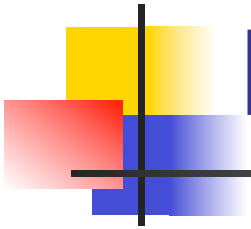
Network Socket Programming - 2

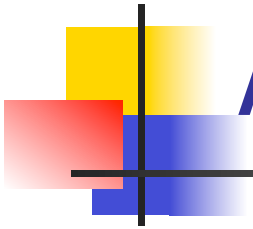
BUPT/QMUL

2019-3-18



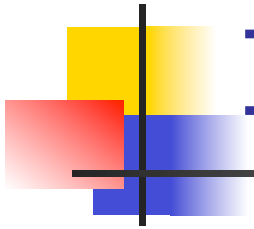
Review

- Basic Concepts in NP
  - Introduction to Network Programming
    - Importance
    - Classes
    - Environments in this course
  - Program Developing
    - Phases
    - Skills
    - Useful tools
  - Basic Concepts
    - Process
    - File Descriptor
    - System Call
    - Signal



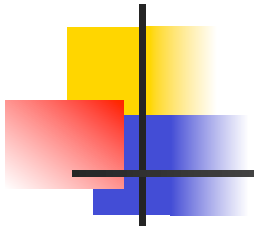
Agenda

- Basic concepts in NP
- Introduction to IP & TCP/UDP
- *Introduction to Sockets*



Introduction to Sockets

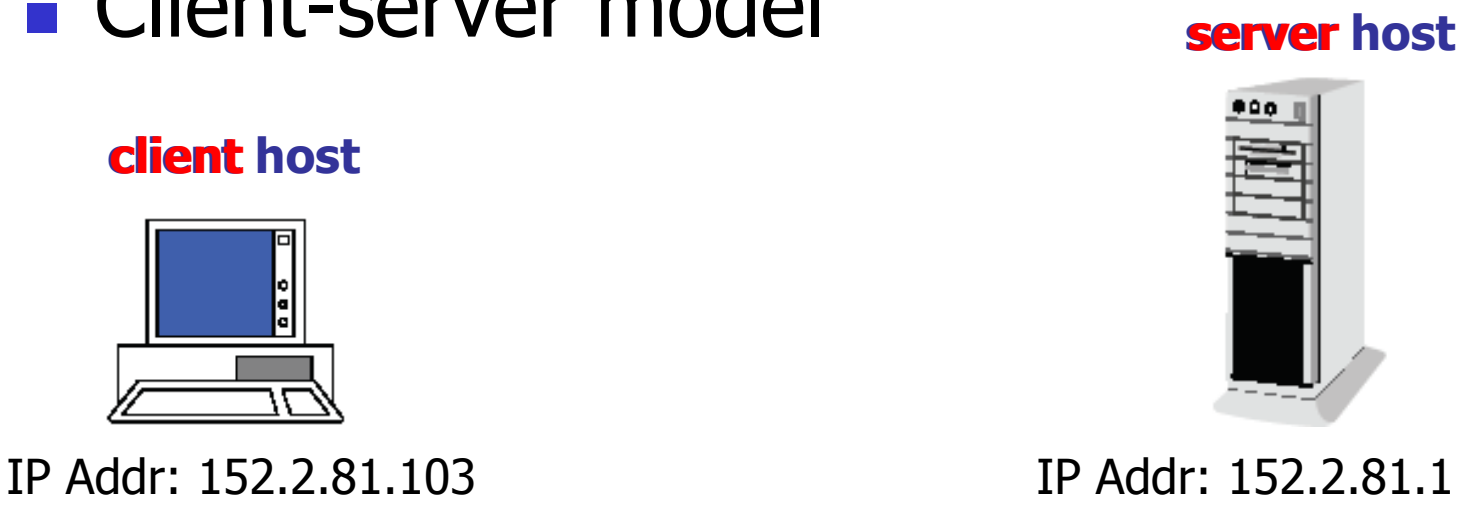
- Reviews of some helpful points
- Sockets interface
- Major system calls
- Sample programs



Introduction to Sockets
Part I: some helpful points

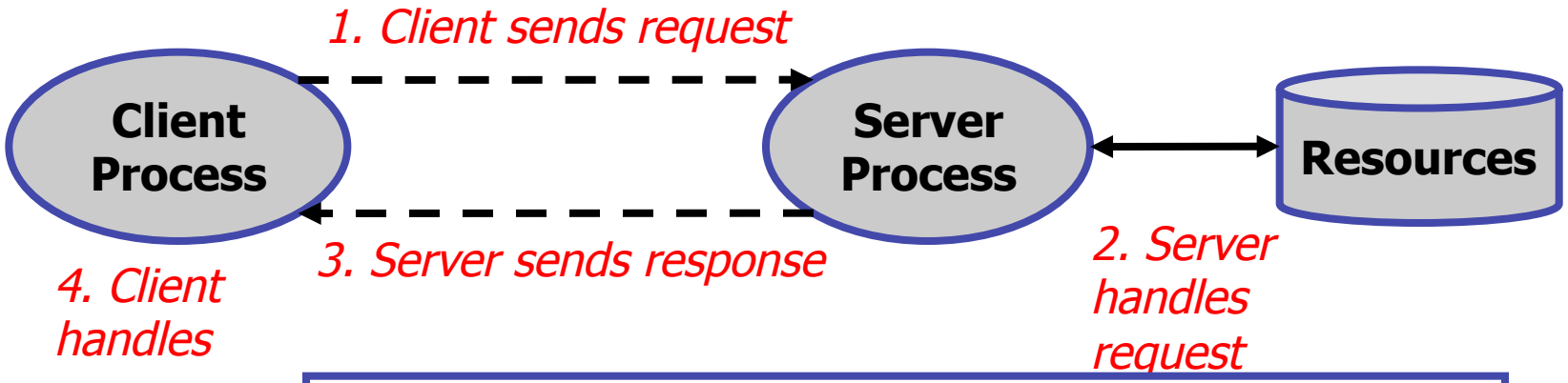
Reviews Of Some Helpful Points

- Client-server model



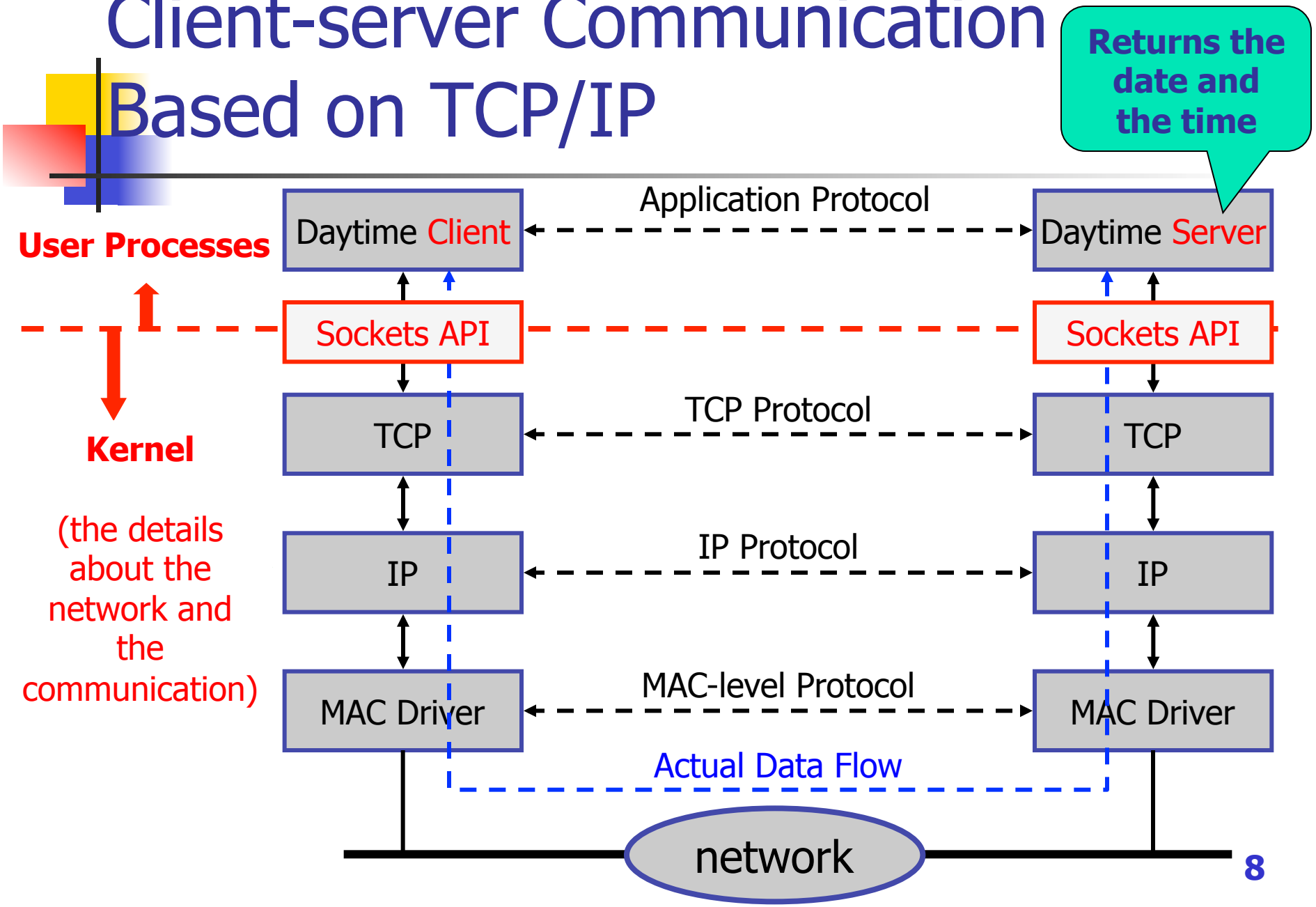
A Client-server Transaction

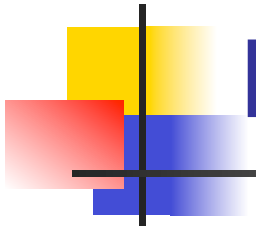
- Most of network applications are based on the client-server model:
  - A ***server*** process and one or more ***client*** processes
  - Server manages some ***resources***.
  - Server provides ***service*** by manipulating resources for clients.



Note: *clients and servers are processes running on hosts (can be the same or different hosts).*

Client-server Communication Based on TCP/IP





Reviews Of Some Helpful Points

- A programmer's view of the Internet
  - 1. Hosts are mapped to a set of 32-bit *IP addresses*.
    - 202.112.96.163
  - 2. The set of IP addresses is mapped to a set of identifiers called Internet *domain names*.
    - 202.112.96.163 is mapped to www.bupt.edu.cn
  - 3. A process on one Internet host can communicate with a process on another Internet host over a *connection*.

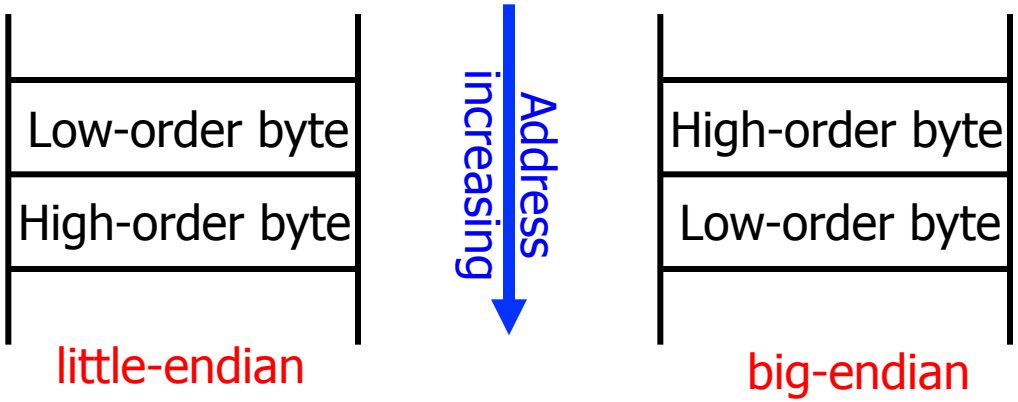
IP Addresses (1)

- 32-bit IP addresses are stored in an *IP Address structure*
  - <netinet/in.h>

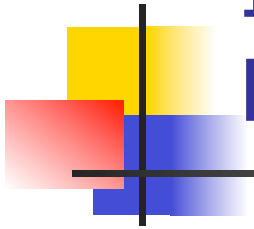
```
/* Internet address.  */
typedef uint32_t in_addr_t;
struct in_addr {
    in_addr_t s_addr;
  };
```

```
/*Defined in <stdint.h>*/
typedef unsigned int uint32_t;
```

- Two ways to store multi-byte integers
  - Big-endian vs. little-endian



IP Addresses (2): Host byte order vs. network byte order



- Host byte order is machine-dependent
  - You can see it in <bits/endian.h>
  - A program used to output the host byte order
- Network byte order is machine-independent (*big-endian*)
- Byte order conversion functions
  - **htonl**: host byte order → network byte order for ***long int***
  - **htons**: host byte order → network byte order for ***short int***
  - **ntohl**: network byte order → host byte order for ***long int***
  - **ntohs**: network byte order → host byte order for ***short int***

IP Addresses (3)

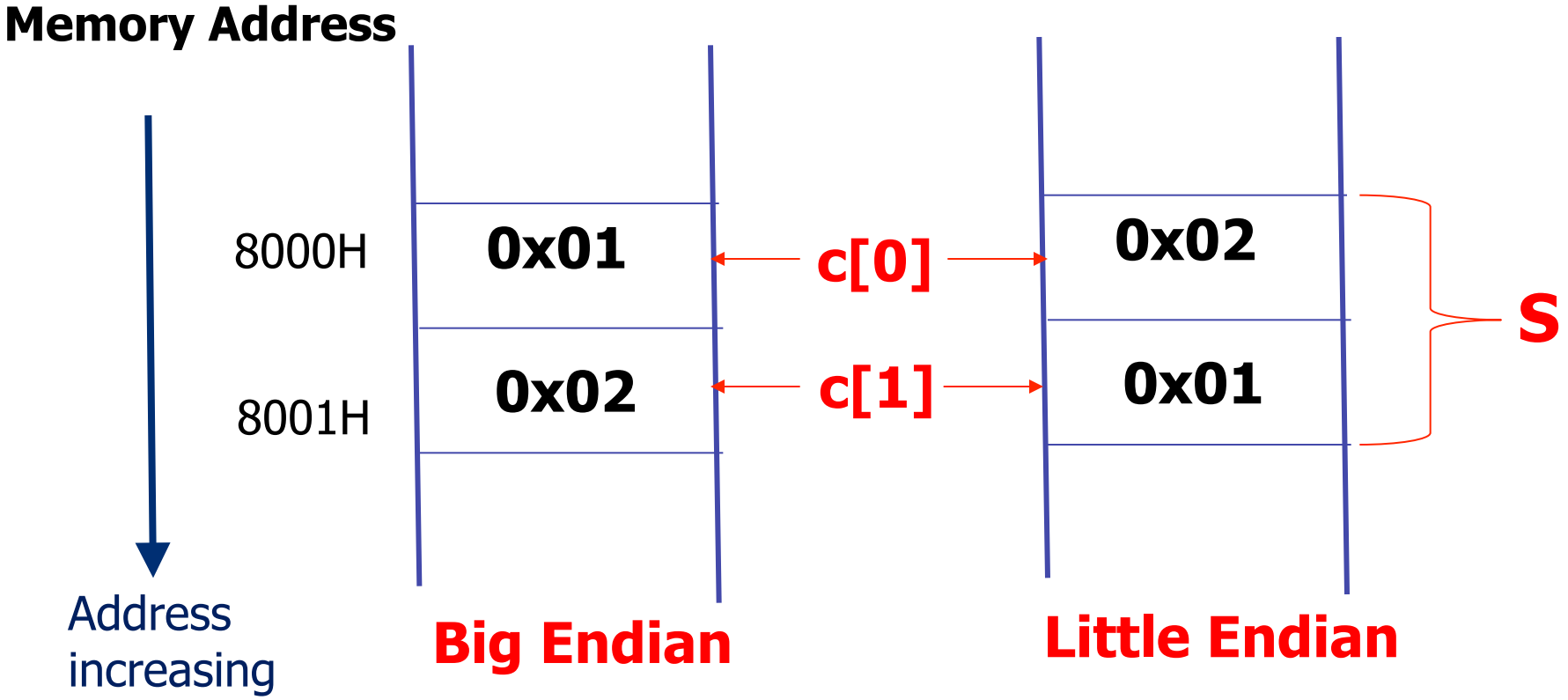
-- A program used to output the host byte order

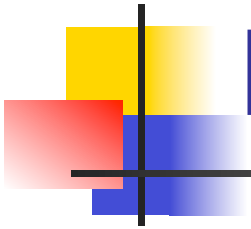
- byteorder.c

```
#include <sys/types.h>
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
        union {
                short s;
                char c[sizeof(short)];
        }un;

        un.s=0x0102;

        if (sizeof(short)==2){
                if (un.c[0]==1 &&un.c[1]==2)
                        printf("big-endian\n");
                else if  (un.c[0]==2 &&un.c[1]==1)
                        printf("littlt-endian\n");
                else
                        printf("Unknow\n");
        }else
                printf("sizeof(short)=%d\n",sizeof(short));

        exit(0);
}
```

a **union** is like a **struct**, only all the data members sit at the same memory location. This means only one of them can be used at a time.

Storage example of Variable *s* and *c*





Domain Name System (1)

- The Internet maintains a mapping between IP addresses and domain names in a huge worldwide distributed database called *DNS*.
  - Conceptually, programmers can view the DNS database as a collection of millions of *host entry structures*
  - <netdb.h>

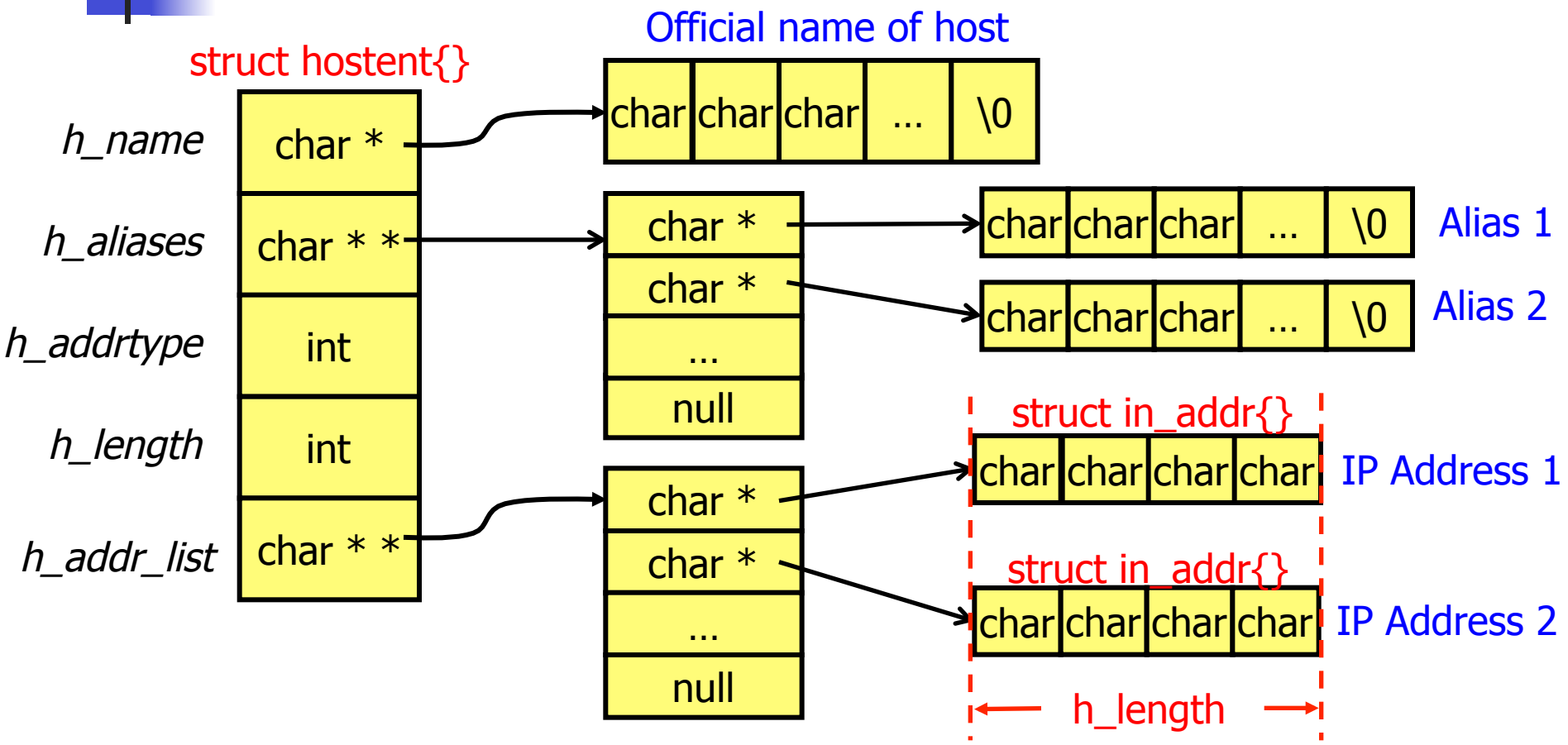
Domain Name System (2): Host Entry Structure

h_addr_list:

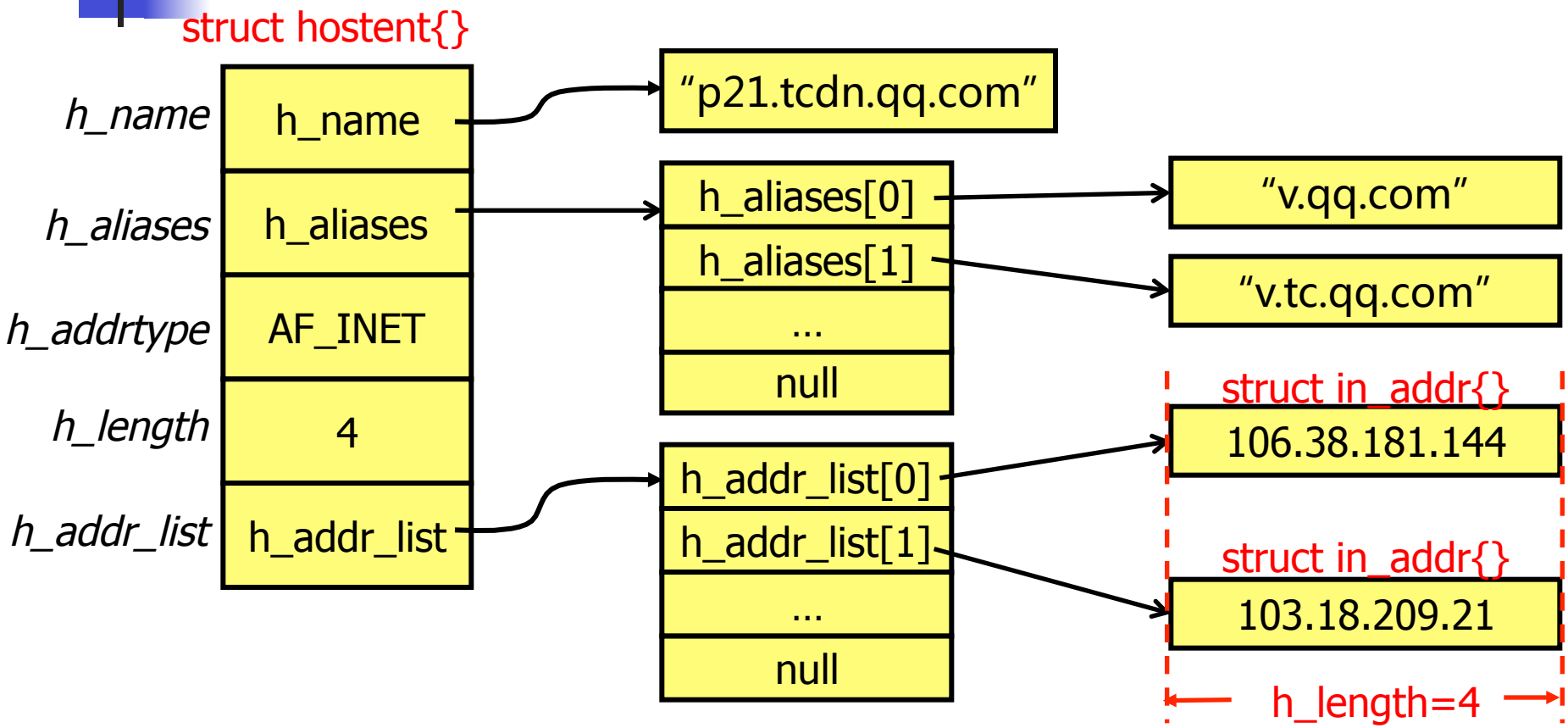
An array of pointers to IP addresses for the host (in network byte order), terminated by a NULL pointer.

```
/* Description of data base entry for a single host. */
struct hostent
{
  char *h_name;             /* Official name of host.  */
  char **h_aliases;         /* Alias list.  */
  int h_addrtype;           /* Host address type.  */
  int h_length;             /* Length of address.  */
  char **h_addr_list;     /* List of addresses from name
                             server.*/
#define h_addr  h_addr_list[0]  /* The first address in
                                   the address list.  */
};
```

Domain Name System (3): Host Entry Structure



Domain Name System (4): Example of Host Entry Structure



You can try the command: *nslookup v.qq.com*

Domain Name System (5)

- Functions for retrieving host entries from DNS
  - gethostbyname: query key is a DNS domain name.

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

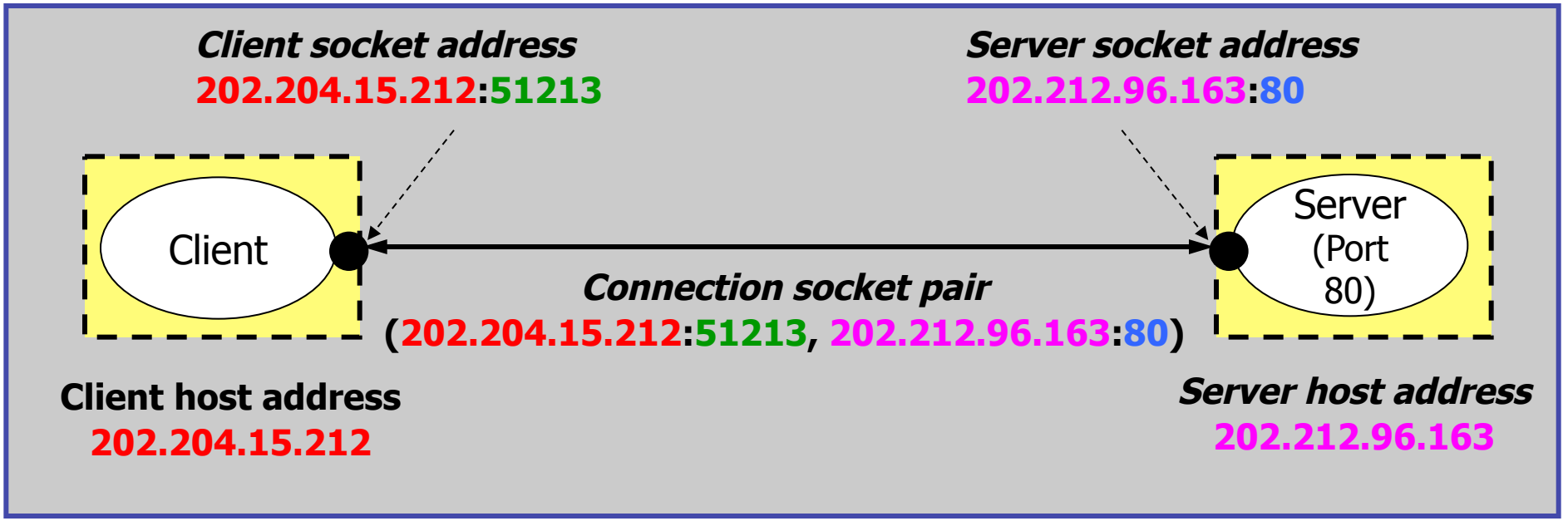
  - gethostbyaddr: query key is an IP address.

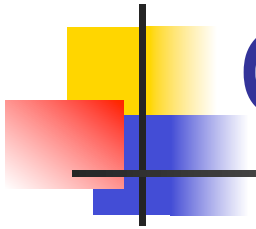
```
#include <netdb.h>
struct hostent * gethostbyaddr (const char *addr, int len, int family );
```

AF_INET for IPv4

Connections

- Clients and servers communicate by sending streams of bytes over *connections*.
- Connections are end-to-end, full-duplex (2- way communication), and reliable.

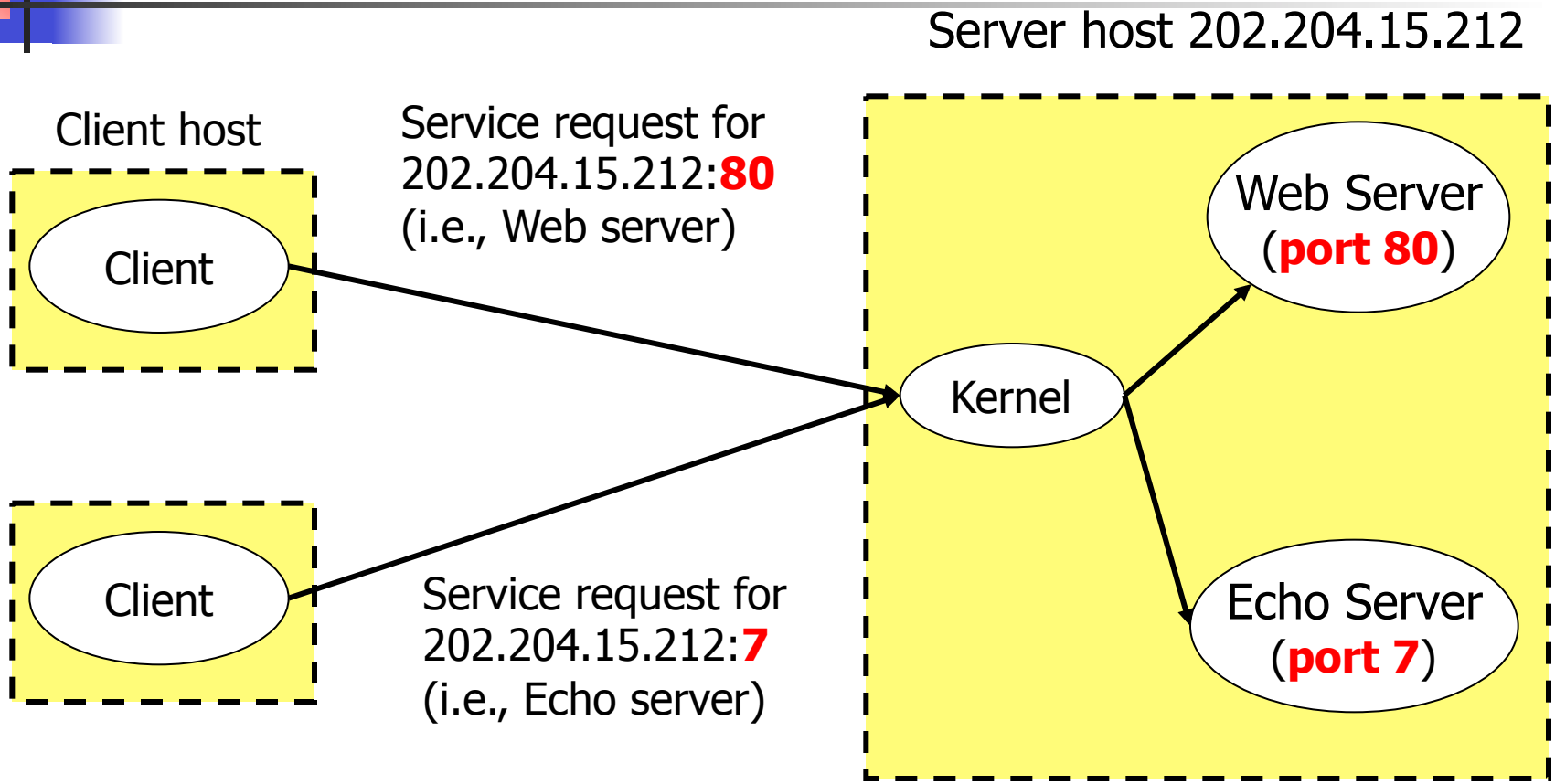


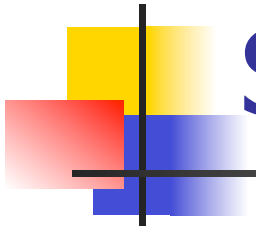


Clients

- Examples of client programs
  - Web browsers, ftp, telnet, ssh
- How does a client find the server?
  - The IP address in the server socket address identifies the host *(more precisely, an adapter on the host)*
  - The (well-known) port in the server socket address identifies the service, and thus implicitly identifies the server process that performs that service.
  - Examples of well-known ports
    - Port 7: Echo server
    - Port 23: Telnet server
    - Port 25: Mail server
    - Port 80: Web server

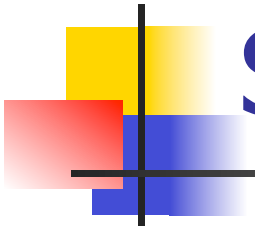
Using ports to identify services





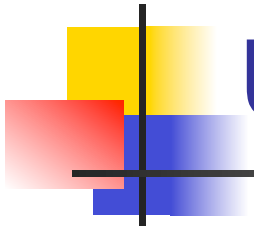
Servers

- Servers are long-running processes (daemons).
  - Typically created at boot-time by the **init process** (pid=1)
  - Run continuously until the machine is turned off
- Each server waits for requests to arrive on a well-known port associated with a particular service.
  - See ***/etc/services*** for a comprehensive list of the services available on a Linux machine
- A machine that runs a server process is also often referred to as a "server"



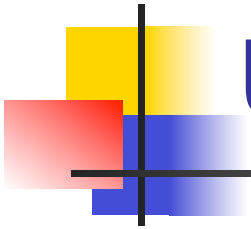
Server examples

Name	Port	Services	Resources
Web server	80	Retrieves files and runs CGI programs on behalf of the client	files/compute cycles (CGI programs)
FTP server	20, 21	stores and retrieve files	files
TELNET server	23	proxies a terminal on the server machine	terminal
Mail server	25	stores mail messages in spool file	email “spool” file



Useful Unix Commands

- netstat
- ifconfig
- ping



Useful Unix Commands - netstat

- Functions: prints information about the Linux networking subsystem, e.g., network connections, routing tables, interface statistics etc.
- netstat
  - Displays a list of open sockets.
- netstat -i
  - Display the information about the network interfaces
- netstat -ni
  - Display the information about the network interfaces using numeric addresses
- netstat -r
  - Display the kernel routing tables
- netstat -nr
  - Display the kernel routing tables using numeric addresses

Useful Unix Commands - netstat

- netstat

```
[root@localhost include]# netstat
Active Internet connections (w/o servers)
Proto Recv-Q Send-Q Local Address               Foreign Address             State
tcp        0      0 192.168.1.253:telnet       192.168.1.27:3256           ESTABLISHED
tcp        0      0 ::ffff:192.168.1.253:ssh   ::ffff:192.168.1.27:2888    ESTABLISHED
tcp        0      0 ::ffff:192.168.1.253:ssh   ::ffff:192.168.1.27:3047    ESTABLISHED
Active UNIX domain sockets (w/o servers)
Proto RefCnt Flags       Type       State         I-Node Path
unix  10     [ ]         DGRAM                    5724   /dev/log
unix  2      [ ]         DGRAM                    6859   @/var/run/hal/hotplug_socket
unix  2      [ ]         DGRAM                    3351   @udevd
unix  2      [ ]         DGRAM                    927082
unix  2      [ ]         DGRAM                    926850
unix  3      [ ]         STREAM     CONNECTED     924266
unix  3      [ ]         STREAM     CONNECTED     924265
unix  3      [ ]         STREAM     CONNECTED     916866 /tmp/.X11-unix/X16
```

Useful Unix Commands - netstat

- netstat –ni

```
[root@localhost /]# netstat -ni
Kernel Interface table
Iface       MTU Met    RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
eth0       1500   0 49922499      0      0      0 20779220      0      0      0 BMRU
lo        16436   0     1934      0      0      0     1934      0      0      0 LRU
```

- Ethernet interface is called eth0 or le0 depending on the machine
- Loop back interface is called lo and the common used IP address is 127.0.0.1

Useful Unix Commands - netstat

- netstat -nr

```
[root@localhost /]# netstat -nr
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
192.168.1.0     0.0.0.0         255.255.255.0   U         0 0          0 eth0
169.254.0.0     0.0.0.0         255.255.0.0     U         0 0          0 eth0
0.0.0.0         192.168.1.1     0.0.0.0         UG        0 0          0 eth0
```

Useful Unix Commands - ifconfig

- Functions: configure the network interfaces, and usually be used to print the configuration of the network interfaces

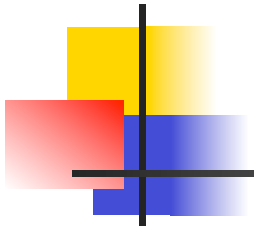
```
[root@localhost /]# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:13:72:4F:9D:3A
          inet addr:192.168.1.253  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::213:72ff:fe4f:9d3a/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:49923781 errors:0 dropped:0 overruns:0 frame:0
          TX packets:20779648 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:647355456 (617.3 MiB)  TX bytes:2713364 (2.5 MiB)
          Base address:0xecc0 Memory:fe6e0000-fe700000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:1934 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1934 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:266858 (260.6 KiB)  TX bytes:266858 (260.6 KiB)
```

Useful Unix Commands - ping

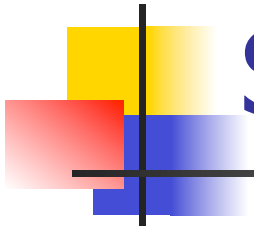
- Functions: Sends a packet to the host specified by destination and prints out the roundtrip time (Uses ICMP messages)

```
[root@localhost etc]# ping 192.168.1.27
PING 192.168.1.27 (192.168.1.27) 56(84) bytes of data.
64 bytes from 192.168.1.27: icmp_seq=0 ttl=128 time=0.261 ms
64 bytes from 192.168.1.27: icmp_seq=1 ttl=128 time=0.219 ms
64 bytes from 192.168.1.27: icmp_seq=2 ttl=128 time=0.181 ms

--- 192.168.1.27 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.181/0.220/0.261/0.034 ms, pipe 2
```

```
[root@localhost etc]# ping www.baidu.com
PING www.a.shifen.com (202.108.22.5) 56(84) bytes of data.
64 bytes from xd-22-5-a8.bta.net.cn (202.108.22.5): icmp_seq=0 ttl=57 time=363 ms
64 bytes from xd-22-5-a8.bta.net.cn (202.108.22.5): icmp_seq=1 ttl=57 time=177 ms
64 bytes from xd-22-5-a8.bta.net.cn (202.108.22.5): icmp_seq=2 ttl=57 time=172 ms

--- www.a.shifen.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 172.446/237.748/363.698/89.081 ms, pipe 2
```

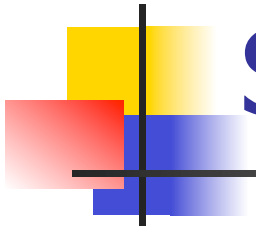


Introduction to Sockets
Part II: sockets interface



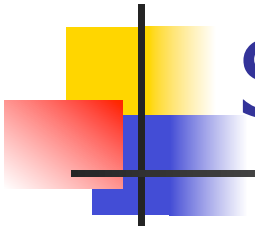
Sockets Interface

- Functions
- Definitions
- Types



Sockets Interface – functions

- Created in the early 80's as part of the original Berkeley distribution of Unix that contained an early version of the Internet protocols
- Provides a user-level interface to the network
- Underlying basis for all Internet applications
- Based on client/server programming model



Sockets Interface – definitions(1)

- What is a socket?
  - To the kernel, a socket is an endpoint of communication.
  - To an application, a socket is a **file descriptor** that lets the application read/write from/to the network.
    - Remember: All Unix I/O devices, including networks, are modeled as files.
- Clients and servers communicate with each other by reading from and writing to socket descriptors.
- The main distinction between regular file I/O and socket I/O is how the application "opens" the socket descriptors.

File Model in Unix/Linux

- In Unix/Linux, all I/O devices are treated as files
  - Identified with File Descriptors
  - File operations
    - open
    - close
    - lseek
    - read
    - write
    - …

Sample File Descriptor Table (One per Process)

0	stdin
1	stdout
2	stderr
3	file
4	device
5	socket
..	…

Sockets Interface – definitions(2)

- **Internet-specific socket address** (bits/socket.h)

```
struct sockaddr_in {
   unsigned short sin_family; /* address family (always AF_INET) */
   unsigned short sin_port; /* port num in network byte order */
   struct in_addr sin_addr; /* IP addr in network byte order */
   unsigned char sin_zero[8]; /* pad to sizeof(struct sockaddr) */
};
```

  - **Address family**: Domains refer to the area where the communicating processes exist. Commonly used domains include:
    - AF_UNIX: for communication between processes on one system;
    - AF_INET (IPv4): for communication between processes on the same or different systems using the DARPA standard protocols (IP/ UDP/TCP)
    - AF_INET6 (IPv6)
    - AF_LOCAL (Unix domain)
    - AF_UNSPEC (the importance will be explained later)
    - …

Sockets Interface – definitions(3)

- **Generic socket address** (<sys/socket.h>)

```
struct sockaddr {
   unsigned short sa_family; /* protocol family */
   char sa_data[14]; /* protocol-specific address,
                        up to 14 bytes. */
};
```

  - **Protocol family**
    - PF_LOCAL: Local to host, pipes and file-domain
    - PF_UNIX: Old BSD name for PF_LOCAL
    - **PF_INET: IP protocol family**
    - PF_AX25: Amateur radio AX.25
    - PF_IPX: Novell internet protocol
    - PF_INET6: IP version 6
    - PF_ATMSVC: ATM SVCs
    - PF_APPLETALK: Appletalk DDP
    - …

Sockets Interface – definitions(4)

- Generic socket address and Internet-specific socket address
  - Pointer to generic socket address is used for address arguments to **connect(), bind() and accept()**
  - Must cast Internet-specific socket address (`sockaddr_in *`) to generic socket address (`sockaddr *`) for connect, bind, and accept

```
struct sockaddr_in serv;
/* fill in serv{}*/
bind (sockfd, (struct sockaddr *)&serv , sizeof(serv));
```

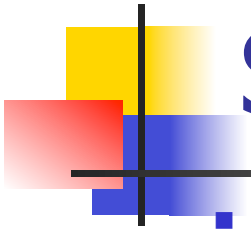
Socket Address

Generic socket address

```
struct sockaddr {
    unsigned short sa_family;      /* PF_INET for IPv4 */
    char sa_data[14];              /* protocol-specific address,
                                      up to 14 bytes. */
};
```

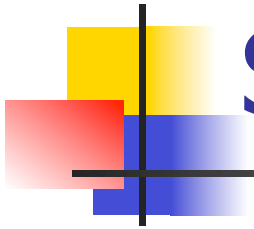
Internet-specific socket address

```
struct sockaddr_in{
    unsigned short  sin_family;   /* AF_INET */
    unsigned short  sin_port;     /* 16-bit port number */
                                  /* Network Byte Order*/
    struct in_addr sin_addr;      /* 32-bit IP Address */
                                  /* Network Byte Order */
    char            sin_zero[8];  /* unused */
}
```



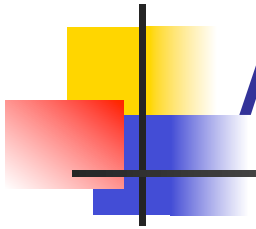
Sockets Interface – types(1)

- Stream Socket
  - Service: reliable (i.e. sequenced, non-duplicated, non-corrupted) bidirectional delivery of byte-stream data
  - Metaphor: a phone call
  - int s = socket (PF_INET, SOCK_STREAM, 0);
- Datagram Socket
  - Service: unreliable, unsequenced datagram
  - Metaphor: sending a letter
  - int s = socket (PF_INET, SOCK_DGRAM, 0);
- Raw Sockets Service
  - allows user-defined protocols that interface with IP
  - Requires `root` access
  - Metaphor: playing with an erector set
  - int s = socket (PF_INET, SOCK_RAW, protocol);
- **SOCK_STREAM and SOCK_DGRAM are the most common types of sockets used within UNIX/Linux**



Sockets Interface – types(2)

- Reliably-delivered Message Socket
  - Service: reliable datagram
  - Metaphor: sending a registered letter
  - Similar to datagram socket but ensure the arrival of the datagrams
  - int s = socket (PF_NS, SOCK_RDM, 0);
- Sequenced Packet Stream Socket
  - Service: reliable, bi-directional delivery of recordoriented data
  - Metaphor: record-oriented TCP
  - Similar to stream socket but using fixed-size datagrams
  - int s = socket (PF_NS, SOCK_SEQPACKET, 0);



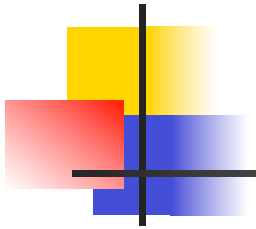
Abbreviations

API	Application Programming Interface
CGI	Common Gateway Interface
DNS	Domain Name System
FTP	File Transfer Protocol
ICMP	Internet Control Message Protocol
IP	Internet Protocol
MAC	Media Access Control
TCP	Transport Control Protocol
UDP	User Datagram Protocol

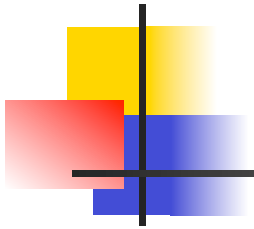
struct

- A complex data type declaration that defines a physically grouped list of variables to be placed under one name in a block of memory
- Allowing the different variables to be accessed via a single pointer
- eg.

```
struct friends_list{
        char name[10];          /*name of  a friend*/
        int age;                /*age of a friend*/
        char telephone[13];     /*telephone of a friend*/
};
```



Introduction to Sockets
Part III: major system calls



Introduction to Sockets
Part IV: sample programs