LECTURE 3.
SOCKET API INTRODUCTION

References:

Chapter 3 & Chapter 11: Unix network programming

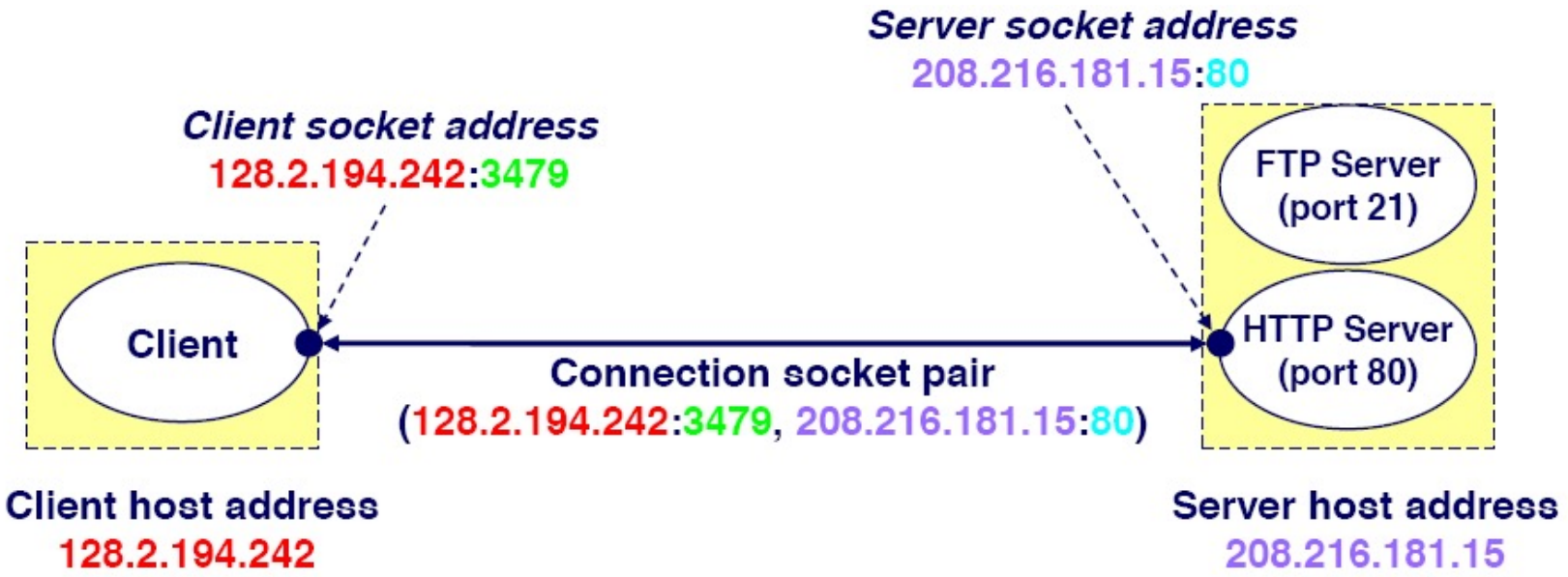
Content

- Socket
- Stream Socket
- Datagram Socket
- APIs for managing names and IP addresses
- Socket Address Structures

Socket

- What is a socket ?
- *Sockets* (in plural) are an application programming interface (API) application program at the TCP/IP stack
- A *socket* is an abstraction through which an application may send and receive data
- A socket allows an application to plug in to the network and communicate with other applications that are plugged in to the same network.

Socket (cont)


Server socket address
208.216.181.15:80
Client socket address
128.2.194.242:3479
FTP Server
(port 21)
HTTP Server
(port 80)
Client
Connection socket pair
(128.2.194.242:3479, 208.216.181.15:80)
Client host address
128.2.194.242
Server host address
208.216.181.15

Socket (cont)

- The main types of sockets in TCP/IP are
  - *stream sockets :* use TCP as the end-to-end protocol (with IP underneath) and thus provide a reliable byte-stream service
  - *datagram sockets :* use UDP (again, with IP underneath) and thus provide a ***best-effort*** datagram service
- Socket Address : include host name and port

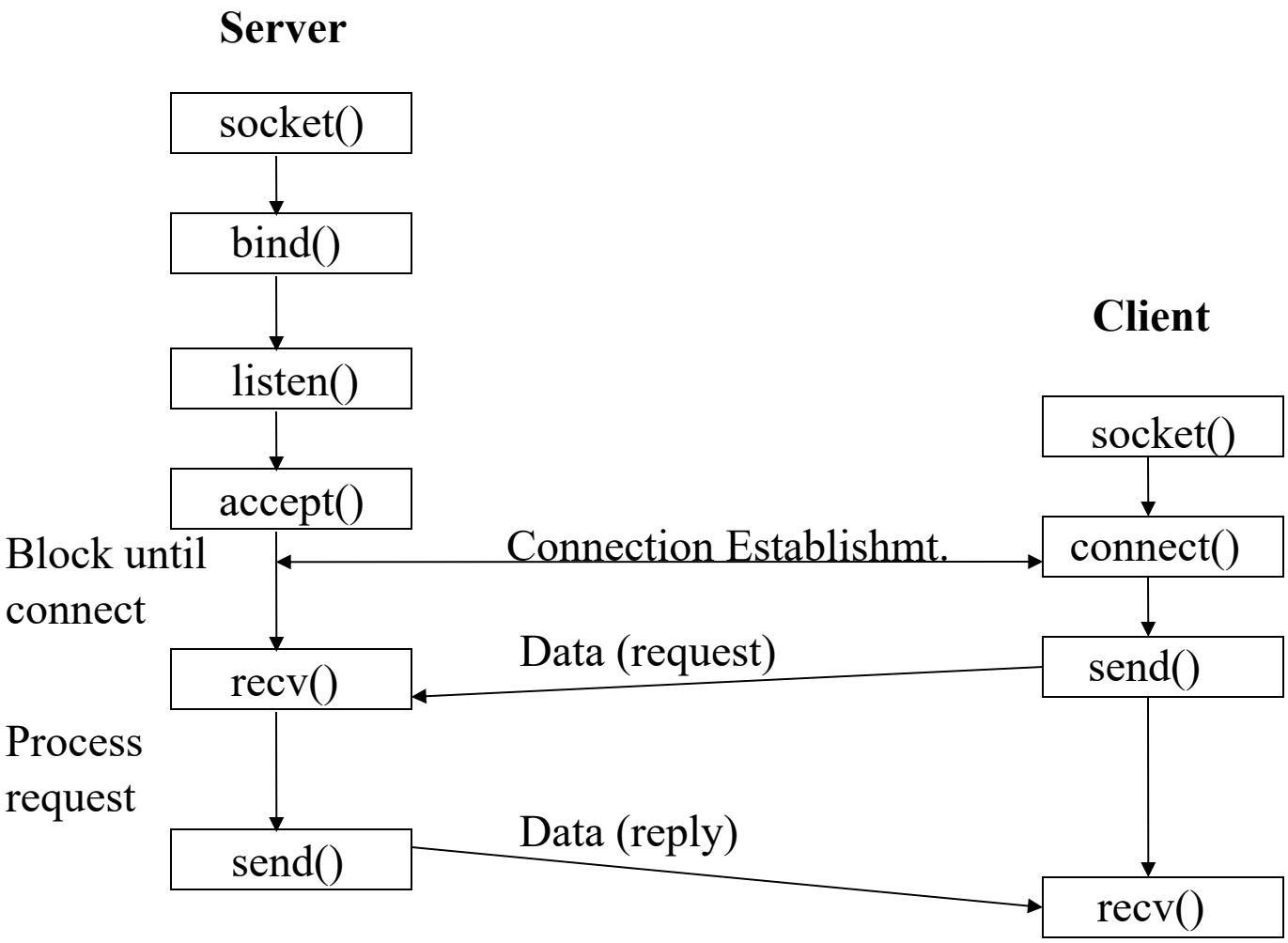
Socket: how to use

- Setup socket
  - Where is the remote machine (IP address, hostname)
  - What service gets the data (port)
- Send and Receive
  - Designed just like any other I/O in unix
  - send – write
  - recv -- read
- •Close the socket

Stream sockets (TCP)

- TCP provides connections between clients and servers
- TCP also provides reliability :
  - When TCP sends data to the other end, it requires an acknowledgment in return
- TCP provides flow control
  - TCP will ensure that a sender is not overwhelming a receiver by sending packets faster than it can consume
- TCP connection is full-duplex
  - Send and receive data over single connection

Stream sockets(TCP)



Stream Socket APIs

- socket()
  - creates a socket of a given domain, type, protocol (buy a phone)
  - Returns a file descriptor (called a socket ID)
- bind()
  - Assigns a name to the socket (get a telephone number)
  - Associate a socket with an IP address and port number (Eg : 192.168.1.1:80)
- connect()
  - Client requests a connection request to a server
  - This is the first of the client calls

Stream Socket APIs (cont)

- accept() :
  - Server accept an incoming connection on a listening socket (request from a client- answer phone)
  - There are basically three styles of using accept:
    - *Iterating server*: only one socket is opened at a time.
    - *Forking server*: after an accept, a child process is forked off to handle the connection.
    - *Concurrent single server*: simultaneously wait on all open socketIds, and waking up the process only when new data arrives

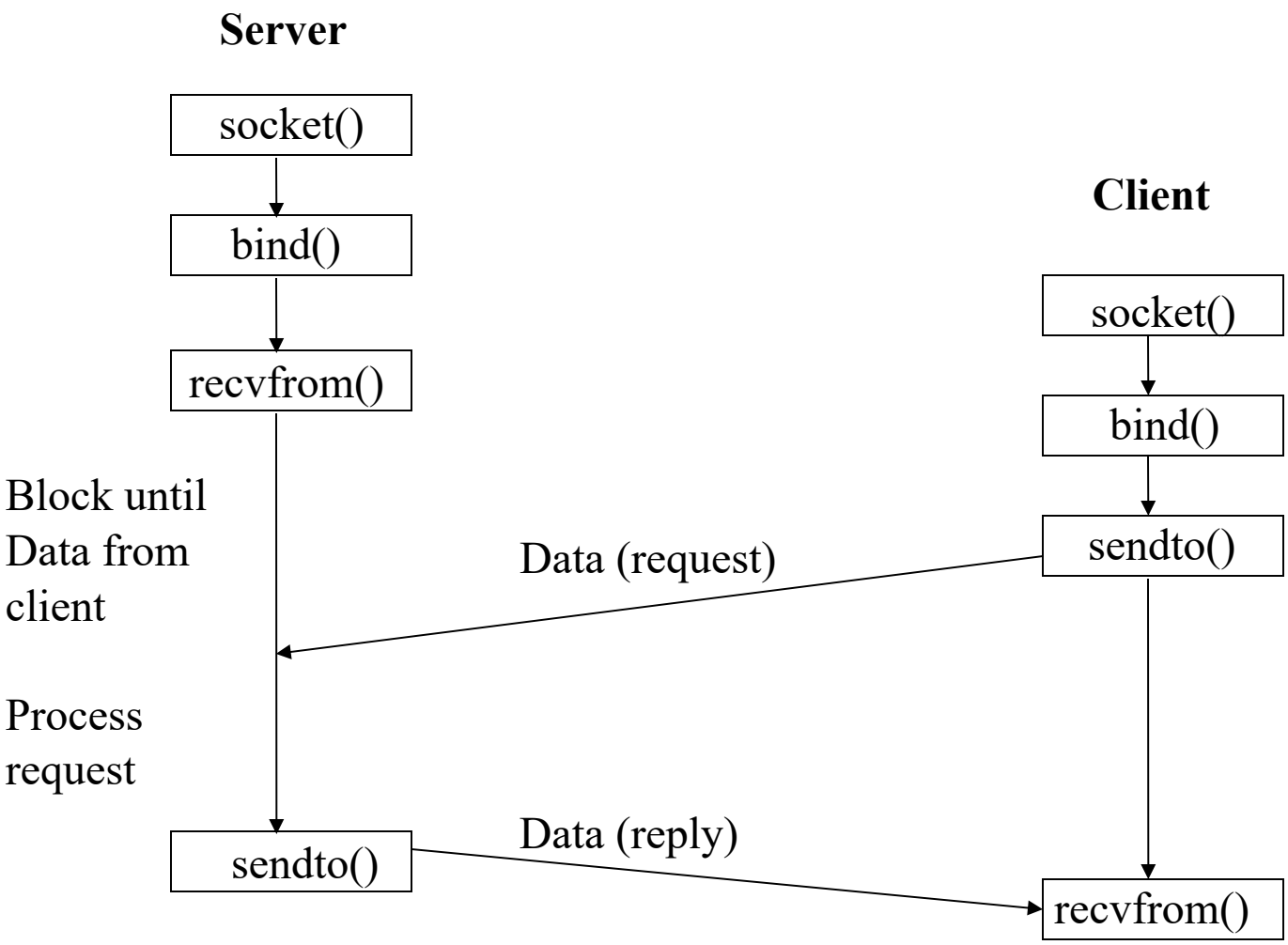
Stream Socket APIs (cont)

- listen()
  - Specifies the number of pending connections that can be queued for a server socket. (call waiting allowance)
- send()
  - Write to connection (speak)
  - Send a message
- recv()
  - read from connection (listen)
  - Receive data on a socket
- close()
  - close a socket (end the call)

Datagram Socket (UDP)

- UDP is a simple transport-layer protocol
- If a datagram is errored or lost, it won't be automatically retransmitted (can process in application)
- UDP provides a *connectionless* service, as there need not be any long-term relationship between a UDP client and server

Datagram Socket (UDP)



Socket programming in C

- <stdio.h>
  - input and output of basic C programs.
- <sys/types.h>
  - Contains definitions of data types used in system calls. These types are used in the next two include files.
- <sys/socket.h>
  - Includes definitions of structures needed for sockets.
- <netinet/in.h>
  - Contains constants and structures needed for internet domain addresses.

Socket Address Structures

- Most socket functions require a pointer to a socket address structure as an argument.
- Each supported protocol suite defines its own socket address structure.
- A Socket Address Structure is a structure which has information of a socket to create or connect with it
- Different types of socket address structures
  - IPv4
  - IPv6

IPv4 socket address structure

```
#include <netinet/in.h>
struct in_addr {
    in_addr_t s_addr;          // 32-bit IPv4 address
                               // network byte ordered
};

struct sockaddr_in {
    uint8_t sin_len;           // length of structure
    sa_family_t sin_family;    // AF_INET
    in_port_t sin_port;        // 16-bit TCP or UDP port number
                               // network byte ordered
    struct in_addr sin_addr;   // 32-bit IPv4 address
                               // network byte ordered
    char sin_zero[8];          // unused
};
```

IPv4 socket address structure

```
#include <netinet/in.h>
struct in_addr {
    in_addr_t s_addr;        /* 32-bit IPv4 address */
                             /* network byte ordered */
};

struct sockaddr_in {
  uint8_t sin_len;           /* length of structure */
  sa_family_t sin_family;    /* AF_INET */
  in_port_t sin_port;        /* 16-bit TCP or UDP port number */
                             /* network byte ordered */
  struct in_addr sin_addr;   /* 32-bit IPv4 address */
                             /* network byte ordered */
  char sin_zero[8];          /* unused */
};
```

- **in_addr_t** is equivalent to the type **uint32_t**
- **uint8_t, uint16_t, unint_32_t:** Integer type with a width of exactly 8, 16, 32 bits.

IPv6 socket address structure

```
#include <netinet/in.h>
struct in6_addr {
   uint8_t s6_addr[16];    // 128-bit IPv6 address
                           // network byte ordered
};

#define SIN6_LEN    // required for compile-time tests

struct sockaddr_in6 {
   uint8_t sin6_len;         // length of this struct
   sa_family_t sin6_family; // AF_INET6
   in_port_t sin6_port;     // transport layer port#
                            // network byte ordered
   uint32_t sin6_flowinfo;  // flow information, undefined
   struct in6_addr sin6_addr; // IPv6 address
                              // network byte ordered
   uint32_t sin6_scope_id;  // set of interfaces for a scope
};
```

IPv6 socket address structure

```
#include <netinet/in.h>
struct in6_addr {
uint8_t s6_addr[16];   /* 128-bit IPv6 address */
                       /* network byte ordered */ };

#define SIN6_LEN       /* required for compile-time tests */

struct sockaddr_in6 {
    uint8_t sin6_len;     /* length of this struct  */
    sa_family_t sin6_family; /* AF_INET6 */
    in_port_t sin6_port;          /* transport layer port# */
                          /* network byte ordered */
    uint32_t sin6_flowinfo;       /* flow information, undefined */
    struct in6_addr sin6_addr; /* IPv6 address */
                               /* network byte ordered */
     uint32_t sin6_scope_id;   /* set of interfaces for a scope */ };
```

IP Number translation

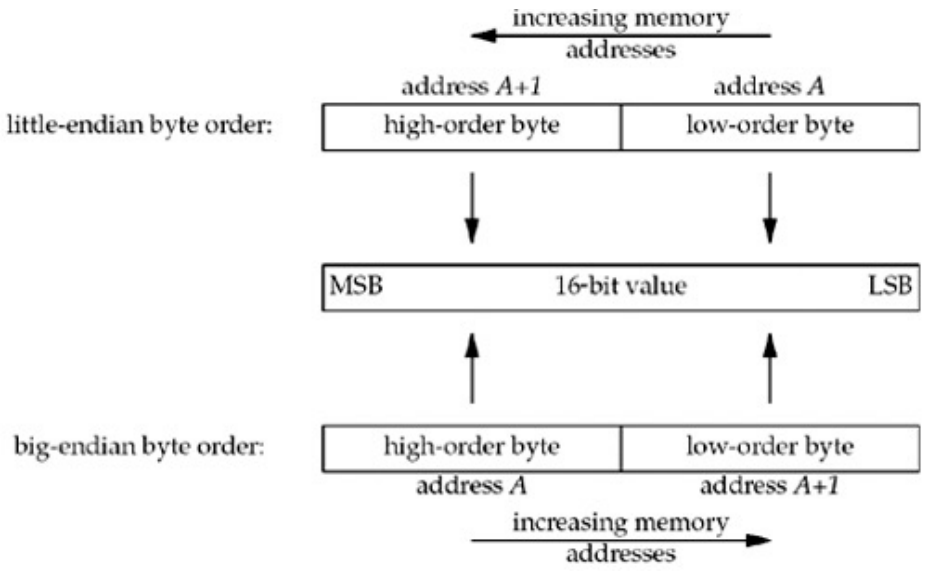
- IP address strings to 32 bit number
- Hence, these routines translate between the address as a string and the address as the number.
- Hence, we have 4 representations:
  - IP number in host order
  - IP number in network order
  - Presentation (eg. dotted decimal)
  - Fully qualified domain name

APIs for managing names and IP addresses

- Auxiliary API:
  - All binary values are network byte ordered
  - `htons(), htonl(), ntohs(), ntohl()`: ***byte ordering***
  - `inet_ntoa(), inet_aton()` : Convert IPv4 addresses from a dots-and-number string (eg : 192.168.1.1) to a struct in_addr and back
  - `inet_pton(), inet_ntop()`: conversion of IPv4 or IPv6 numbers between presentation and strings

Byte Ordering

- There are two ways to store the two bytes in memory
  - little-endian byte order
  - big-endian byte order



Byte Ordering (cont)

- There is no standard between these two byte-orderings
  - The Internet protocols use big-endian byte ordering
  - Host order can be big- or little-endian
    - X86: little-endian
    - SPARC: big-endian
- Conversion
  - *htons(), htonl(): host to network short/long*
  - *ntohs(), ntohl(): network order to host short/long*
- What need to be converted?
  - Address, port?
    - Because destination host reads address, TCP/UDP port number from IP,TCP packets sent from source

htons(), htonl(), ntohs(), ntohl()

- Convert multi-byte integer types from host byte order to network byte order

```
#include <netinet/in.h>
uint32_t htonl(u_long hostlong);  // host to network long
uint16_t htons(u_short hostshort);// host to network short
uint32_t ntohl(u_long netlong);   // network to host long
uint16_t ntohs(u_short netshort); // network to host short
```

- Each function returns the converted value.

inet_aton()

```
#include <arpa/inet.h>
int inet_aton(const char *cp, struct in_addr *inp)
```

- Convert IP addresses from a dots-and-number string to a struct `in_addr`
- Return:
  - The value non-zero if the address is valid
  - The value 0 if the address is invalid

```
struct in_addr someAddr;
if(inet_aton("10.0.0.1", &someAddr))
   printf("The address is valid");
else printf ("The address is invalid");
```

inet_ntoa()

```
#include <arpa/inet.h>
char *inet_ntoa(struct in_addr in);
```

- Convert IP addresses from a struct `in_addr` to a dots-and-number string
- Return: the dots-and-numbers string

```
struct in_addr someAddr;
if(inet_aton("10.0.0.1", someAddr))
   printf("The address is valid");
else printf ("The address is invalid");
char *addrStr;
addrStr = inet_ntoa(someAddr);
```

inet_addr()

```
#include <arpa/inet.h>
in_addr_t inet_addr(const char *cp);
```

- Convert IP addresses from a dots-and-number string to a struct `in_addr`
- Return:
  - The value -1 if there's an error
  - The address as an `in_addr_t`

```
struct in_addr someAddr;
someAddr.s_addr = inet_addr("10.0.0.1");
```

For IPv6?

- The inet_aton(), inet_addr(), inet_network(), inet_makeaddr(), inet_lnaof(), inet_netof(), and inet_ntoa() functions can only manipulate IPv4 addresse

inet_pton()

```
#include <arpa/inet.h>
int inet_pton(in family, const char *cp, void *addr)
```

- Convert IP addresses from a dots-and-number string to a struct `in_addr` or `in6_addr`
- `family` is `AF_INET` or `AF_INET6`
- Return:
  - The value non-zero if the address is valid
  - The value 0 if the address is invalid

inet_ntop()

```
#include <arpa/inet.h>
const char *inet_ntop(int family, const void *addr,
                                  char *cp, size_t len);
```

- Convert IP addresses from a struct `in_addr` to a dots-and-number string
- Return: the dots-and-numbers string

```
struct sockaddr_in sa;
char str[INET_ADDRSTRLEN];

// store this IP address in sa:
inet_pton(AF_INET, "192.0.2.33", &(sa.sin_addr));

// now get it back and print it
inet_ntop(AF_INET, &(sa.sin_addr), str, INET_ADDRSTRLEN);
printf("%s\n", str);
```

APIs for managing names and IP addresses

- `gethostname()`: Returns the name of the system
- `gethostbyname()` : returns information about a host specified by a host name.

ADDRESS RESOLUTION

Content

- IPv4 and IPv6
- DNS
- Address and Name APIs

IPv4

- Developed in APRANET (1960s)
- 32-bit number
- Divided into classes that describe the portion of the address assigned to the network (netID) and the portion assigned to endpoints (hosten)
  - A : netID – 8 bit
  - B : netID – 16 bit
  - C : netID – 24 bit
  - D : use for multicast
  - E : use for experiments

IPv4 problem

- IPv4 addresses is being exhausted
- Have to map multiple private addresses to a single public IP addresses (NATs)
  - Connect 2 PCs use private address space ?
  - NAT must be aware of the underlying protocols

→ Develope a new version of IP Address : IPv6

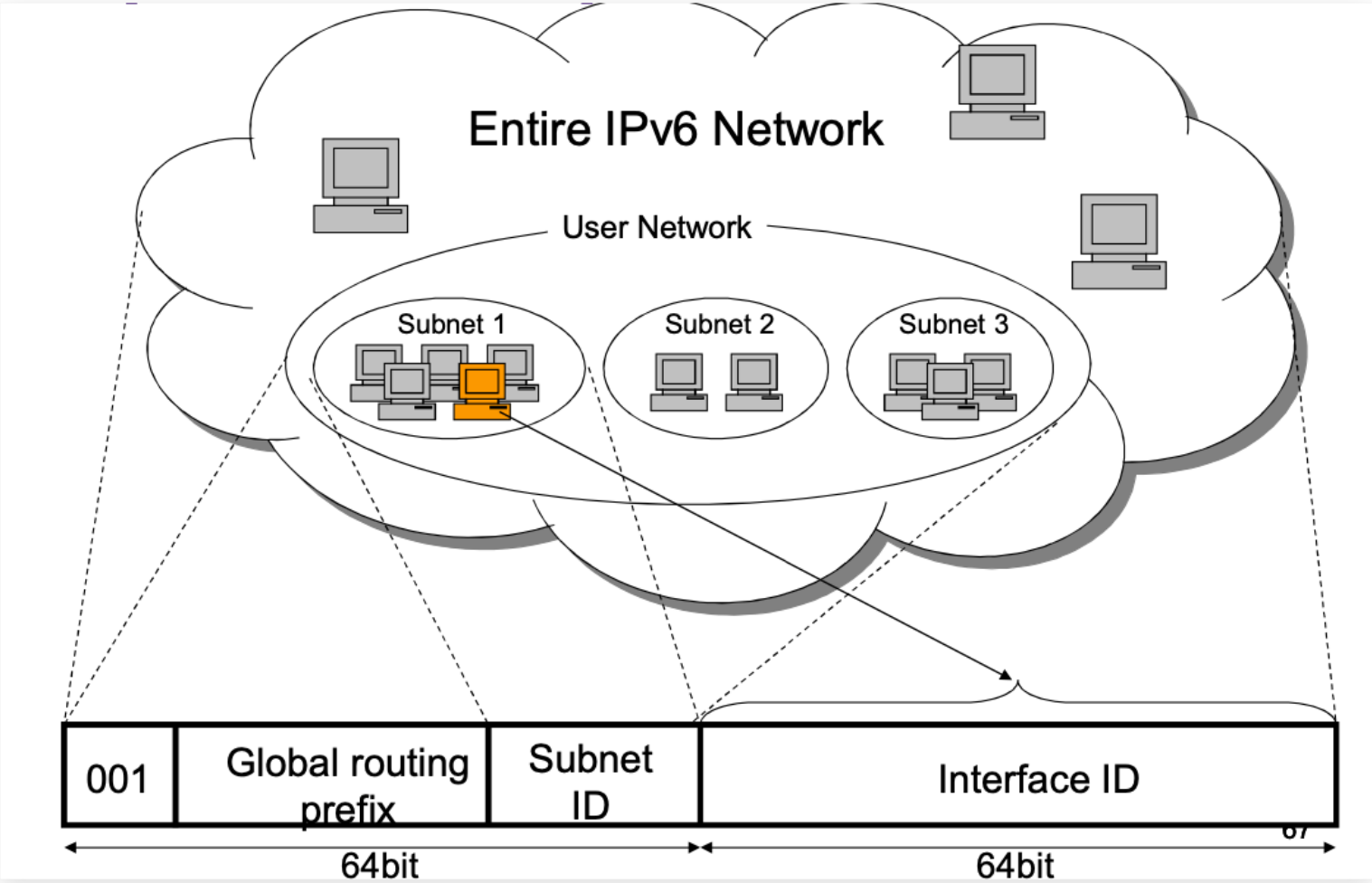
IPv6

- IPv6 address is 128 bits
  - To subdivide the available addresses into a hierarchy of routing domains that reflect the Internet's topology
- IPv6 address is typically expressed in 16-bit chunks displayed as hexadecimal numbers separated by colons

  Example : 21DA:00D3:0000:2F3B:02AA:00FF:FE28:9C5A

  or : 21DA:D3:0:2F3B:2AA:FF:FE28:9C5A

IPv6


Entire IPv6 Network
User Network
Subnet 1
Subnet 2
Subnet 3
001
Global routing prefix
Subnet ID
Interface ID
64bit
64bit

DNS (Domain Name System)

- Computers use IP Addresses to connect hosts
  - What about humans ? – IP Addresses are very complex and hard to remember (for people)
- Use name instead of IP Address → Domain Name System
- Problem of DNS
  - People use names, Computers use IP Addresses → translate between two spaces
  - Domain name system must be hierarchical (for management and maintain)
- Domain name space : divide to zones

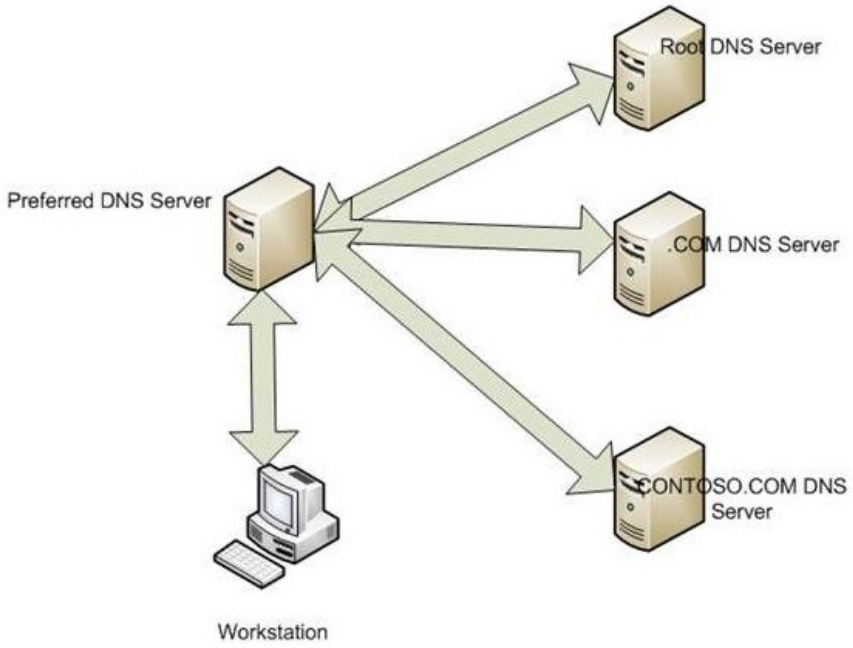
DNS (cont)

- How to translate between domain name-IP Address and reverse ?
  - DNS Resolver
  - DNS Server
- A DNS query
  - A *non-recursive query* : DNS server provides a record for a domain for which it is authoritative itself, or it provides a partial result without querying other servers
  - A *recursive query* : DNS server will fully answer the query by querying other name servers
- DNS primarily uses User Datagram Protocol (UDP) on port number 53 to serve requests

DNS (cont)

- Address resolution mechanism
  - Local system is pre-configured with the known addresses of the root server in a file of *root hints*
  - Query one of the root servers to find the server authoritative for the next level down
  - Querying level down server for the address of a DNS server with detailed knowledge of the lower level domain until reach the DNS

    Server return final address



DNS (cont)

- A *Resource Record* (RR) is the basic data element in the domain name system
- All records use the common format specified in RFC 1035 (in IP networks)
- **RR (Resource record) fields**
  - NAME (variable)
    - Name of the node to which this record pertains.
  - TYPE (2)
    - Type of RR. For example, MX is type 15
  - CLASS (2)
    - Class code
  - TTL (4)
    - Unsigned time in seconds that RR stays valid
  - RDLENGTH (2)
    - Length of RDATA field
  - RDATA (variable)
    - Additional RR-specific data

List of Address and Name APIs

#include <sys/socket.h>

- **gethostbyaddr()**
  - Retrieve the name(s) and address corresponding to a network address.
- **gethostname()**
  - Retrieve the name of the local host.
- **gethostbyname()**
  - Retrieve the name(s) and address corresponding to a host name.
- **getprotobyname()**
  - Retrieve the protocol name and number corresponding to a protocol name.
- **getprotobynumber()**
  - Retrieve the protocol name and number corresponding to a protocol number.
- **getservbyname()**
  - Retrieve the service name and port corresponding to a service name.
- **getservbyport()**
  - Retrieve the service name and port corresponding to a port.

New APIs for IPv6

- Those APIs only supports IPv4 but IPv6 will be replace IPv4 in the future, so we need APIs support IPv6
- They are
  - getaddrinfo
  - getnameinfo
- These APIs have replaced the IPv4 specific routines

gethostbyaddr()

```
#include <netdb.h>
#include <sys/socket.h>
struct hostent *gethostbyaddr (in_addr *addr, socklen_t len,
                               int family);
```

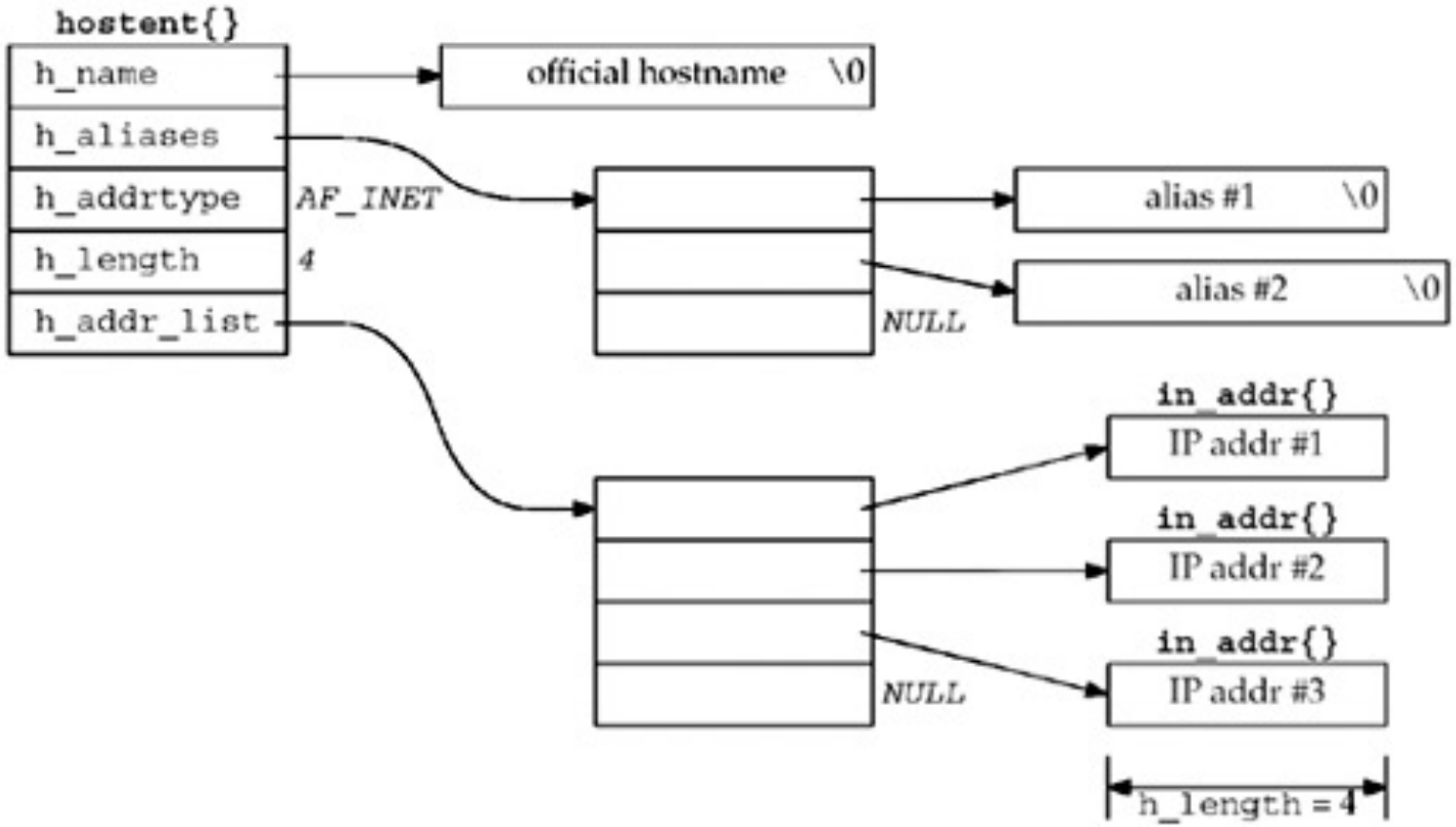
- Get host information corresponding to an address.
- Parameters:
  - [IN] `addr`: A pointer to an address in network byte order.
  - [IN] `len`: The length of the address, which must be 4 for AF_INET addresses.
  - [IN] `family`: The type of the address, which must be AF_INET.
- Return value
  - If no error occurs, returns a pointer to the `hostent` structure
  - Otherwise it returns a NULL pointer and a specific error number

struct hostent

```
struct hostent {
      char   *h_name;        // official (canonical) name of host
      char   **h_aliases;    // pointer to array of pointers to
                                 alias names
      int      h_addrtype;   // host address type: AF_INET
      int      h_length;     // length of address: 4
      char   **h_addr_list;  // ptr to array of ptrs with IPv4 addrs
};
```

- What is this struct `hostent` that gets returned?
- It has a number of fields that contain information about the host in question.

struct hostent



gethostname()

```
#include <sys/unistd.h>
#include <sys/socket.h>
int gethostname(char *name, size_t len);
```

- Return the standard host name for the local machine.
- Parameters:
  - [OUT] `name`: points to a buffer that will receive the host name.
  - [IN] `len`: the length of the buffer
- Return value
  - If no error occurs, returns 0
  - Otherwise it returns SOCKET_ERROR and a specific error code

gethostbyname()

```
#include <netdb.h>
#include <sys/socket.h>
struct hostent *gethostbyname (const char *hostname);
```

- Get host information corresponding to a hostname.
- [IN] `name`: Points to the name of the host
- Returns a pointer to a `hostent` structure
- Return value
  - If no error occurs, returns a pointer to the `hostent` structure described above.
  - Otherwise it returns a NULL pointer and a specific error number

getservbyname()

```
#include <netdb.h>
#include <sys/socket.h>
struct servent *getservbyname (const char *servname,
                               const char *protoname);
```

- Get service information corresponding to a service name and protocol.
- Parameters:
  - [IN] `servname`: A pointer to a service name.
  - [IN] `protoname`: An optional pointer to a protocol name.
    - If this is NULL, getservbyname() returns the first service entry for which the name matches the `s_name` or one of the `s_aliases`.
    - Otherwise getservbyname() matches both the name and the proto.
- Returns
  - non-null pointer if OK
  - NULL on error

```
struct servent *sptr;
sptr = getservbyname("ftp", "tcp");
```

struct servent

```
struct servent {
        char *s_name;
        char **s_aliases;
        int s_port;
        char *s_proto;
};
```

- s_name
  - Official name of the service.
- s_aliases
  - A NULL-terminated array of alternate names.
- s_port
  - The port number at which the service may be contacted. Port numbers are returned in network byte order.
- s_proto
  - The name of the protocol to use when contacting the service.

getservbyport()

```
#include <netdb.h>
#include <sys/socket.h>
struct servent *getservbyport (int port, const char *protoname);
```

- Get service information corresponding to a port and protocol.
- Parameters:
  - [IN] `port`: The port for a service, in network byte order.
  - [IN] `protoname`: An optional pointer to a protocol name.
    - If this is NULL, returns the first service entry for which the port matches the `s_port`.
    - Otherwise getservbyport() matches both the port and the proto.
- Return
  - non-null pointer if OK
  - NULL on error

```
struct servent *sptr;
sptr = getservbyport (htons (53), "udp");
```

getpeername ()

```
#include <sys/socket.h>
int getpeername(int sockfd, struct sockaddr *addr,
                                        socklen_t *addr_len);
```

- Retrieve the address associated with the remote socket
- Parameters:
  - [IN] `sockfd`: the local socket connecting to remote socket
  - [OUT] `addr`*:* points to the `sockaddr` struct
  - [IN, OUT] `addr_len`: points to the `socklen_t` value initiated to indicate the amount of space pointed to by `addr`.
- Return:
  - On success, returns 0
  - On error, return -1 and `errno` set to indicate the error