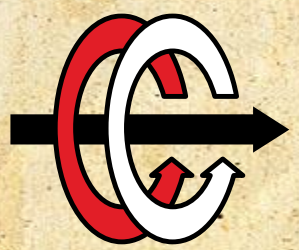
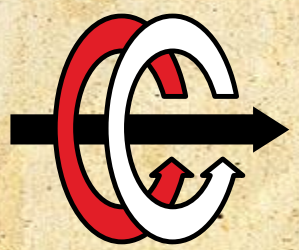


Socket Programming in C/C++

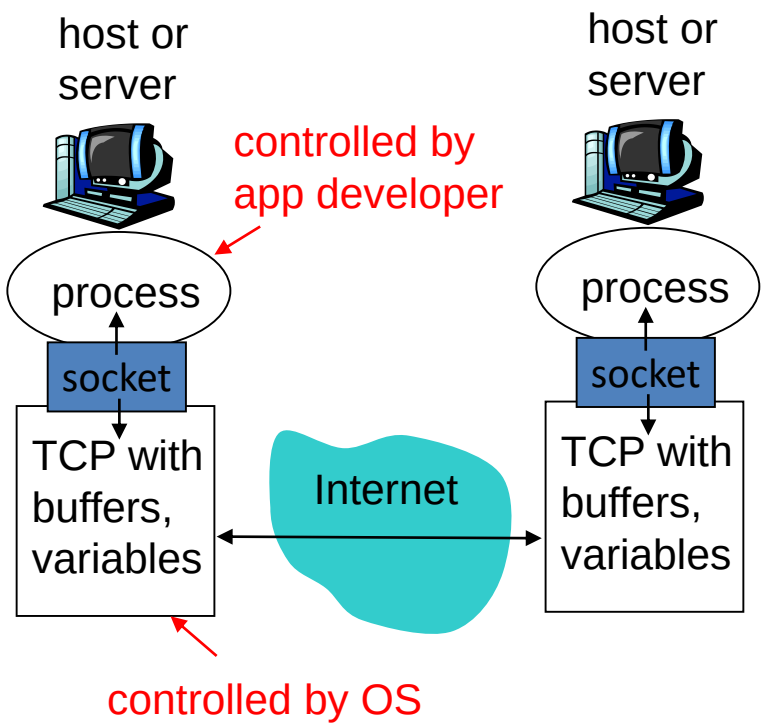


Sockets

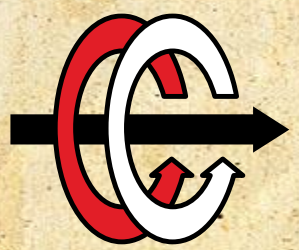
Definition & Characteristics

Introduction

Sockets are a protocol independent method of creating a connection between processes. Sockets can be either



- connection based or connectionless:
  Is a connection established before communication or does each packet describe the destination?
- packet based or streams based:
  Are there message boundaries or is it one stream?
- reliable or unreliable.
  Can messages be lost, reordered, or corrupted?



Socket characteristics

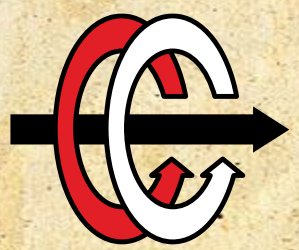
Domain, type and transport protocol.

Common domains

- **AF_INET** (ARPA Internet protocols)
- **AF_UNIX** (UNIX internal protocols)
- **AF_ISO** (ISO protocols)
- **AF_NS** (Xerox Network Systems protocols)

Common types

- **SOCK_STREAM** (virtual circuit: received in order, reliably)
- **SOCK_DGRAM** (datagram: arbitrary order, unreliable)



Socket characteristics (cont'd)

Each socket type has one or more protocols. Ex:

- **TCP/IP** (virtual circuits)
- **UDP** (datagram)

Use of sockets:

- Connection–based sockets communicate client-server: the server waits for a connection from the client
- Connectionless sockets are peer-to-peer: each process is symmetric.

Socket Functions

socket: creates a socket of a given domain, type, protocol (buy a phone)

bind: assigns a name and an address to the socket (get a telephone number)

listen: specifies the number of pending connections that can be queued for a server socket. (call waiting allowance)

accept: server accepts a connection request from a client (answer phone)

connect: client requests a connection request to a server (call)

send, sendto: write to connection (speak)

recv, recvfrom: read from connection (listen)

close: end the call

Socket Functions

socket: creates a socket of a given domain, type, protocol (buy a phone)

bind: assigns a name and an address to the socket (get a telephone number)

listen: specifies the number of pending connections that can be queued for a server socket. (call waiting allowance)

accept: server accepts a connection request from a client (answer phone)

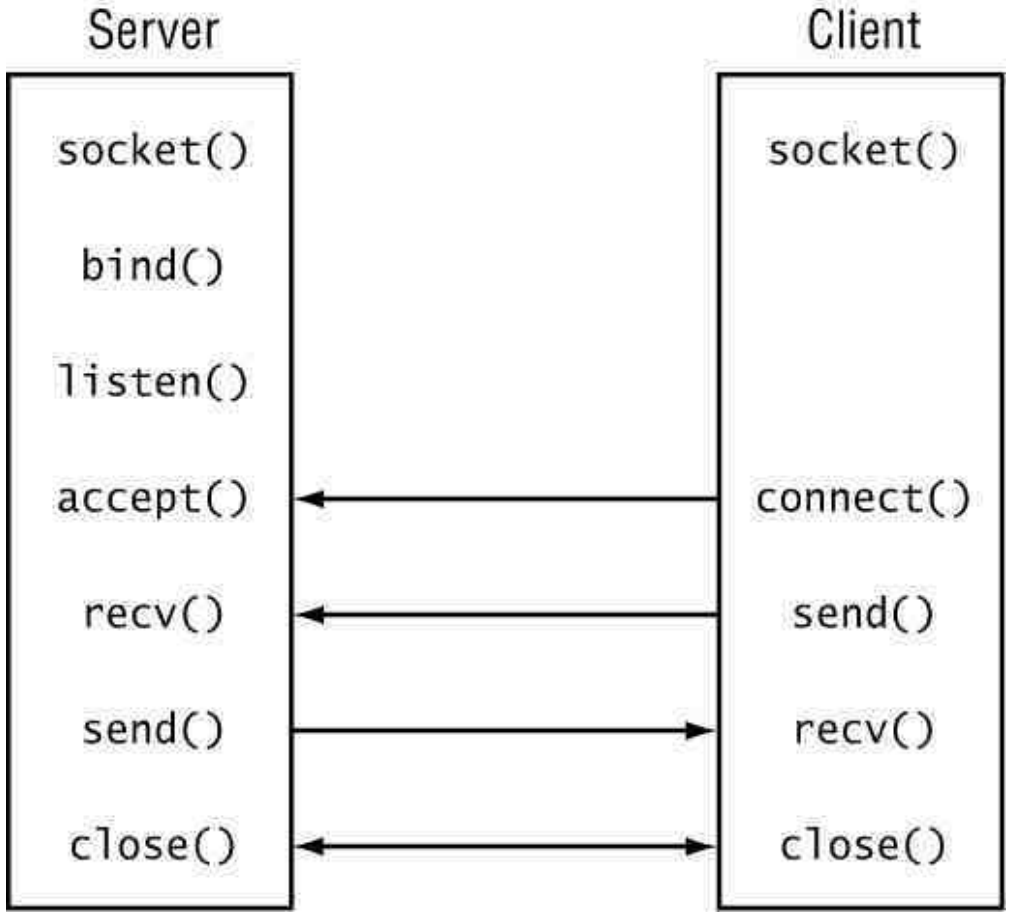
connect: client requests a connection request to a server (call)

send, sendto: write to connection (speak)

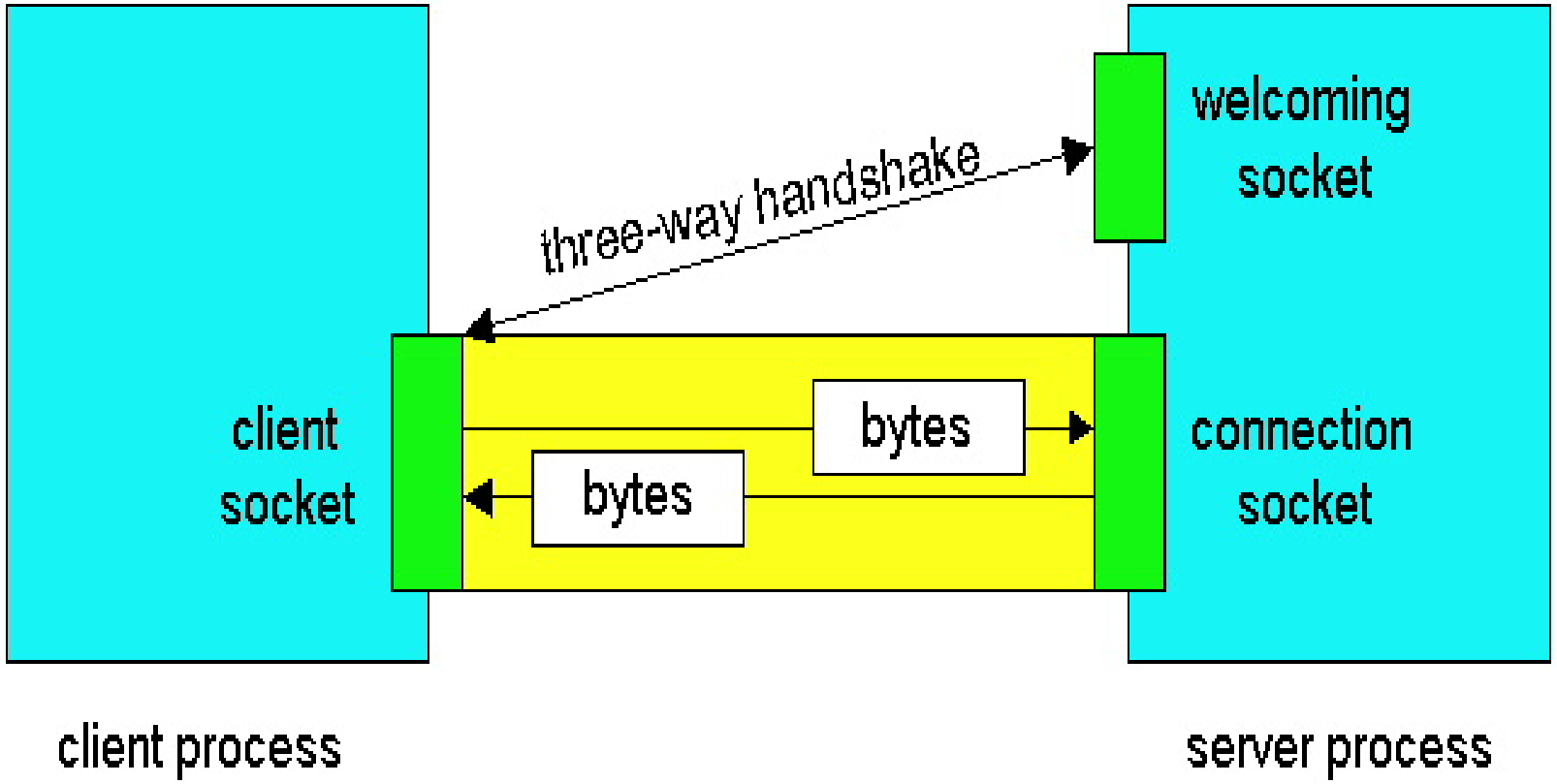
recv, recvfrom: read from connection (listen)

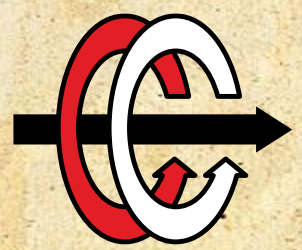
close: end the call

TCP-based sockets

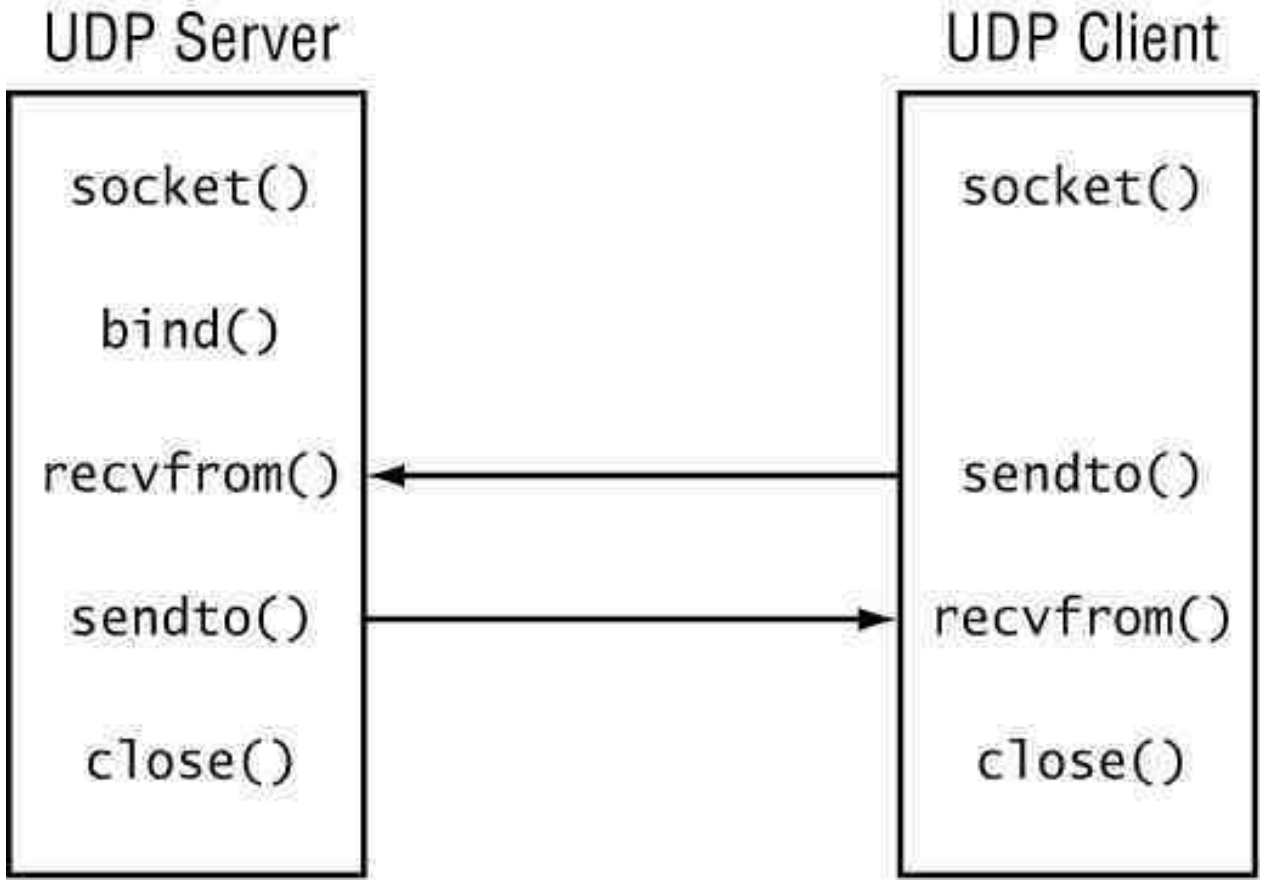


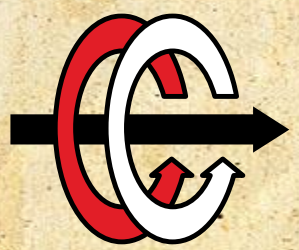
Accept (three-way handshake)





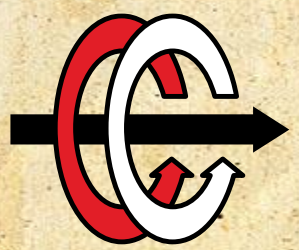
UDP-based sockets





Socket API

Functions

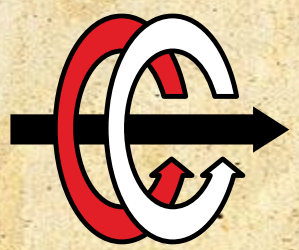


socket

int socket (int domain , int type , int protocol) ;

Returns a file descriptor (socket ID) if successful, -1 otherwise.

- domain is AF_INET for internet communication to IP addresses
- The type argument can be:
  - SOCK_STREAM: Establishes a virtual circuit for stream
  - SOCK_DGRAM: Establishes a datagram for communication
  - SOCK_RAW (IP level).
- protocol specifies the protocol used. It is usually 0 to use the default protocol for the chosen domain and type.



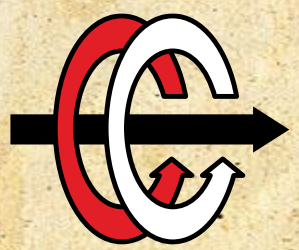
bind

int bind (int sid , struct sockaddr *addrPtr , int len)

- sid: is the socket id
- addrPtr: is a pointer to the address family dependent address structure
- len: is the size of *addrPtr

Associates a socket id with an address to which other processes can connect. In internet protocol the address is

[ipNumber, portNumber]

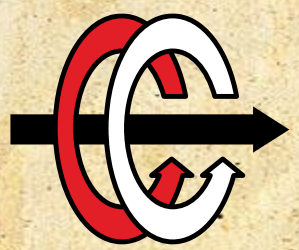


connect

this is the first of the client calls

int connect (int sid, struct sockaddr * addrPtr, int len)

- Specifies the destination to form a connection with (addrPtr), and returns a 0 if successful, -1 otherwise.



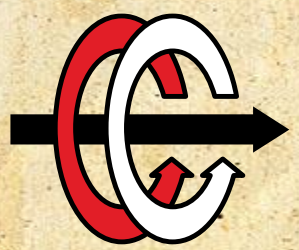
Port usage

Note that the initiator of communications needs a fixed port to target communications.

This means that some ports must be reserved for these “well known” ports.

Port usage:

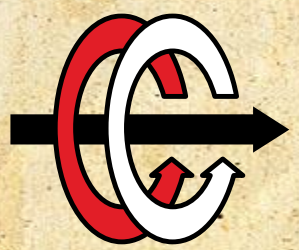
- 0-1023: These ports can only be bound by root
- 1024-49151: well known ports
- 49152-65535: ephemeral (short-term) ports for OS



listen

int listen (int sid , int size)

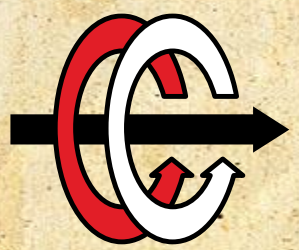
- Where size is the number of pending connection requests allowed (typically limited by Unix kernels to 5).
- Returns 0 on success, -1 on failure.



accept

int accept (int sid, struct sockaddr *addrPtr, int* lenPtr)

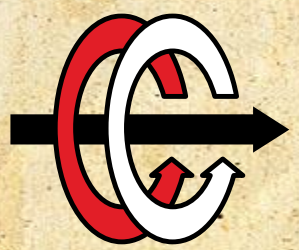
- Returns the socketId and address of client connecting to socket. If lenPtr or addrPtr equal zero, no address structure is returned. lenPtr is the maximum size of address structure that can be called, returns the actual value.
- Waits for an incoming request, and when received creates a socket for it.



send

int send (int sid, const char * bufferPtr, int len, int flag)

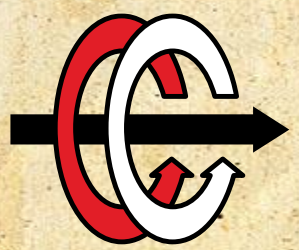
- Send a message. Returns the number of bytes sent or -1 if failure. (Must be a bound socket).



sendto

int sendto (int sid, const void *bufferPtr, size_t bufferLength, int flag, struct sockaddr *addrPtr, socklen_t addrLength)

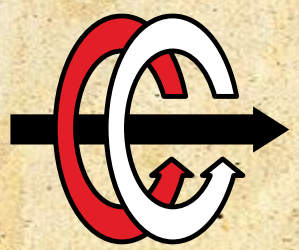
- for connectionless protocols
- sends a buffer, at bufferPtr, of length bufferLength to the address specified by addrPtr of size addrLength.
- returns number of bytes sent or -1 on error.



recv

int recv (int sid, const char * bufferPtr, int len, int flag)

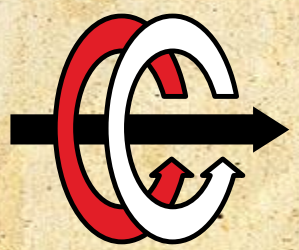
- Receive up to len bytes in bufferPtr. Returns the number of bytes received or -1 on failure.



recvfrom

int recvfrom (int sid, void *bufferPtr, int bufferLength, int flag, sockaddr *addrPtr, int *addrLengthPtr)

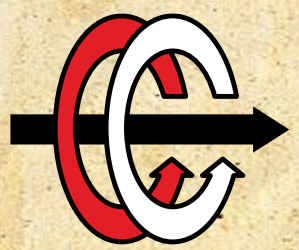
- for connectionless protocols
- receive a buffer in bufferPtr of maximum length bufferLength from an unspecified sender.
- sender address returned in addrPtr, of size *addrLengthPtr.
- returns number of bytes received or -1 on error.



close

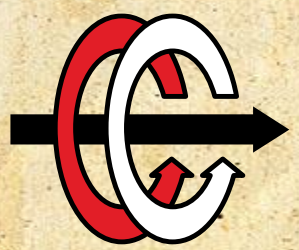
int close (int sid);

Both client and server close the sockets after data transfer



Programming

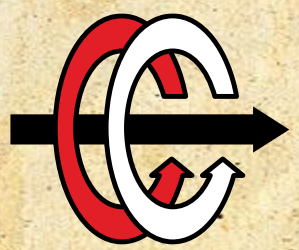
APIs and Examples



APIs for managing names and IP addresses

A number of auxiliary APIs used:

- The hostent structure: describes IP, hostname pairs
- gethostbyname: hostent of a specified machine
- htons, htonl, ntohs, ntohl: byte ordering
- inet_pton, inet_ntop, inet_addr: conversion of IP numbers between presentation and strings



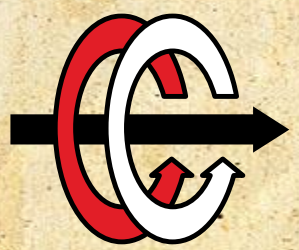
sockaddr

For the internet family:

```
struct in_addr {
long s_addr;
};
```

```
struct sockaddr_in {
int  sin_family;            // AF_INET identifiers
int  sin_port;              // port number,
                            // if 0 then kernel chosen
struct in_addr sin_addr; // IP address
                            // INADDR_ANY refers to the IP
                            // addresses of the current host
char sin_zero[8];           // Unused, always zero
};
```

server.sin_addr.s_addr=inet_addr("127.0.0.1");



hostent

```
struct hostent   {
  char*   h_name;      //official (canonical) name of the host
  char **h_aliases;    //null terminated array of alternative hostnames
  int      h_addrtype; //host address type AF_INET or AF_INET6
  int      h_length;   //4 or 16 bytes
  char **h_addr_list; //IPv4 or IPv6 list of address
}
```

- Error is return through h_error which can be:
  - HOST NOT FOUND
  - TRY AGAIN
  - NO RECOVERY
  - NO DATA

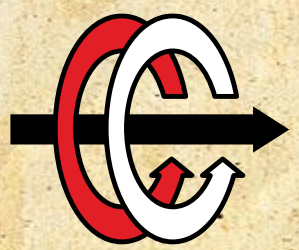
gethostbyname

Struct hostent* gethostbyname (const char* hostname)

- Auxiliary functions
- Translates a DNS name into a hostent.

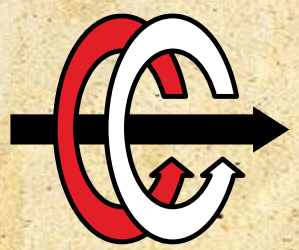
- Example:

```
Struct hostent* hostEntity =  gethostbyname(“eee.metu.edu.tr”);
Memcpy (socketAddr->sin_addr, hostEntity->h_addr_list[0], hostEntity->h_length);
```



Network byte ordering

```
unsigned long htonl(unsigned long n)
        // host to network conversion of a 32-bit value;
unsigned short htons(unsigned short n)
        // host to network conversion of a 16-bit value;
unsigned long ntohl(unsigned long n)
        // network to host conversion of a 32-bit value;
unsigned short ntohs(unsigned short n)
        // network to host conversion of a 16-bit value.
```



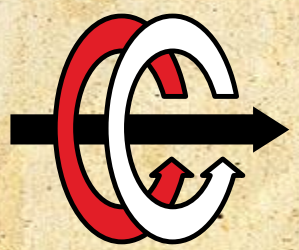
IP Number translation

inet_pton, inet_ntop, inet_addr routines translate between the address as a string and the address as the number.

We have 4 representations:

- IP number in host order
- IP number in network order
- Presentation (ex. dotted decimal)
- Fully qualified domain name

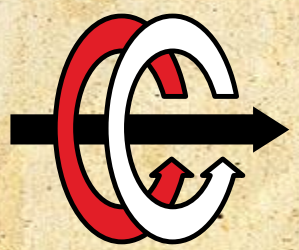
Only the last needs an outside lookup to convert to one of the other formats.



Example: TCP/IP Server Code

Without error checking.

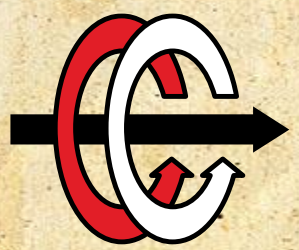
```
sockaddr_in serverAddr ;
sockaddr &serverAddrCast  = (sockaddr &)  serverAddr ;
  // get a tcp/ip socket
serverAddr.sin_family = AF_INET;
  // get an address from any internet interface on this server.
serverAddr.sin_addr.s_addr = htonl (INADDR_ANY) ;
serverAddr.sin_port = htons(13) ;
int  listenFd = socket (AF_INET , SOCK_STREAM, 0 ) ;
bind (listenFd , &serverAddrCast, sizeof ( serverAddr ) ) ;
Listen (listenFd, 5 );
for (   ;   ;   ){
int connectFd = accept (listenFd, (sockaddr*) NULL, NULL ) ;
   // read and write operations on connectFd
close(connectFd) ;
}
```



Example: TCP/IP Client Code

Without error checking.

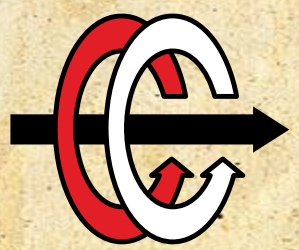
```
sockaddr_in serverAddr ;
sockaddr &serverAddrCast  = (sockaddr &)  serverAddr ;
  // get a tcp/ip socket
serverAddr.sin_family = AF_INET;
  // host IP # in dotted decimal format !
serverAddr.sin_addr.s_addr = inet_addr("144.122.167.37");
serverAddr.sin_port = htons(13) ;
connect (sockFd, serverAddrCast, sizeof (serverAddr));
  // . . read and write operations on sockFd . .
close(connectFd ) ;
}
```



Example: UDP Server

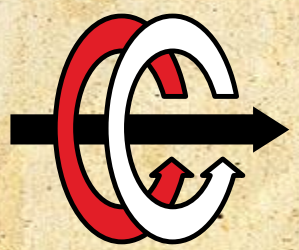
```
int socketId = socket (AF_INET , SOCK_DGRAM, 0 ) ;
sockaddr_in serverAddr , clientAddr ;
sockaddr &serverAddrCast = ( sockaddr &) serverAddr ;
sockaddr &clientAddrCast = ( sockaddr &) clientAddr ;
// allow connection to any addr on host for hosts with multiple network connections and ast server port .
serverAddr.sin_family = AF_INET ;
serverAddr.sin_port = htons ( serverPort ) ;
serverAddr.sin_addr.s_addr = INADDR_ANY;
bind ( socketId, &serverAddrCast, sizeof (addr) ) ;

int size = sizeof(clientAddr);
recvfrom ( socketId , buffer , bufferSize , 0 , clientAddrCast , &size ) ;
sendto ( socketId , buffer , bufferSize , 0 , clientAddrCast , size ) ; // reply to the client just received from
close(socketId);
```



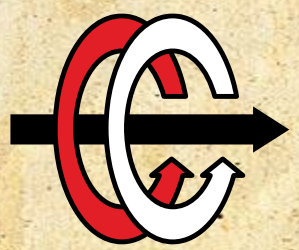
Example: UDP Client

```
int socketId = socket (AF_INET , SOCK_DGRAM, 0 ) ;
sockaddr_in serverAddr , clientAddr ;
sockaddr &serverAddrCast = ( sockaddr &) serverAddr ;
sockaddr &clientAddrCast = ( sockaddr &) clientAddr ;
// specify server address and port
serverAddr.sin_family = AF_INET ;
serverAddr.sin_port = htons ( serverPort ) ;
struct hostent *hp = gethostbyname ( hostName ) ;
memcpy ((char *) &serverAddr.sin_addr, ( char *)hp−>h_addr , hp−>h_length ) ;
// no need to bind if not peer-to-peer
int size = sizeof(serverAddr);
sendto ( socketId , buffer , bufferSize , 0 , serverAddrCast , size ) ;
Recvfrom (socketId , buffer , bufferSize , 0 , serverAddrCast , &size );
close(socketId);
```



Timeout for Blocking Functions

- Recv() or recvfrom() waits until some data arrives
- If arriving packet is lost (especially for UDP), recvfrom() function waits forever
- This can be avoided with timeout functions
  - Select, poll, alarm are some example functions.



THANKS

Adopted from; **Mani Radhakrishnan and Jon Solworth**

Socket Programming in C/C++; September 24, 2004

Prepared by **Yiğit Özcan**; April, 2013

Edited by **Utku Civelek**; April, 2016