# CS3530: Network Programming Basics

November 8th, 2019

Kotaro Kataoka

# Topics

- Unix Commands for Networking
- TCP Echo Client and Server
  Set1. Echo Client with only one action
  Set1. Simple Echo Server
  Set2. Echo Client with loop actions with BYE
  Set3. Echo Server with multiple clients (non-blocking mode)
- DNS?
- Assignment

# Unix Commands for Networking

Is you laptop connected to
IITH or IITH-Temp?

# ip and ss (1/3)

- Checking the network information of your NIC

```
$ ip addr show eth0
$ ip addr show
$ ifconfig eth0
$ ifconfig
```

- Configuring an IP address to your system manually if dynamic configuration is not available

```
$ ip addr add 192.168.0.30/27 dev eth0
```

# ip and ss (2/3)

- Checking the status your network

- Routing Table

```
$ ip route
$ netstat -rn
```

- Sockets
  - a: all listening and non-listening sockets
  - p: Process using socket

```
$ ss –ap
$ ss -ani
$ netstat –ap
$ netstat -ni
```

# ip and ss (3/3)

- Showing the mapping between IP address and MAC address

```
$ ip neighbor
$ ip -6 neigh
$ arp –n
```

- Normally the default gateway and computers on the same link are visible

# ping / traceroute

- ping: Checking if the destination is reachable

```
$ ping 192.168.1.1
```

- traceroute: Check the path to the destination

```
$ traceroute 192.168.1.1
-A Perform AS Path lookups
-I Use ICMP for probes
-T Use TCP for probes
```

# Socket programming *with TCP*

client must contact server

- server process must first be running
- server must have created socket (door) that welcomes client's contact

client contacts server by:

- Creating TCP socket, specifying IP address, port number of server process
- *when client creates socket:* client TCP establishes connection to server TCP

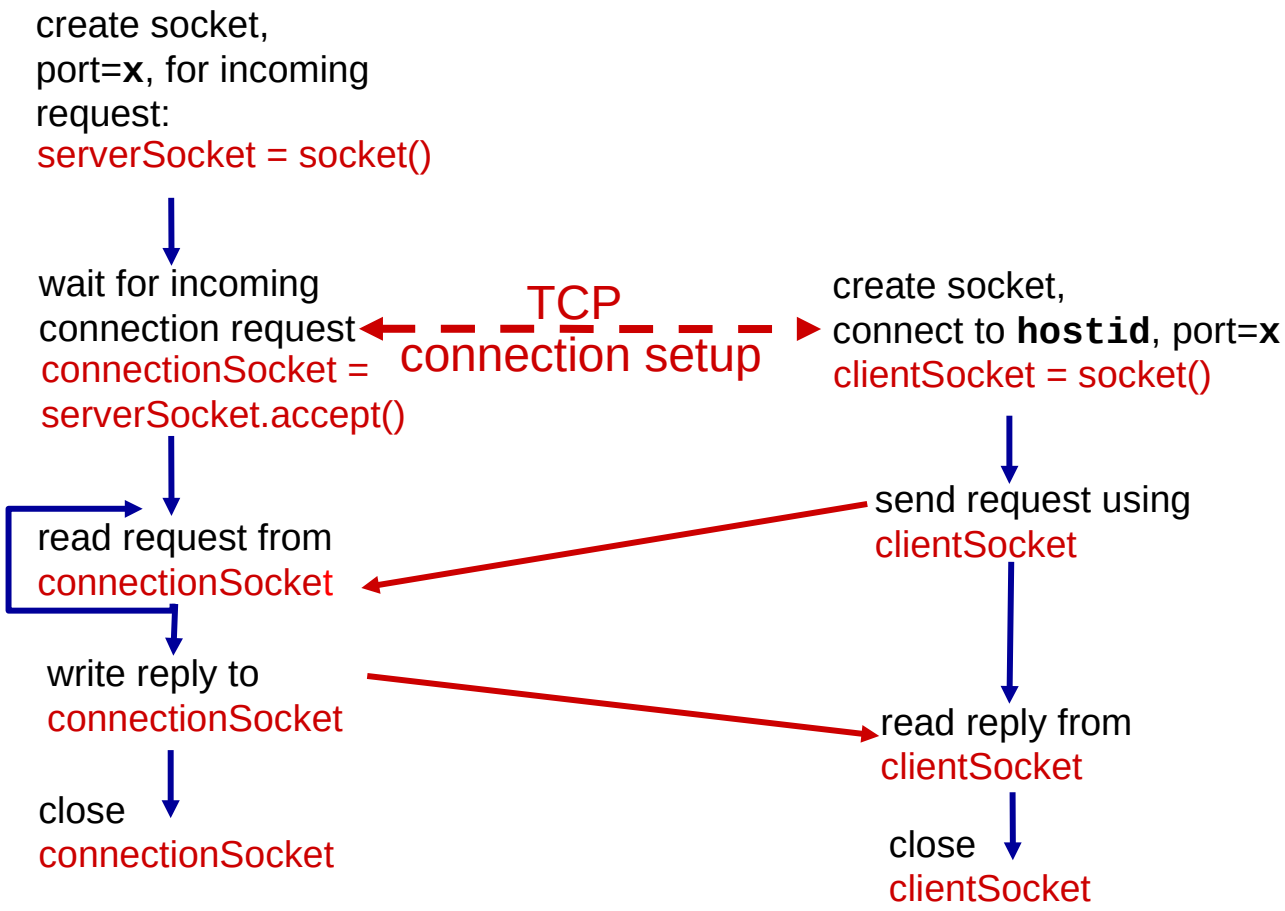
- when contacted by client, *server TCP creates new socket* for server process to communicate with that particular client
  - allows server to talk with multiple clients
  - source port numbers used to distinguish clients (more in Chap 3)

application viewpoint:

TCP provides reliable, in-order byte-stream transfer ("pipe") between client and server

# Client/server socket interaction: TCP

server (running on **`hostid`**) client



# Echo Client with only one action

1. Program starts with Server IP Address, Message and Server Port  main()
2. Create socket socket()
3. Set server parameters to socket
4. Connect to server  connect()
5. Send message  send()
6. Receive message  recv()
7. Show message echoed by server printf()
8. Destroy socket and die  close()

Let’s get started.

# socket() System Call (1/2)

- Cerates socket
- int socket(int family, int type, int proto)
  - Protocol Familiy
    - AF_LOCAL / PF_LOCAL Host-internal protocols
    - AF_INET / PF_INET Internet version 4 protocols
    - AF_ROUTE / PF_ROUTE Internal Routing protocol
    - AF_INET6 / PF_INET6 Internet version 6 protocols
    - etc
  - Socket Type
    - SOCK_STREAM
    - SOCK_DGRAM
    - SOCK_RAW
  - Protocol
    - Normally “0” except the case of RAW

# socket() System Call (2/2)

- Return value
  - Success: socket descriptor
  - Failure: -1

- Example

```
int sd;
sd = socket(AF_INET, SOCK_STREAM, 0);
if (sd < 0) {
something bad happened…
}
```

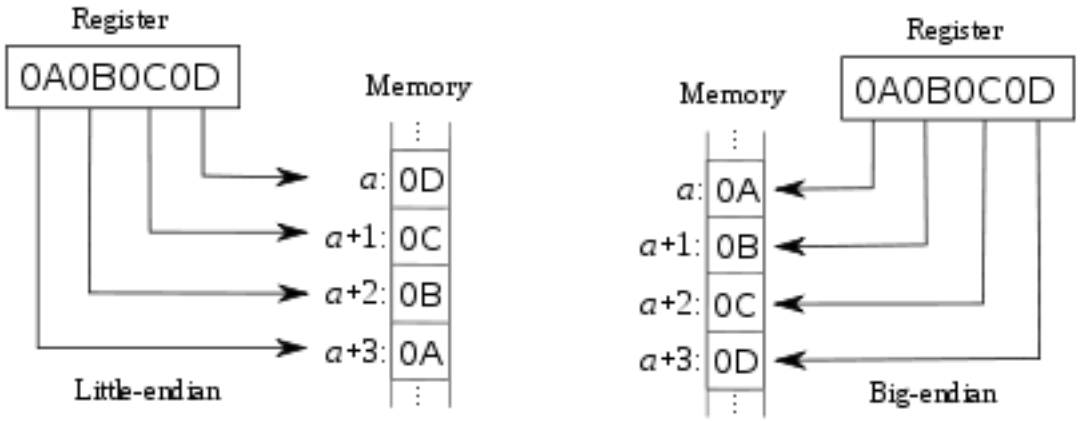
# struct sockaddr_in

- Specification of a local or remote endpoint

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/ip.h> /* superset of previous */

struct sockaddr_in {
    sa_family_t     sin_family;     /* address family: AF_INET */
    in_port_t       sin_port;   /* port in network byte order */
    struct in_addr  sin_addr;   /* internet address */
};

/* Internet address. */
struct in_addr {
    uint32_t        s_addr;     /* address in network byte order */
};
```

# inet_pton() Function

- Returns IP address in text format to that in binary format
  - Endianness
  - Network byte order (Big-endian)



# connect() System Call

- Initiate TCP connection on a socket

- Set the server address

```
struct sockaddr_in servAddr;
memset(&servAddr, 0, sizeof(servAddr));
servAddr.sin_family = AF_INET;
int err = inet_pton(AF_INET, servIP, &servAddr.sin_addr.s_addr);
if (err <= 0) {
perror("inet_pton() failed");
exit(-1);
}
servAddr.sin_port = htons(servPort);
```

- Connect to server

```
if (connect(sockfd, (struct sockaddr *) &servAddr, sizeof(servAddr)) < 0) {
perror("connect() failed");
exit(-1);
}
```

# Echo Server

1. Program starts with Server Port  main()
2. Create socket socket()
3. Set server parameters to socket  bind()
4. Wait for client  listen()
5. Establish TCP connection  accept()
6. Receive message  recv()
7. Send back message  (ECHO) send()
8. Repeat 5 to 7

# bind() System Call

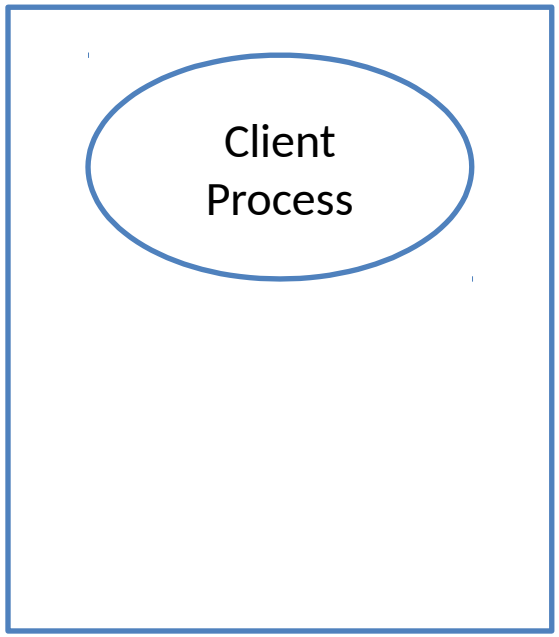
- Called on the server
- Bind a socket with a specific endpoint (me!!)

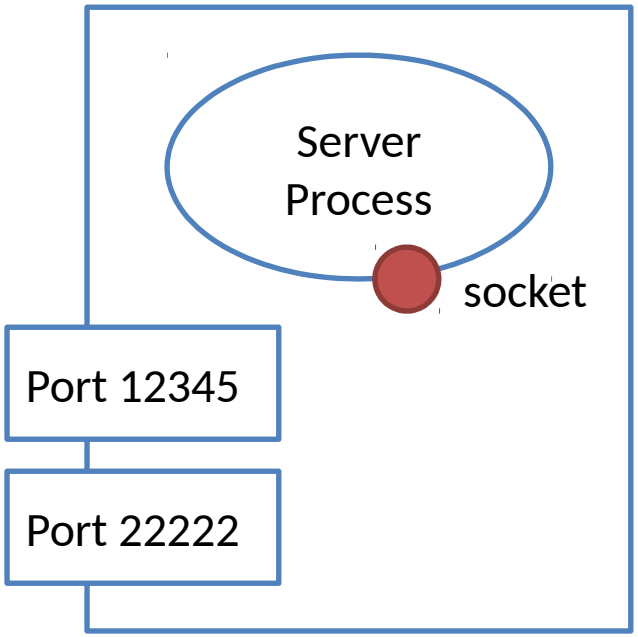
- int bind(int sockfd,struct sockaddr *addr,int addrlen);
- Example

```
struct sockaddr_in server;

memset((void *)&server, 0, sizeof(server));
server.sin_family = AF_INET;
server.sin_port = htons(12345);
server.sin_addr.s_addr = INADDR_ANY;

bind(sd,(struct sockaddr *)&server, sizeof(server));
```

# Creating Socket and Binding on Sever

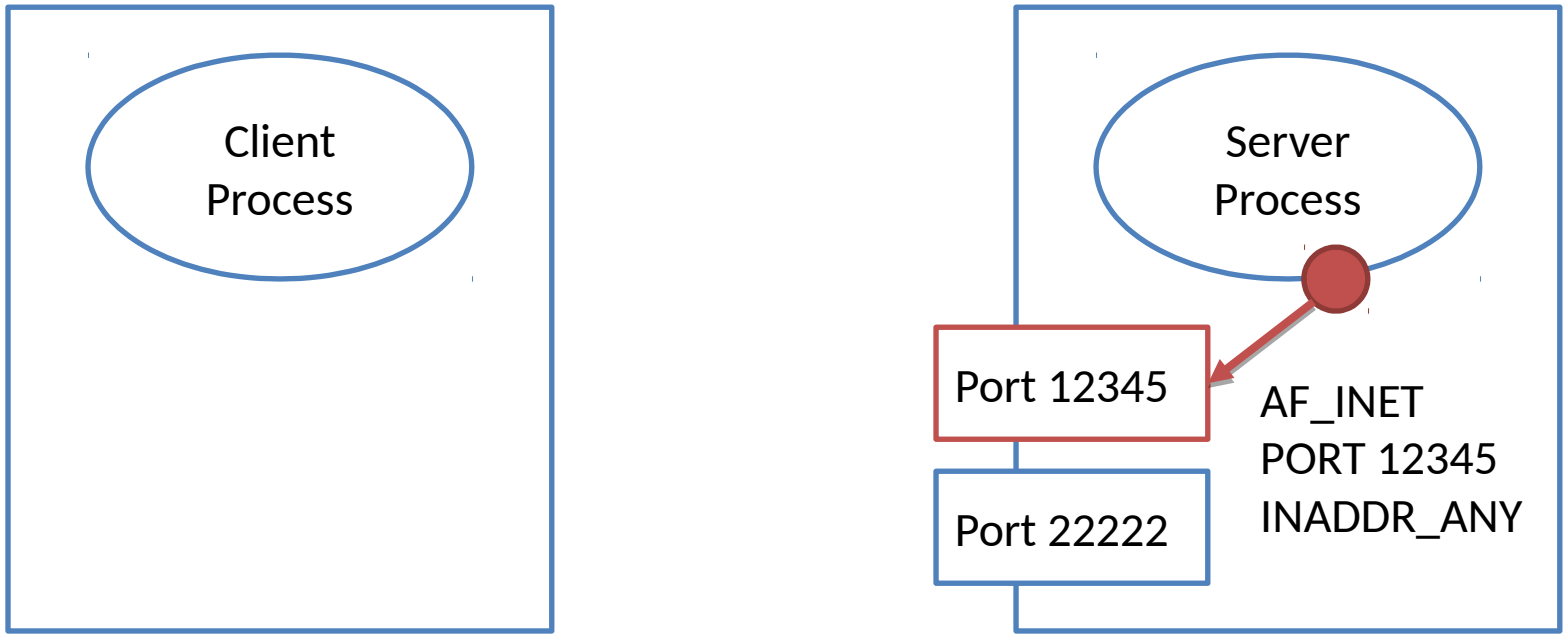
- socket() is called



# Creating Socket and Binding on Sever

- Bind to listening port on server




```
[sfc-cpu:7:29] netstat -an
Active Internet connections (including servers)
Proto      Recv-Q    Send-Q    Local Address  Foreign Address      (state)
udp4       0      0      *.12345          *.*
```

# listen() System Call

- int
  listen(int socket, int backlog);

- Listen on a socket and wait for a connection

# accept() System Call

- int
  accept(int socket, struct sockaddr *restrict address, socklen_t *restrict address_len);

- Accept a connection on a socket
- “address” contains the address of connecting host (i.e. client)

# send() and recv() System Call

- ssize_t
  send(int socket, const void *buffer, size_t length, int flags);

- ssize_t
  recv(int socket, void *buffer, size_t length, int flags);

- send() and recv() do not have an argument to store address information because they will be called after establishing connection (from and to are both known)

# Echo Client with Loop Action with BYE

- Extend the simple echo client

- You can repeat to type message
- You can stop your client by command “BYE”

- Program starts with Server IP Address and Server Port

# Echo Server with Multiple Clients

- Server serves for multiple clients
- Non blocking
  - Do not wait for the interrupt from a specific input
  - Use select()

# DNS?

GetAddrInfo.c

# IPv6 Address

```
#include <sys/socket.h>
#include <netinet/in.h>

struct sockaddr_in6 {
    sa_family_t     sin6_family;    /* AF_INET6 */
    in_port_t       sin6_port;      /* port number */
    uint32_t        sin6_flowinfo;  /* IPv6 flow information */
    struct in6_addr sin6_addr;      /* IPv6 address */
    uint32_t        sin6_scope_id;  /* Scope ID (new in 2.4) */
};

struct in6_addr {
    unsigned char   s6_addr[16];    /* IPv6 address */
};
```

# addrinfo

- Used to store the result of DNS lookup from resolver

```
struct addrinfo {
    int             ai_flags;
    int             ai_family;
    int             ai_socktype;
    int             ai_protocol;
    size_t              ai_addrlen;
    struct sockaddr  *ai_addr;
    char                *ai_canonname;
    struct addrinfo   *ai_next;
};
```

# getaddrinfo() (1/2)

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

struct addrinfo hints;          // Give initial hints to resolver
struct addrinfo *result, *rp;       // Result will be stored here

// Prepare the hints for resolver
memset(&hints, 0, sizeof(struct addrinfo));
hints.ai_family = AF_UNSPEC;     // IPv4 or v6
hints.ai_socktype = SOCK_STREAM;// Hardcoded TCP as dummy
hints.ai_protocol = IPPROTO_TCP;   // Hardcoded TCP as dummy
hints.ai_flags = AI_CANONNAME;       // I want canonical name!!
```

# getaddrinfo() (2/2)

```
int s = getaddrinfo(hostName, NULL, &hints, &result);

for (rp = result; rp != NULL; rp = rp->ai_next) {
    SHOW CORRESPONDING IPv4/v6 ADDRESS;
    or
    ATTEMPT connect() to IPv4/v6 ADDRESS;
}
```

- getaddrinfo() takes hostname/address and/or port number as the 1st and 2nd arguments.

# How does  work?

```
struct sockaddr_in *saIn4;
struct sockaddr_in6 *saIn6;
char addrString[INET6_ADDRSTRLEN];
memset(addrString, 0, sizeof(addrString));

switch (rp->ai_family) {
case AF_INET:
    saIn4 = (struct sockaddr_in *) rp->ai_addr;
    inet_ntop(rp->ai_family, &saIn4->sin_addr.s_addr, addrString, sizeof(addrString));
    break;

case AF_INET6:
    saIn6 = (struct sockaddr_in6 *) rp->ai_addr;
    inet_ntop(rp->ai_family, &saIn6->sin6_addr.s6_addr, addrString, sizeof(addrString));
    break;

...
```

# Assignment

# Compulsory Assignment #1: Extending echo client/server functions

Q-1. Add two features to Echo Client /Server, and demonstrate them. In the report, you must describe the new features with their benefit. Significance of the features will impact the marks given.

Q-2. Let two computers talk with each other. Select one mode.

- EASY Mode: 1 server and 1 client.
  Extend server program to accept typing on server itself. No echo.
- NORMAL Mode: 1 server and 2 clients.
  Extend server program to forward a message from client 1 to client 2. You may skip management of client ID.
- HARD Mode: 1 server and N clients.
  Say N clients may connect to the server. Client 1 wants to talk with client m. How do you manage multiple clients and select the one you want to talk to?

# Compulsory Assignment #2: Making echo client/server “protocol independent”

Q-3. Revise echo client and server to be protocol independent (support both IPv4 and IPv6).

- Hint 1: sockaddr is too small for sockaddr_in6. sockaddr_storage has enough size to support both sockaddr_in and sockaddr_in6. (You will see this in server side program.)
- Hint 2: integrate getaddrinfo to avoid typing IPv6 address on your CLI
- Hint 3: you may use hostname (IPv4: “localhost”, IPv6: “ip6-localhost” address to develop / demonstrate the software on Ubuntu. They’re written in “/etc/hosts”.

# Implementation Guidance

- Individual Assignment
- You may choose any programing language (C, JAVA, Python, Perl and etc.)
- The software must be based on Socket Programming.
- Wrappers of API must not be used (messaging etc). Use send/recv or read/write using TCP socket.
- Keep the record of **Reference**

# Submission Guidance

- Deadlines
  - Submission of Report to Google Classroom: TBD
  - Demo to TAs: TBD

- Requirement of Report
  - The core idea of your answer to each question. Better visibility like screenshot of application will be appreciated.
  - **Reference (web sites, books, etc.) which corresponds to each answer.** This time, especially for "Q-3", there are not huge variety of answers. That's why, the reference is important. **Answers without appropriate reference may not get mark.**
  - Human readable source code as separate files
  - README as a separate file so that TAs and instructor can compile source code and execute the binary anytime
  - Submit all files as one tar ball or zip ball.