Book: Unix network (W. Richard Stevens) Vol. 1

1/11/19

D.C. concepts (M.L. Liu)

D.C. in Java 9 (Raja Malleshwara Rao)

Topic:

1) TCP, UDP } socket programming
2)
3) Fundamental of distributed computing
4) synchronization
5) DC paradigms
6) Distributed object
7) Advanced DC paradigms
8) SOA (Service Oriented Architecture)
   - Value
   - Characteristic
   - SOA concepts
     - XML ltd
     - Web service
     - Service design principles
     - SOAP
     - WSDL (Web Service Discription Language)
     - Basic SOA architecture (BPEL, ESB)
     - Restful Web Service, micro service, data intensive computing
     - Message driven programming (JMS)

⇒ Inter process communication (IPC)
- pipe, FIFO, Shared memory

comm. between two related processes (created by fork()) (K)

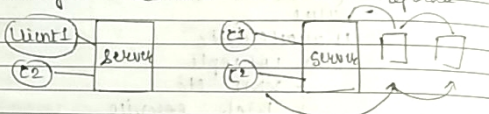
comm. between two unrelated processes (K)

faster (U)

U - user mode
K - Kernel mode

Task) Problem
↓
distributed into processes
(same or different machines)

For DC:
i) Task
ii) fixed medium
iii) Transfer medium

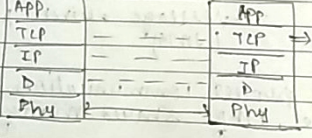
⇒ 

(Single point failure)

once the problem of server is fixed, the instance are destroyed, talks on previous server.

Basic ex. of DC: Socket prog
other ex.: Cloud computing, grid computing

TCP/UDP (Transport layer protocol)



Ack. must come from Transport layer.

Postal service: when receiver calls or writes back it not ack.

when the receiver signs for the package & this signed receipt is sent to the sender; it called ack.

SMS is also connection-less. We don't need to establish the connection for delivery or sending the message. Whatsapp messenger is connection oriented because it is end-to-end.

3/7/19 Client and server are user processes TCP & IP are part of protocol stack within the kernel.

TCP/IP layers:

i) Appl^n
ii) Transport
iii) Network
iv) Host to Network (DLL & Physical)

Read & write commands read/write from the TCP buffer (Transport layer) to the Appl^n (user process) or vice versa Acknowledgement and retransmission are a part of TCP/IP stack which resides in kernel space. It has nothing to do with user space.

Responsibilities of layers:

i) Physical
- Transmit message in form of electric signals.
- Amplification of signals.

ii) Host to Network
- Allows upper layers to access the media
- Controls how data is placed over the network (MAC)

iii) Network
- Sending packets across one or more networks
- best effort ~~efficient~~ delivery
- Non-reliable
- Reliability must be implemented by upper layer

Protocols
- App can bypass TCP or UDP
  ⇒ Directly use IP (called raw Socket)
  Ex - ping

- App can even bypass IP
  ⇒ Directly read/write DLL frames
  Ex - tcpdump

How to use Transport protocols? 4/7/19
→ Directly by app
→ Indirectly by services such as FTP, SMTP

POSIX std allows us to use the transport layer protocols

IETF is responsible for providing IPv4 & IPv6 to Unix.

Page No.
Date

Port Numbers - used to identify processes (2 bytes)

Special addresses:
127.0.0.1 - Loopback address
0.0.0.0 - Unspecified address

Multihomed hosts (Ex- Router)
A machine residing in two different networks. (2 NIC / 2 ethernet cables)

UDP - One socket can be used to send datagrams to different servers.

TCP waits for 1 round-trip time before resending the packet
↑
difference of time when packet sent and acknowledgement received. (ICMP Packet) [ping cmd]

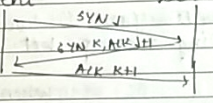
A|7|19 TCP Connection Establishment
Three way handshake occurs.
- Connection is always initiated by the client.

i) Server must be prepared to accept an incoming connection.
ii) Client issues an active open by calling connect.
iii) The server (TCP) must acknowledge (ACK) the client's (TCP) SYN and server (TCP) must also send its SYN.

Server - process is sending data
server (TCP) - TCP (buffer) residing on the server

→ Server in listening mode → passive open
→ To handle multiple clients - fork system call is used.
→ The no of clients connected to a server are maintained in the backlog queue (on the server)

TCP Options:
- MSS (Max. Segment Size) is for the connection



May have to fragment if conn. pkt is slow

SYN J - max. data that server can send to client at a time.
SYN K - max. data that client can send to server (buffer empty)

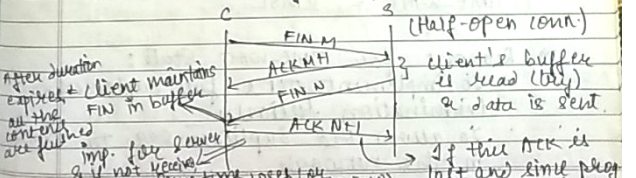
- Window Scale Option (is for the machine)
  - 1 GB is the max. transfer rate (Max. Window) and not the connection speed.
  - Used so that the sender cannot overflow the receiver's buffer.
- Timestamp option
  - Needed for high-speed conn.
  - One packet is sent and reaches a router

beyond which network is down. The router sends an ICMP and the sender in turn retransmit the packet through a different route. If both the packets make it to the destination at the same time, the packet with the latest timestamp is considered.

- And used to distinguish two packets having same sequence no.
- Can also be used to calculate time to live value.

⇒ 99% conn are established by client but, only 50% conn are terminated by the client.
If server does not have the right to close the conn, its active conn. queue may become full.

close system call only means that the app is not going to send anymore data but, it can receive FIN segment is sent. (In this time, the buffer is read and required data is sent by the receiver of FIN)



8/7/19

⇒ 11 states possible in TCP

⇒ If ACK of client's seq. is sent along with server's reply, it is called piggybacking

⇒ The end that performs active close enters in TIME-WAIT state. It maintains the contents of info. for some time. It keeps this in the buffer in case the ACK for FINN is lost. (App. layer buffers have been cleared)

If it does not wait then a new conn requested by the client can be destroyed when FINN is received from the server (which was not meant for this new conn. at all).

⇒ MSL (Max. Segment lifetime) - time that any IP datagram can live on Internet

TTL (Time to live): cannot exceed 255 hop count. TTL is the # of hop limit for IP packet.

TIME-WAIT = 2MSL

⇒ Reasons for TIME-WAIT State:
- To implement TCP's full duplex conn. termination reliably
- To allow old duplicate seq. to expire in the network

Page No.
Date

Port numbers.
Socket (IP address + port no.)

Ranges:
- well known : 0 - 1023 (controlled by IANA)
- Registered : 1024 - 49151 (registered with IANA)
- Dynamic : 49152 - 65535 (ephemeral ports)

Ex: while using Skype, the port no. is not controlled by IANA but, it is registered with it.

⇒ In Unix, ports less than 1024 can only be assigned by Super user process.
[Kernel can choose/assign IP add. & port no. {1024-5000}]

⇒ [*:21, *:*] → remote (client)
- ex-server
- kernel can choose any IP addr. but, port no. is fixed.
- (used when server is multi-homed)
- we can only know about this when a request comes in.

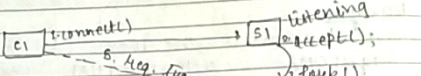
⇒ listening port of any server is always fixed.

⇒ client starts on host with IP: 206.168.112.219 and does an active open to the server's IP: 12.106.32.254
client TCP (kernel) assigns ephemeral port no. 1500
(server port no. : 21)

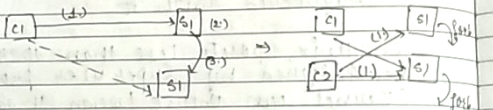
⇒ Multiplexing, demultiplexing → written layers ?

[If the machine is not in any network Kernel assigns IP addr. as 127.0.0.1]

⇒



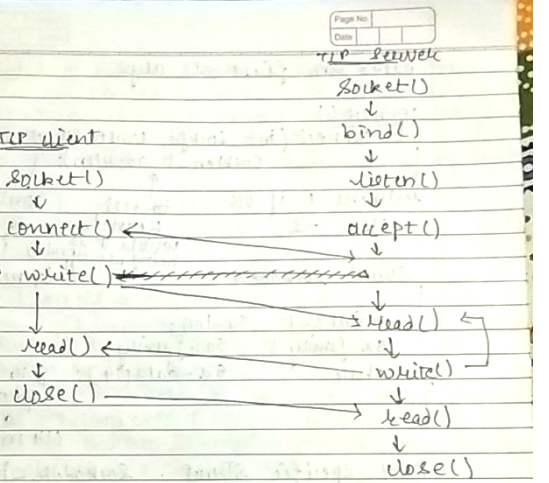
If listening of S1 (child) is not closed then:



Hence, generates 4 child processes

But since UDP is not reliable, we can handle many clients on a single server itself.

⇒ Seq. arrives with port no. 1500, it is delivered to first child.
Seq. arrives with port no. 1501, it is delivered to second child.
Any seq ~~getting for~~ coming from port no 21 is delivered to the listening socket (parent).



Socket func.

int socket (int family, int type, int protocol)
returns: non-negative descriptor if OK, -1 otherwise

family: AF_INET for IPV4
↑
Address family.

type: SOCK_STREAM for TCP
SOCK_DGRAM for UDP

protocol: IPPROTO_TCP - TCP
IPPROTO_UDP - UDP

If we don't want to write this explicitly write 0. Proto. is automatically choosen

based on first 2 args

connect()

int connect(int sockfd, const struct sockaddr *servaddr, socklen_t addrlen);

returns 0 if OK, else, -1

to tell Kernel. socklen_t is actually uint32_t

contains IP & port no info

struct sockaddr { // generic struct
    uint8_t sa_len;
    sa_family_t sa_family;
    char sa_data[14]; // 14 bytes of prot. specific addr (14 in case of IPv4)
};

Prot. specific struct : sockaddr_in (for IPv4)

sin_ → points to IPv4

192.168.3.2 - Not in numeric form. It is a string. Thus, we need to use structures.

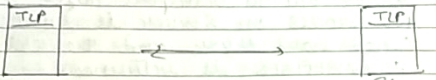
15/7/19 address family in the struct is used to specify family of server.
in_addr stores the 32-bit IP address of server.
family, port no. & IP addr are mandatory. Other variables may or may not be assigned.

bind() - Binds IP addr. & port no. to the process.

int bind(int sockfd, struct sockaddr*, socklen_t len)

⇒ Network should be Big Endian (well defined)

(byte ordering)



Client

m/c itself knows about its endianity. Thus, if it is little endian it calls hton (host to network) sys. call to convert it into big endian format. If m/c is big endian, this step is skipped.

Server

Since, server knows about its own endianity, it calls the sys. call ntoh (network to host) if it is little endian

htons - for 16-bit
htonl - for 32-bit

presentation format - 192.168.0.1
dotted decimal (presentation) → binary (network)

Server: ntop

const char *inet_ntop(int family, const void *addr, length)

Client: pton

Responses to connect()

i) ETIMEDOUT (Error time out)
Client's SYN did not reach server.
Client resends 8 times. (fixed)
after every (6 - 2 min)

ii) Error due to transport layer (RST)
No process on server is running
Called hard error and the error
ECONNREFUSED is returned.

If server is busy, the client's packet is simply dropped and no acknowledgement is sent.

EHOSTUNREACH is generated when no response is received and neither ICMP is received.

⇒ listen()
int listen (int sockfd, int backlog)
↑
maintains no. of conn. at a given time

Backlog is sum of 2 queues:
- Conn. which have completed 3-way handshake
- SYN received but, 3-way handshake remaining

(These queues are maintained by the kernel).

accept()
client's IP & port no. are read from TCP buffer and ntoh or ntop is called and result is put into appln layer variable (struct clientaddr)

getsockname()
To get IP addr. & port no. of process. (~~of~~ machine itself)

getpeername()
IP addr. & port no. of the remote machine (Ex- client's IP & port no. when called on server, after conn. is established)

```
/* echoTCPServer.c */
#include <sys/types.h>
#include <sys/socket.h>
#include <stdio.h>
#include <netinet/in.h>

#define MAXLINESIZE 100
#define SERV_PORT 5555

int listensd, clientsd;
char buffer[MAXLINESIZE+1];
struct sockaddr_in servaddr;
int noBytesRead = 0;

void processClient(int);

int main(){
```

```
if ((listenSd = socket(AF_INET, SOCK_STREAM,0))<0)
{
  fprintf(stderr, "Cannot create socket \n");
  exit(-1);
}

bzero(&servAddr, sizeof(servAddr));
servAddr.sin_family = AF_INET;
servAddr.sin_port = htons(SERV_PORT);
servAddr.sin_addr.s_addr = htonl(INADDR_ANY);

bind(listenSd, (struct sockaddr*)&servAddr,
                    sizeof(servAddr));

listen(listenSd, 5);

for(;;)                          sockaddr_in class can be used
{
  clientSd = accept(listenSd, (struct sockaddr*)NULL, NULL);
  if(fork()==0) { close(listenSd);
                  processClient(clientSd);
                  close(clientSd);
                  exit(0); } close(clientSd);
}
```

17/7/19 echoTCPClient.c

→ bzero function assigns '0' value to all its data members.

```
bzero(&servAddr, sizeof(servAddr));
```

(pointer (address) to the struct; size of the struct)

→ write(connectSd, sendBuffer, strlen(sendBuffer)+1);

(even if only 5 bytes are in sendBuffer 100 bytes are sent if sizeof is used then)

In server prog:

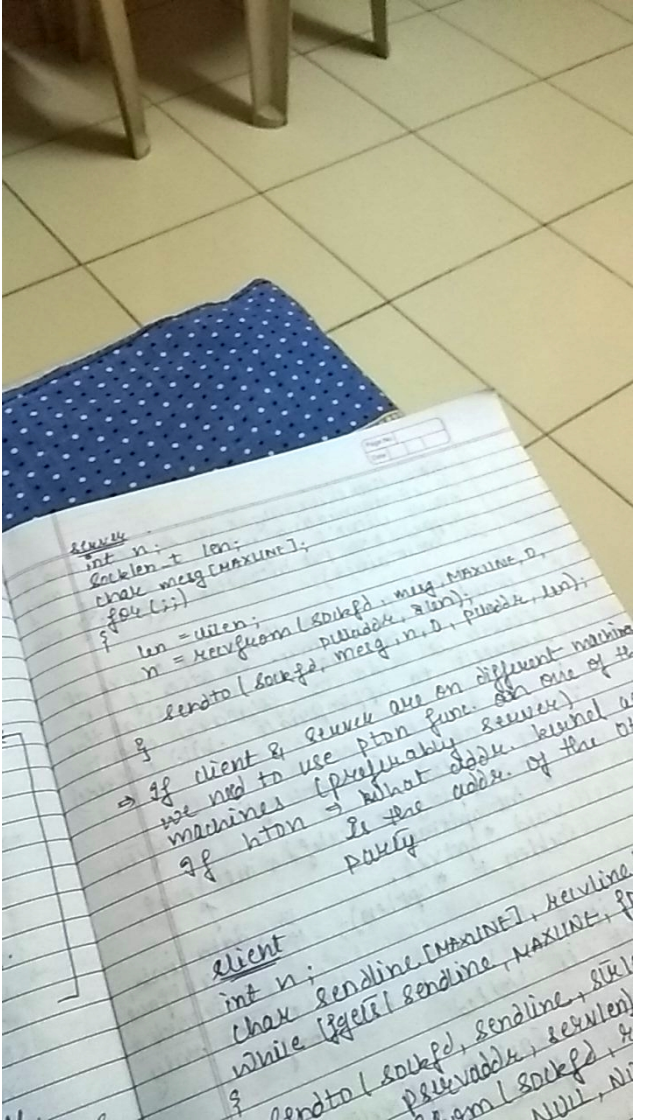
⇒ Addresses are assigned at transport layer and sockets exist at data link layer. Hence, assigning the family while assigning address is required.

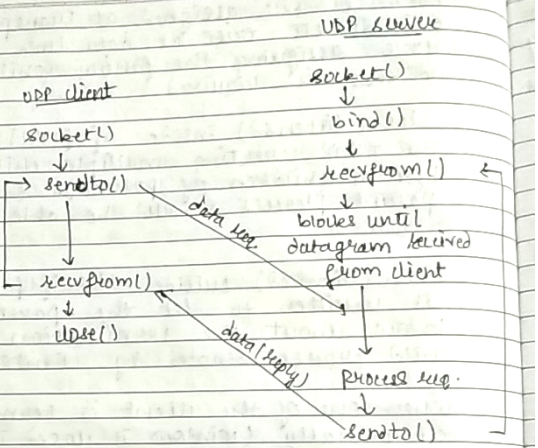
⇒ close(listensd) inside if(fork()==0) is to avoid creating multiple child processes (from children of parent). Only the parent process should be able to create a child.

⇒ close(clientsd) outside the if(fork()==0) is written to let the parent process know about the termination of the child process denoted by clientsd.

⇒ When one of the client is terminated abnormally (window is closed instead of ctrl+c), the server is not notified immediately and the child becomes a zombie process (Since, the descriptor connected/associated with the child process (of server) does not exist anymore but, the parent process does not know about it and assumes that its child is still communicating; it is said to be a zombie process).

22/7/19 UDP client/server





→ recvfrom()

```
ssize_t recvfrom(int sockfd, void *buff,
                 size_t nbytes, int flags,
                 struct sockaddr *from,
                 socklen_t *addrlen);
```

Stored in kernel space and to bring it into user's memory

INADDR_ANY - gives liberty to kernel to choose an address. If the addr. is not assigned, kernel assigns 127.0.0.1 to it.

Server

```
int n;
socklen_t len;
char mesg[MAXLINE];
for(;;)
{
    len = clilen;
    n = recvfrom(sockfd, mesg, MAXLINE, 0,
                 pcliaddr, &len);
    sendto(sockfd, mesg, n, 0, pcliaddr, len);
}
```

⇒ If client & server are on different machines we need to use pton func. on one of the machines (preferably server)
If hton → what addr. kernel assigns is the addr. of the other party

Client

```
int n;
char sendline[MAXLINE], recvline[MAXLINE+1];
while(fgets(sendline, MAXLINE, fp) != NULL)
{
    sendto(sockfd, sendline, strlen(sendline), 0,
           psrvaddr, servlen);
    n = recvfrom(sockfd, recvline, MAXLINE, 0,
                 NULL, NULL);
```

Server is the sender (we know) and we already know its IP (as we sent the data to it) so no need to receive its IP again.

Page No.
Date

23/7/19

Since, there is only one server, and it is communicating with a client, any other client's request is simply dropped. (Or put into the buffer, if one is maintained)

Multiplexing & demultiplexing occurs at transport layer only. In case of TCP it is due to this mechanism that the server is able to forward client requests to appropriate child (server) process.

To know about socket options after it is created:

```
int getsockopt (int sockfd, int level,
        int optname,
        void *optval,
        socklen_t *optlen);
```

level: 13 data link, transport, network

In case of TCP, it can be called only after it is created but, before connect is called.

For server, socket options can be set only before listen is called.

getpeername() is of no use in UDP and is called after connect on client and after accept on server.

23/7/19

optval - whether socket is broadcasting (value = 1) or not.

Binary options (True/False)
Value options (Specific values/structures).

Options:

SO_BROADCAST (binary opt.) (by default false) (can be get & set)
SO_ERROR (cannot be set)
SO_KEEPALIVE (for TCP only) (If there is no activity for 2 hours, server sends a msg. to that particular client. If ACK is not received, server closes the connection)
SO_RCVBUF, SO_SNDBUF - size of send & receive buffer
SO_LINGER - delays comm. protocol for some time. (for TCP only) when close is called

Options for network level
IP: Level = IPPROTO_IP

TCP: Level = IPPROTO_TCP
Options: SO_KEEPALIVE, NODELAY

⇒ If dest is also broadcast address, SO_BROADCAST has to be set to 1; else EACCES is returned.

Possible peer responses for KEEPALIVE:
- ACK (everything OK)
- RST (peer crashed & rebooted): ECONNRES

- No response to keep alive probe

RST
- Socket's pending error is set to ECONNRESET
- Socket is closed. (the socket of the sender who sent ~~ACK~~ KEEPALIVE probe)

No response
- 8 additional probes (each 75 sec. apart) are sent
- On any of these probes, if an ICMP is received (from a router), the router keeps the packet alive and probes 8 more times. If still no response then EHOSTUNREACH is returned.

Changing the waiting (keep alive) time is not advisable as these parameters are set based on per-kernel basis. On modification, it can cause problems on every host.

a keep alive probe
When ~~an ACK~~ does not get any response the same sender sends FIN because it may happen that the peer is rebooting and when it regains its control, it knows that the conn was terminated. Otherwise, it stays under the impression that it is still connected to that particular host

→ When a process completes, all the sockets are automatically closed but, if some data is present in the buffer, it can be lost. Thus, linger is used. It waits for 1 linger seconds before closing the socket completely (after close() is called)
If any data is found, the process is put to sleep until:
- either all data is send & ack by peer TCP
- Or linger time expires.

If socket is nonblocking (asynchronous) it will not wait for linger seconds and closes immediately.

The problem that if some data is left in the buffer even after linger seconds, it is discarded; is handled by SHUTDOWN.

Shutdown waits for FIN (only sent when every data has been read & written according to requirement).
SHT-RD
SHT-WR
SHT-RDWR

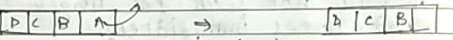
UDP does not send ack. but, if we want we can create an application level function which sends an intimation

(1 or 'Y' or 'yes') to the sender as soon as data is received (even if it further processes the data later)

Transport (segment)
↓ → in between exists an MTU which depends on the network config. (MTU - Max. Transfer Unit)
DLL (frame)

⇒

queue, if new packet arrives, it is dropped



A is read

even now, if a new packet arrives, it is dropped.

Hence, we need to use or implement a circular queue only.