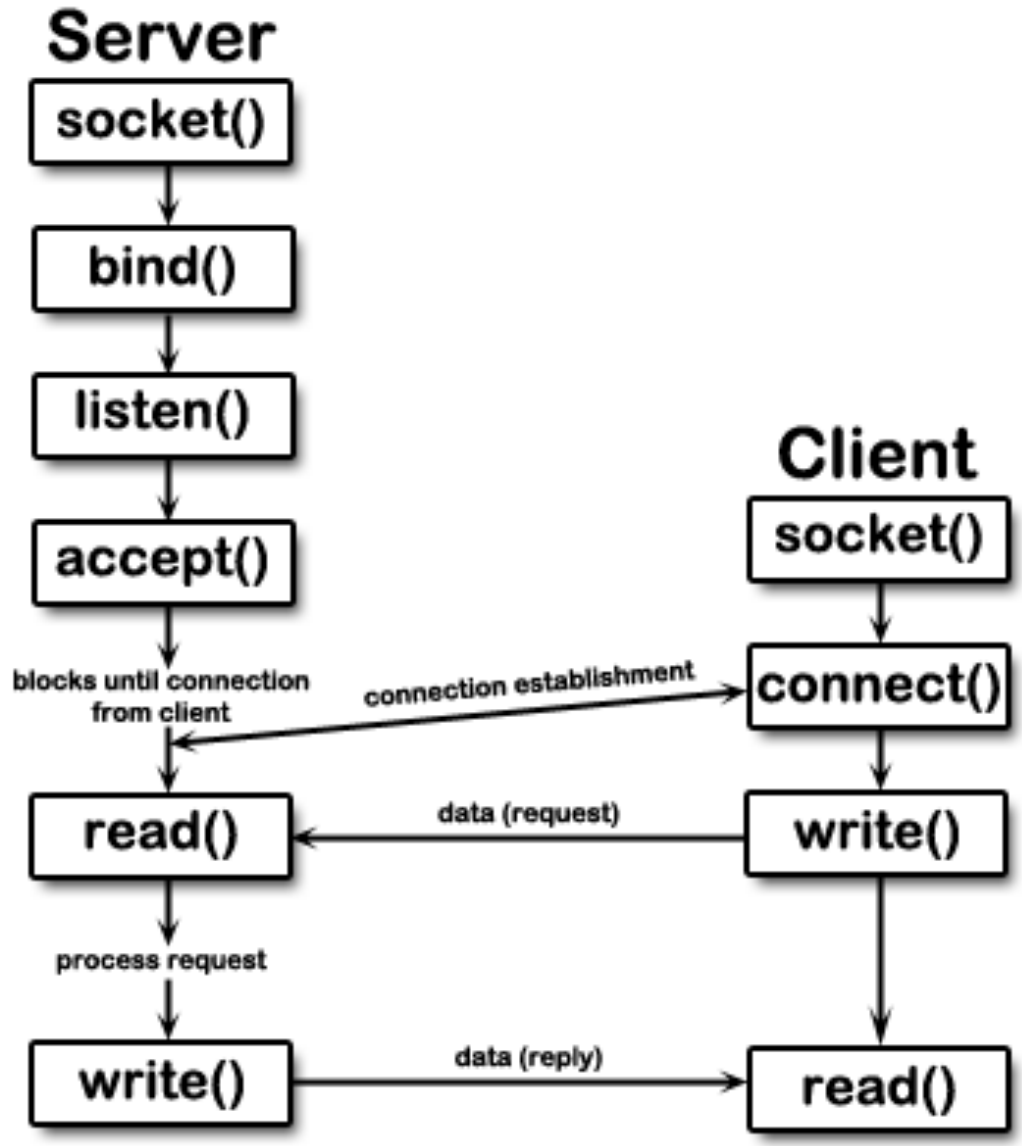
TCP Server Side

- Need to have a well known address
- Communication structure to follow file based (Linux)
- Should always be available
- How does machine know its availability
- Concurrent clients and clients in queue (waiting)
- Handling new clients
- Support from OS to know active clients (concurrent)
- Termination of communication

TCP Client Side

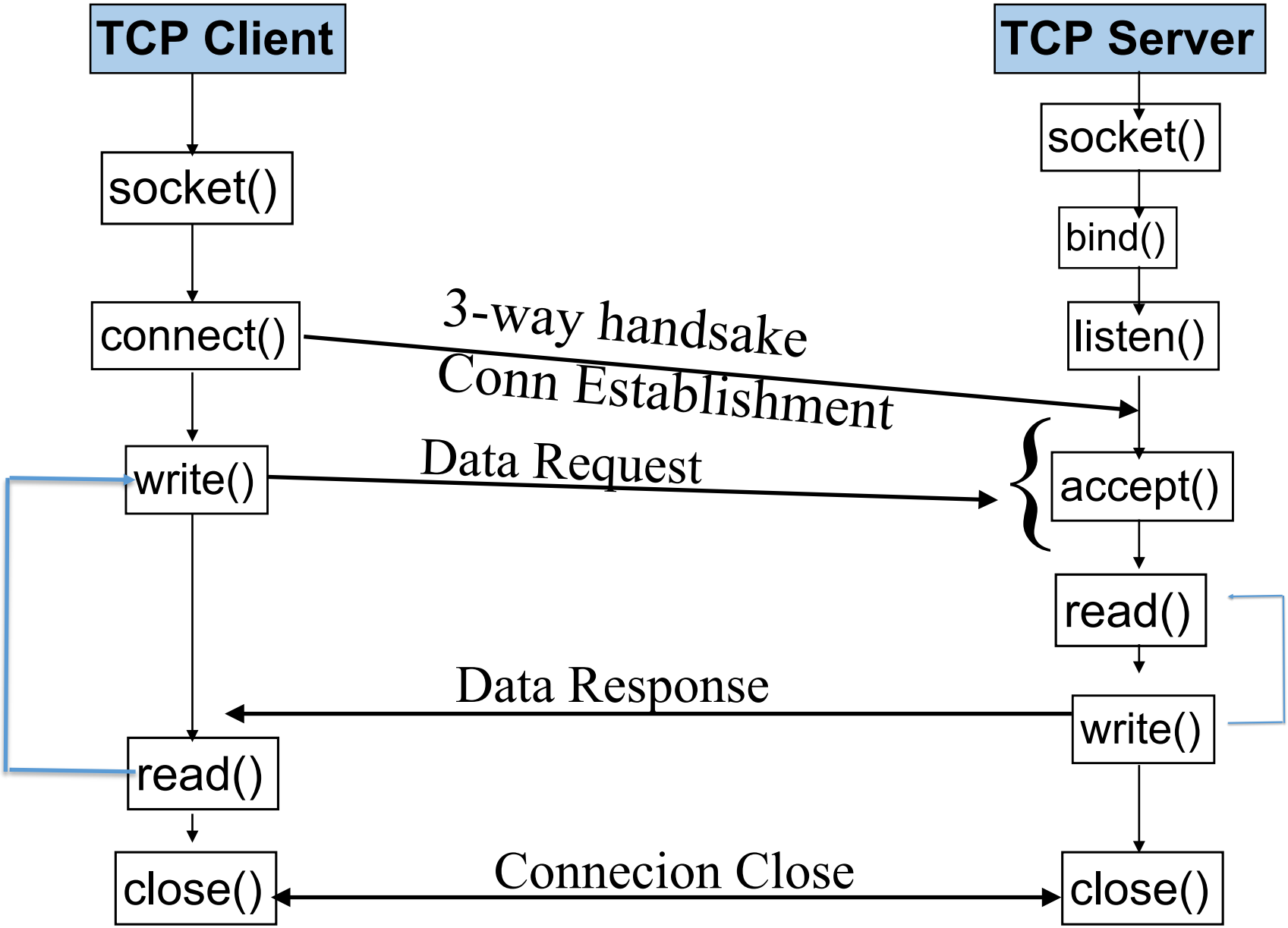
- Communication structure to follow file based (Linux)
- Does it need (a well known) address?
- How to determine its address
- Need to know server availability
- How does it get confirmation on data delivery
- Communication with multiple servers
- Termination of communication

Typical Diagram found in internet/literature.



ftp://www.cs.uregina.ca/pub/class/330/Sockets/sockets.html

TCP Client/Server Communication



TCP Connection State

- Use netstat -nat to know the current state
  - LISTEN
  - SYN-RECD
  - SYN-SENT
  - ESTABLISHED
  - CLOSED -> LAST_ACK
  - FIN_WAIT1
  - FIN_WAIT2
  - TIME_WAIT

Day 2: Basics of Socket Programming

- Overview: sockets
- Simple client server programs
- Handson-1: Writing TCP and UDP server
- Errors in socket programming
- Network byte order, buffer mgmt, socket close
- Handson-2: Network byte order, socket close
- Multiple concurrent client communication
- Socket call return value and reliability
- Handson-3: Handling concurrent clients, listen
- TCP Streaming and UDP Message boundary
- Handson-4: TCP streaming & UDP msg boundary
- Summary

Simple TCP Server (tcpserver01.c)

```
listenfd = socket (AF_INET, SOCK_STREAM, 0);
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = htonl (INADDR_ANY);
servaddr.sin_port = htons (SERVER_PORT)
bind(listenfd, (struct sockaddr *) &servaddr,
sizeof(servaddr));
listen(listenfd, LISTENQ);

connfd = accept(listenfd, (struct sockaddr *)
&cliaddr, &clilen);
recv(connfd, buf, sizeof(buf), 0);
/* Do data processing as per business logic */
send(connfd, buf, strnlen(buf, sizeof(buf)), 0);

close(connfd);
```

Simple TCP Client(tcpclient01.c)

```
sock = socket (AF_INET, SOCK_STREAM, 0);
servaddr.sin_family = AF_INET;
servaddr.sin_addr.s_addr = inet_addr(ip_addr);
servaddr.sin_port = htons (SERVER_PORT)

connect(sock, (struct sockaddr *) &servaddr,
sizeof(servaddr));
/* Send and receive data as per business logic*/
send(sock, buf, SEND_SIZE, 0);

recv(sock, buf, BUFFSIZE, 0);;

close(sock);
```

TCP Communication

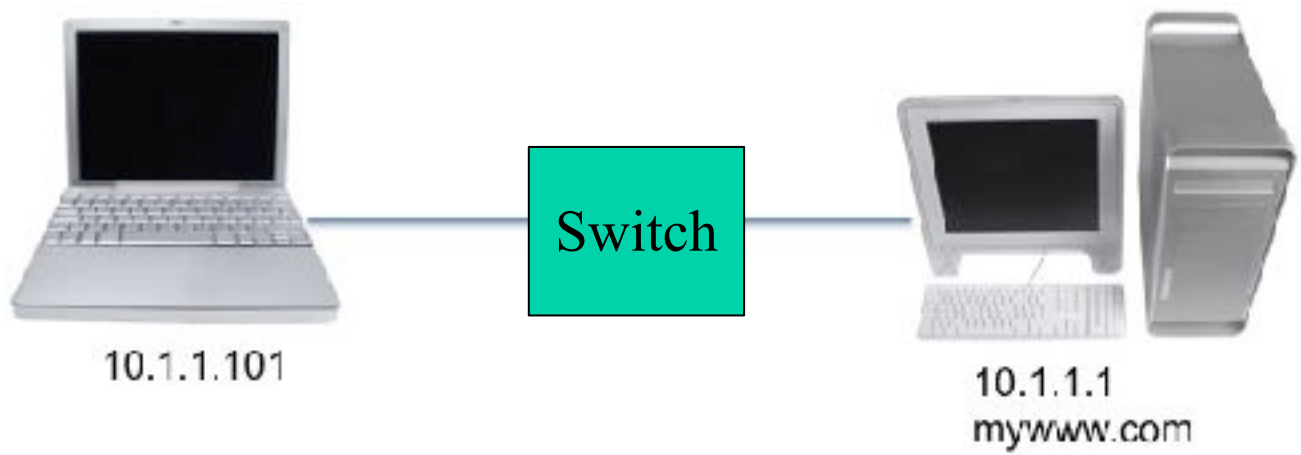
- What happens when client starts connection
  - Server not listening?
  - Server accepts only one connection and 2nd client connects?
  - Multiple client connects, server has initiated listen() but not accepted any connection.
- TCP Connection state on server

```
netstat -natp
```

Day 2: Basics of Socket Programming

- Overview: sockets
- Simple client server programs
- Handson-1: Writing TCP and UDP server
- Errors in socket programming
- Network byte order, buffer mgmt, socket close
- Handson-2: Network byte order, socket close
- Multiple concurrent client communication
- Socket call return value and reliability
- Handson-3: Handling concurrent clients, listen
- TCP Streaming and UDP Message boundary
- Handson-4: TCP streaming & UDP msg boundary
  Summary

Setup Requirement



Hands-On 1: TCP client/server

- Write a TCP server program
  - e.g. `tcpserver01.c`
- Use `nc` as a client to connect and exchange data
  - Use C client for more flexibility `tcpclient01.c`
- Use **wireshark** to analyze the TCP Connection setup, data exchange and tear down.
- Explore basic TCP connection states
  - use **netstat -nat** command
  - Server: LISTEN, ESTABLISHED, CLOSED
  - Clinet: ESTABLISHED, TIME_WAIT

Hands-On 1: TCP client/server

- Modify TCP server to add delay between bind and listen, listen and accept.:
  - `tcpserver02.c`
  - `tcpserver03.c`
- Use multiple **nc** clients (multiple windows) to connect to a single server.
  - Use tcp client for more flexible options
- Analyze how many clients can actively communicate?
- Analyze TCP states of multiple connections
- Use wireshark to analyze the TCP Connection setup, data exchange and tear down.

Hands-On 1: UDP client/server

- Write a simple TCP server
- Use multiple nc UDP (option -u) clients (multiple windows) to connect to a single UDP server.
- Analyze active communications with all the clients.
- Analyze the behaviour difference with TCP
- Use wireshark to analyze the UDP communication.
- Modify the data in such a way so as to have same checksum value.

Day 2: Basics of Socket Programming

- Overview: sockets
- Simple client server programs
- Handson-1: Writing TCP and UDP server
- Errors in socket programming
- Network byte order, buffer mgmt, socket close
- Handson-2: Network byte order, socket close
- Multiple concurrent client communication
- Socket call return value and reliability
- Handson-3: Handling concurrent clients, listen
- TCP Streaming and UDP Message boundary
- Handson-4: TCP streaming & UDP msg boundary
- Summary

TCP Communication - Errors

- Two server programs are invoked concurrently.
  - One fails.
  - bind() call fails. Why?
    - Understand socket uniqueness.
- Server program is invoked immediately (within a minute) after the exit of previous server pgm.
  - Exits with errors. why?
  - Understand TCP TIME_WAIT states?

TCP Communication - Errors

- More clients connect to server than specified in queue size in listen()?
  – Client timeout, Reset?
- Client connects to a port on which server is not listening.
  – Understand TCP reset
- Server bind() on one IP Address and client connects on another.
  – Use of IPADDR_ANY.

Day 2: Basics of Socket Programming

- Overview: sockets
- Simple client server programs
- Handson-1: Writing TCP and UDP server
- Errors in socket programming
- Network byte order, buffer mgmt, socket close
- Handson-2: Network byte order, socket close
- Multiple concurrent client communication
- Socket call return value and reliability
- Handson-3: Handling concurrent clients, listen
- TCP Streaming and UDP Message boundary
- Handson-4: TCP streaming & UDP msg boundary
- Summary

TCP Communication: Errors

- Server does not do htons(port).
  – which port number is assigned.
  – Understand network byte order

TCP Communication - Errors

- Program buffer not cleaned up before read
- Program buffer smaller than received data
- Program reads only partial data
- Program reads after client closes the connection
- Program reads when client is not sending data and external signal is received

Day 2: Basics of Socket Programming

- Overview: sockets
- Simple client server programs
- Handson-1: Writing TCP and UDP server
- Errors in socket programming
- Network byte order, buffer mgmt, socket close
- Handson-2: Network byte order, socket close
- Multiple concurrent client communication
- Socket call return value and reliability
- Handson-3: Handling concurrent clients, listen
- TCP Streaming and UDP Message boundary
- Handson-4: TCP streaming & UDP msg boundary
- Summary

Hands-On 2

- Write a server program without htons(port) on bind
- Interrupt (kill signal) a running server program
- Run server program with multiple IP addresses
- Connecting TCP client to a non-existent TCP server
- Connecting UDP Client to a non-existent UDP server
- Analyze the TCP/UDP Communication in wireshark

Day 2: Basics of Socket Programming

- Overview: sockets
- Simple client server programs
- Handson-1: Writing TCP and UDP server
- Errors in socket programming
- Network byte order, buffer mgmt, socket close
- Handson-2: Network byte order, socket close
- Multiple concurrent client communication
- Socket call return value and reliability
- Handson-3: Handling concurrent clients, listen
- TCP Streaming and UDP Message boundary
- Handson-4: TCP streaming & UDP msg boundary
- Summary

Concurrent TCP Clients

- Approach 1:
  - Spawn a child (or thread) for each new client
    - tcpserver06.c
    - Child process deals with the client.
    - Parent process wait for new client to connect
  - Number of concurrent clients limited by server resources
    - From a single client machine: max 64K. why?

Sample Server Program

```
for (; ; ) {
   connfd = accept(listenfd,
      (struct sockaddr *)&cliaddr, &clilen);
   if ( (childpid = fork()) == 0) { /*
child process */
      printf("new request from %s:%d\n”,
         inet_ntoa(cliaddr.sin_addr),
         ntohs(cliaddr.sin_port));
      close(listenfd);
      str_echo(connfd); /* child process */
      exit (0);
   }
}
```

Sample Server Program…

```
void str_echo(int connfd) {
   char buf[BUFFSIZE];
   int size;
   bzero(buf, sizeof(buf));
   while ((size = recv(connfd, buf,
sizeof(buf), 0) > 0)) {
      send(connfd, buf, strnlen(buf,
sizeof(buf)), 0);
      bzero(buf, sizeof(buf));
   }
   close(connfd);
}
```

Sample Server Program…

- Issues with Approach 1:
  - Works well for few clients
  - Scaling becomes an issue
    - Each child process acquires system resources
  - Parent needs to handle child process exit
    - Else it becomes a zombie process
- Solution:
  - tcpserver07.c
  - parent needs to do waitpid()

Sample server code:

```
for (; ; ) {
  /* Handle all zombie processes */
  while ((childpid = waitpid(-1,
&child_status, WNOHANG)) > 0) {
     /* Child exited */
  }
  connfd = accept(listenfd,
     (struct sockaddr *) &cliaddr, &clilen);
  if ((childpid = fork()) == 0) {
    /* child process code */
  }
}
```

Sample Server Code : tcpserver07.c

- Further issues
  - Till a new child connects,
    - old zombie children still exists
    - Zombie cleanup depends upon new client request
  - Solution: implement signal handling
    - Cleanup should happen on SIGCHLD
      - tcpserver08.c
- Creating a child process is a costly process from system perspective.
- Each child is independent
    - Sharing of resources could still be a herculean task

Sample server code: tcpserver08.c

```
signal(SIGCHLD, child_handler);

static void child_handler(int signo) {
    pid_t child_pid;
    int status;
    do {
        child_pid = waitpid(-1,
&status, WNOHANG);
    } while ( child_pid != -1 );

signal(SIGCHLD, child_handler);
}
```

Day 2: Basics of Socket Programming

- Overview: sockets
- Simple client server programs
- Handson-1: Writing TCP and UDP server
- Errors in socket programming
- Network byte order, buffer mgmt, socket close
- Handson-2: Network byte order, socket close
- Multiple concurrent client communication
- Socket call return value and reliability
- Handson-3: Handling concurrent clients, listen
- TCP Streaming and UDP Message boundary
- Handson-4: TCP streaming & UDP msg boundary
  Summary

Return value of socket calls

- recv()/read()
  - n (>0): number of bytes received
  - 0 - an orderly shutdown at the other end, like EOF
  - -1: an error, and `errno` set to error values
- write()/send()
  - n (>0) implies data is copied into kernel buffer
    - does not imply that data has gone to other end.
  - -1: an error, and `errno` set to error values

Day 2: Basics of Socket Programming

- Overview: sockets
- Simple client server programs
- Handson-1: Writing TCP and UDP server
- Errors in socket programming
- Network byte order, buffer mgmt, socket close
- Handson-2: Network byte order, socket close
- Multiple concurrent client communication
- Socket call return value and reliability
- Handson-3: Handling concurrent clients, listen
- TCP Streaming and UDP Message boundary
- Handson-4: TCP streaming & UDP msg boundary
- Summary

Hands-On 3

- Connect with multiple clients having varying lifetime
  – Can use nc as client
- Explore creation of zombie processes
- Explore cleaning of zombie processes
- Explore signal SIGCHLD to clear up child zombie process
- Analyze the network traffic in wireshark,

Day 2: Basics of Socket Programming

- Overview: sockets
- Simple client server programs
- Handson-1: Writing TCP and UDP server
- Errors in socket programming
- Network byte order, buffer mgmt, socket close
- Handson-2: Network byte order, socket close
- Multiple concurrent client communication
- Socket call return value and reliability
- Handson-3: Handling concurrent clients, listen
- TCP Streaming and UDP Message boundary
- Handson-4: TCP streaming & UDP msg boundary
- Summary

Transport Layer Protocol Characteristics

- Connection less
  - May arrive out of order
    - In order delivery requires pkts to be numbered.
  - No acknowledgement, Packets may be lost
  - No prior handshake
- Connection oriented
  - Setup, data transfer and teardown phase
  - Provides reliability, ordered delivery
  - Handles error control in a better way

Transport Layer Reliability Support

- Reliability
  - Needs error and flow control
    - Compels slower service
- Unreliable protocol
  - No extra overheads
- Reliability at data link layer
  - Provides error and flow control
  - Why do we need it at Transport layer when Link layer provides the same

UDP Overview

- No *frills, bare bones* Internet transport protocol
- B*est effort* service, UDP segments may be:
  - lost, delivered out-of-order to application
- Connectionless
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others
- UDP used by:
  - streaming apps (loss tolerant, rate sensitive)
  - DNS, SNMP
- Reliable transfer over UDP:
  - Requires reliability at application layer, application-

UDP Design Requirement

- How to design a simple transport layer ?
- Just provide transport on top of IP
  - Multiplexing and demultiplexing
  - Little bit of error checking
  - No handshake
- Rest all has to be managed by application
  - Application practically talks to IP
- DNS uses UDP
  - What happens when query/response is lost?

TCP Characteristics

- *point-to-point*: One sender, one receiver
- *reliable, in-order byte steam: No "message boundaries"*
- *pipelined: TCP congestion and flow control set window size*
- *full duplex data: Bi-directional data flow in same connection*
  - *MSS: maximum segment size, determined from link/ frame size*
- *connection-oriented*: Handshaking (exchange of control msgs) inits sender, receiver state before data exchange
- *flow controlled*: Sender will not overwhelm receiver

Day 2: Basics of Socket Programming

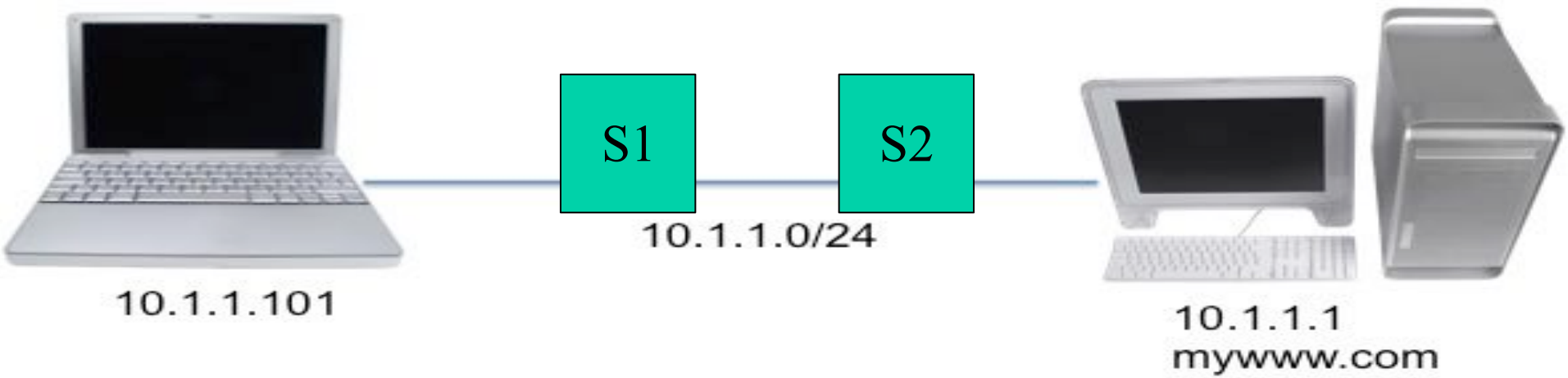
- Overview: sockets
- Simple client server programs
- Handson-1: Writing TCP and UDP server
- Errors in socket programming
- Network byte order, buffer mgmt, socket close
- Handson-2: Network byte order, socket close
- Multiple concurrent client communication
- Socket call return value and reliability
- Handson-3: Handling concurrent clients, listen
- TCP Streaming and UDP Message boundary
- Handson-4: TCP streaming & UDP msg boundary
- Summary

Hands-On 4:a

- Between two machines communicate UDP packets using `nc` (`-u` option).
- Send data content as below and compute the checksum using the **IP Address** and **port** number used and verify it with checksum value in wireshark capture.
  - "ABCDE" and
  - "ABCDEUQUQUQ"
- Are the two checksum same? Why or why not?
  - What other characters can you attach to "ABCDE" to have the same checksum value

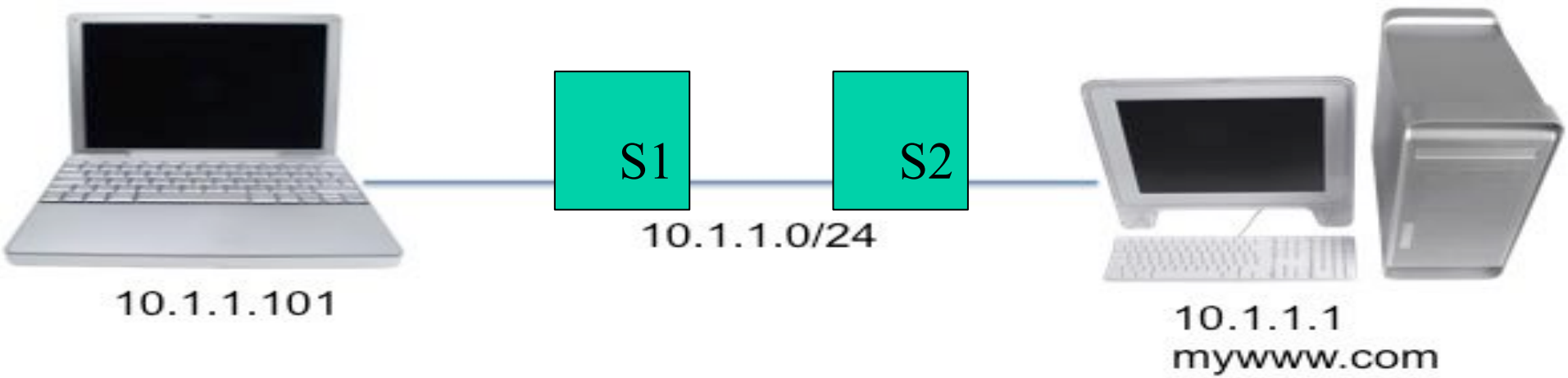
Hands-On 4b: UDP Packet Loss

- Data Transfer
  - Clients sends data (100 bytes) every 2s (10 times)
  - `AA`.. (1st pkt), `BB`..(2nd pkt), …, `JJ`..(10th pkt)
  - Break the link between switch at 7th sec and restore after 12 sec
- Server reads full 100 bytes.
- What would server receive and display?



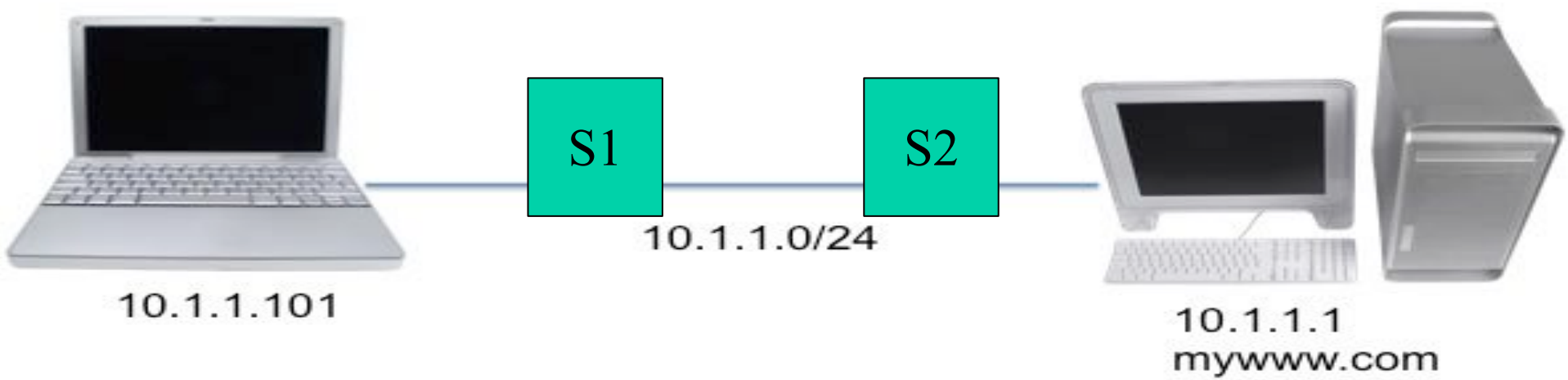
Hands-On 4c: UDP Messaging

- Data Transfer
  - Clients sends data (100 bytes) every 2s (10 times)
  - `AA`.. (1st pkt), `BB`..(2nd pkt), …, `JJ`..(10th pkt)
  - Break the link between switch at 7th sec and restore after 12 sec
- Server reads 50 bytes at a time
  - What would server receive and display?



Hands-On 4d: TCP Streaming and Packet Loss

- Data Transfer
  - Clients sends data (100 bytes) every 2s (10 times)
  - `AA`.. (1st pkt), `BB`..(2nd pkt), …, `JJ`..(10th pkt)
  - Break the link between switch at 7th sec and restore after `12th` sec
- Server reads 50 bytes at a time
- What would server receive:



Day 2: Basics of Socket Programming

- Overview: sockets
- Simple client server programs
- Handson-1: Writing TCP and UDP server
- Errors in socket programming
- Network byte order, buffer mgmt, socket close
- Handson-2: Network byte order, socket close
- Multiple concurrent client communication
- Socket call return value and reliability
- Handson-3: Handling concurrent clients, listen
- Understanding select() call
- Handson-4: write programs with select, timeouts
- Summary

Summary

- Sockets overview
- Evolutions of server programs
  - Simple server programs
  - Server program with one child process for each client
  - Issues with buggy server code (zombie processes)
  - Single program handling multiple concurrent clients
- UDP message boundary
- TCP message streaming and recovery on packet loss.

Thank You