网络编程简介

授课老师：赵增华
助教：杨金峰
邮件： myjfm@163.com

两台计算机通过网络进行通信

IP 地址

- IP 网络中每台主机都必须有一个惟一的 IP 地址；
- IP 地址是一个逻辑地址；
- 因特网上的 IP 地址具有全球唯一性；
- 32 位， 4 个字节，常用点分十进制的格式表示，例如： 192.168.0.16

端口

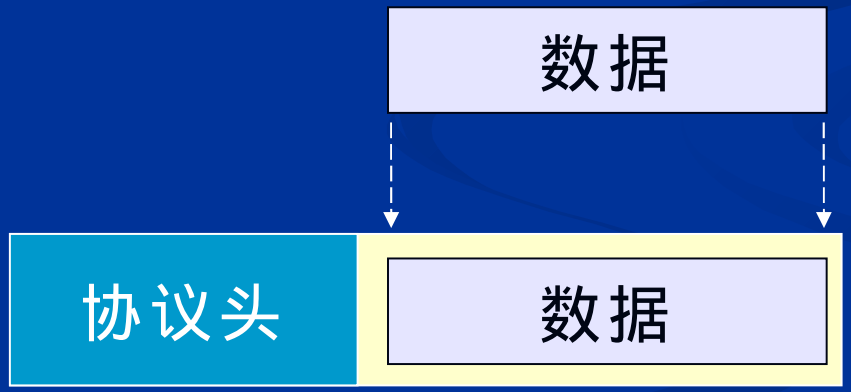
- 端口是一种抽象的软件结构（包括一些数据结构和 I/O 缓冲区）。应用程序通过系统调用与某端口建立连接（ binding ）后，传输层传给该端口的数据都被相应的进程所接收，相应进程发给传输层的数据都通过该端口输出。
- 端口用一个整数型标识符来表示，即端口号。端口号跟协议相关， TCP/IP 传输层的两个协议 TCP 和 UDP 是完全独立的两个软件模块，因此各自的端口号也相互独立，端口通常称为协议端口 (protocol port) ，简称端口。
- 端口使用一个 16 位的数字来表示，它的范围是 0~65535 ， 1024 以下的端口号保留给预定义的服务。例如： http 使用 80 端口。

协议

- 为进行网络中的数据交换（通信）而建立的规则、标准或约定。（= 语义 + 语法 + 规则）
- 不同层具有各自不同的协议。

数据封装

- 一台计算机要发送数据到另一台计算机，数据首先必须打包，打包的过程称为封装。
- 封装就是在数据前面加上特定的协议头部。



TCP/IP 模型

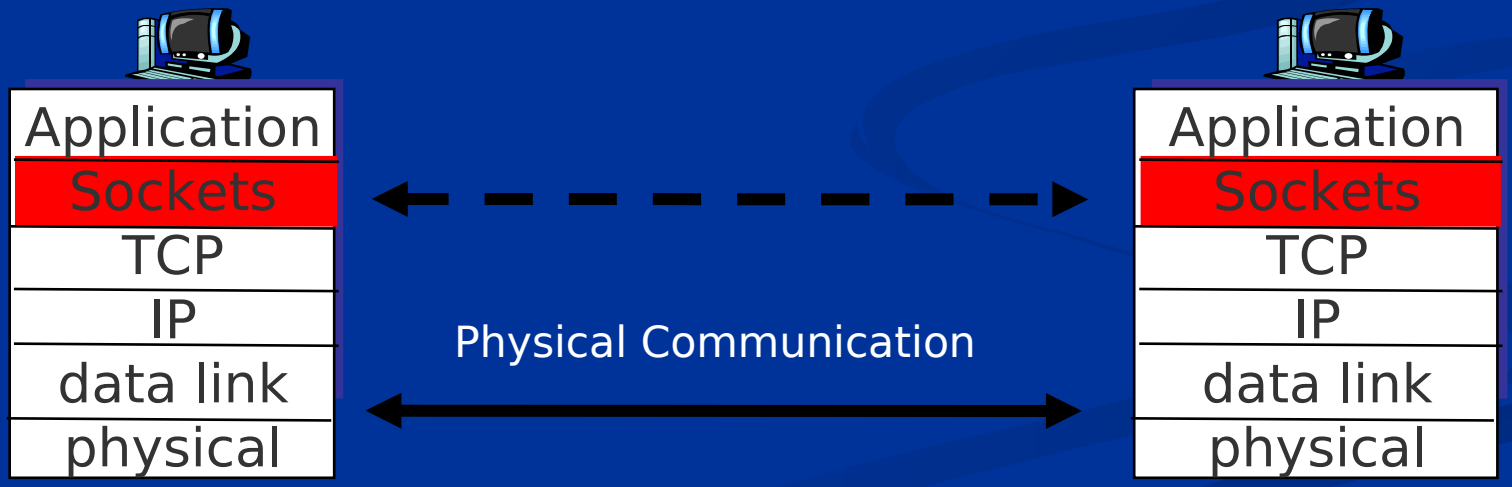
- **TCP/IP** 起源于美国国防部高级研究规划署 **(DARPA)** 的一项研究计划——实现若干台主机的相互通信。
- 现在 **TCP/IP** 已成为 **Internet** 上通信的工业标准。
- **TCP/IP** 模型包括 **5** 个层次：
  - 应用层
  - 传输层
  - 网络层
  - 数据链路层
  - 物理层

套接字 (socket) 的引入

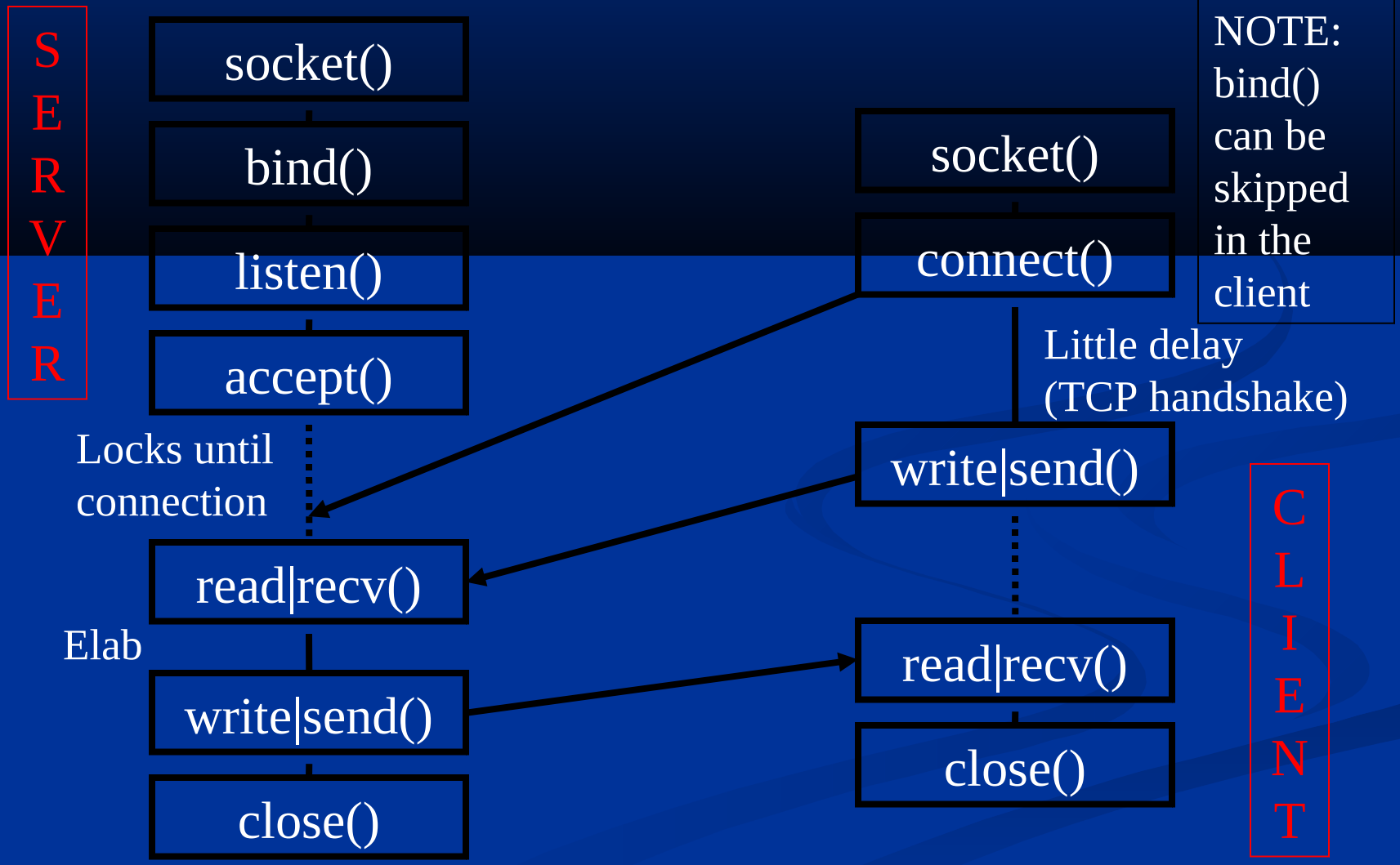
- 为了能够方便的开发网络应用软件，由美国伯克利大学在 Unix 上推出了一种应用程序访问通信协议的操作系统调用 socket(套接字)。 socket 的出现，使程序员可以很方便地访问 TCP/IP ，从而开发各种网络应用的程序。
- 随着 Unix 的应用推广，套接字在编写网络软件中得到了极大的普及。后来，套接字又被引进了 Windows 等操作系统中。 Java 语言也引入了套接字编程模型。

Socket

- Socket 是由应用层程序创建，用于和传输层通信的接口
- socket 由 OS 维护，类似于文件描述符。
- 它允许与远程主机上的进程发送 / 接收消息。



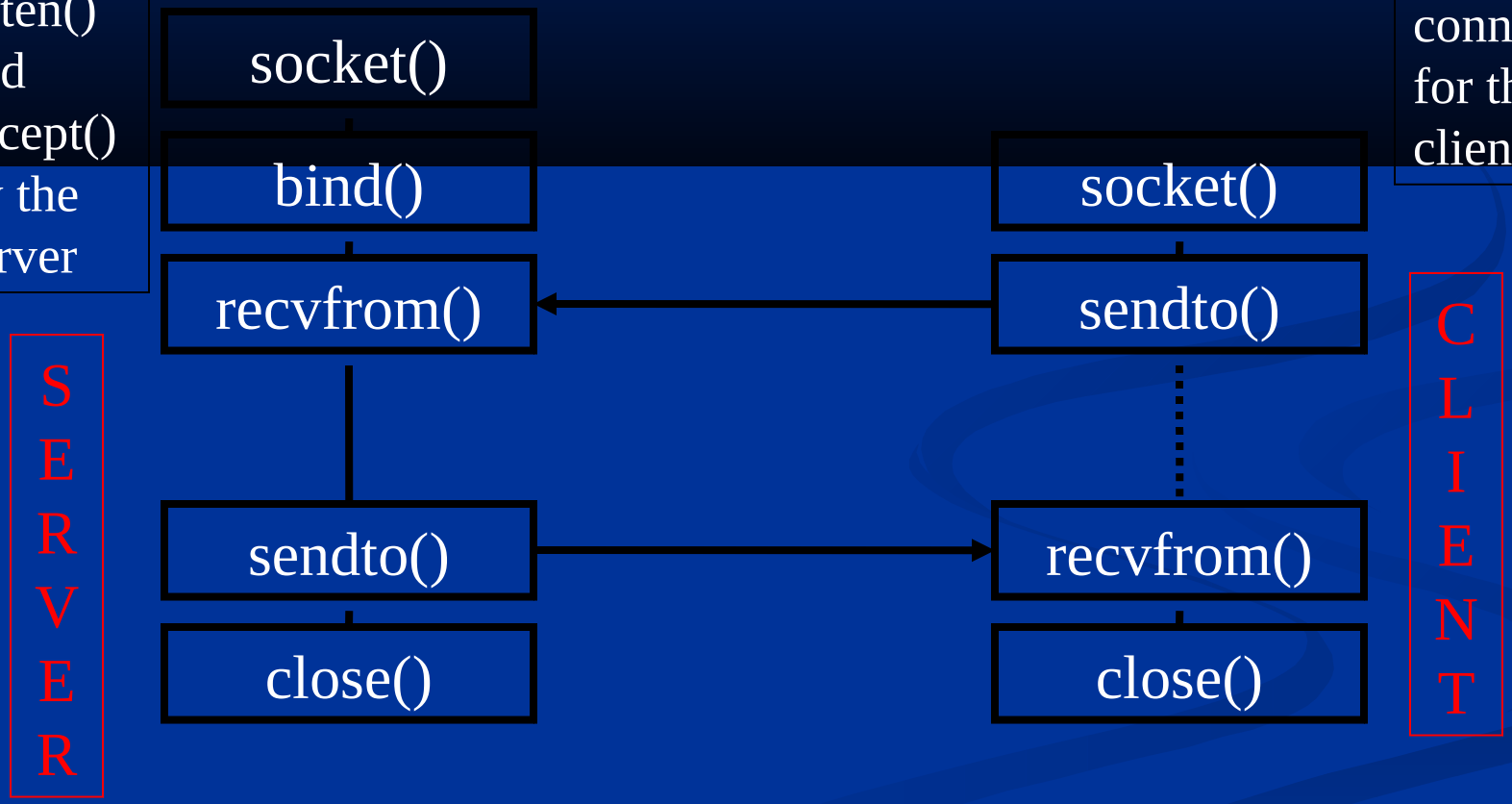
TCP API : a session



UDP API: a session

NOTE: no listen() and accept() by the server

NOTE: no connect() for the client



Socket API: socket()

要使用 socket 用于通信，首先需要创建 socket：

```
/*socket_type=SOCK_STREAM|SOCK_DGRAM*/
/*AF_INET means we use IP (version 4)*/
/*0 is for IP (not ICMP or others)*/
int sd = socket(AF_INET, socket_type, 0);
```

Protocol Family	**Symbolic Name**
TCP/IP Internet	AF_INET
Xerox NS	AF_NS
Intra-host Unix	AF_UNIX
DEC DNA	AF_DECNET

Service Type	**symbolic name**
datagram (UDP)	SOCK_DGRAM
reliable, in order (TCP)	SOCK_STREAM
raw socket	SOCK_RAW

Socket API：bind()

功能：将套接字绑定上本机 IP 地址以及端口号

```
struct sockaddr local_addr;
int size = sizeof(local_addr);
bind (sd, &local_addr, size);
```

Note:

1、只有服务器端必需调用此函数，客户端不需要

2、传递的是 sockaddr 结构而不是 sockaddr_in 结构

一般我们需要操作 sockaddr_in 结构，然后将其强制类型转换为 sockaddr 结构即可。

指定 IP 地址

三种形式：

1、主机名 (string) 如 localhost

2、点分十进制 (string) 如 192.168.1.8

3、二进制 (u_long)

inet_addr(): 点分十进制 to 二进制

gethostbyname(): 主机名 to 二进制

指定端口

1-255	reserved for standard services
21	ftp
23	telnet
25	SMTP
80	HTTP
1-1023	Available only to priviledged users
1024-4999	Usuable by system and user processes
5000-	Usuable by user processes only

两个需要注意的结构

```
struct sockaddr_in
{
    u_char sin_len;
    u_short sin_family;
    u_short sin_port;
    struct in_addr sin_addr;
    char sin_zero[8];
};
```

```
struct sock_addr
{
    u_char sa_len;
    u_short sa_family;
    char sa_data[14];
};
```

```
struct in_addr
{
    /* 32 位 ip 地址，某些系统以 u_long 代替 */
    in_addr_t s_addr;
};
```

Socket API: close(), shutdown()

直接关闭套接字：（类似于关闭文件）

```
close(sd);
```

关闭单向 / 双向连接：

```
shutdown(sd, how);
/* how = 0 -> disable receptions */
/* how = 1 -> disable transmission */
/* how = 2 -> the same as close() */
```

TCP API (SERVER): listen()

功能：将套接字置为被动套接字

```
listen (sd, req_no);
```

req_no：允许请求连接的队列项的最大值

Note:

1、服务器可能会在处理当前请求的同时又有新的连接请求到来，因此应该设置队列以排队请求

2、调用 listen() 之后服务器还并不能接收连接请求

TCP API (SERVER) : accept()

功能：在相应套接字监听连接请求

```
int new_sd, sd, size;
struct sockaddr remote_addr;
new_sd = accept(sd, &remote_addr, &size);
```

note:

1、当队列中没有连接请求时 accept 将服务器进程阻塞。

2、利用 sd 进行连接请求的监听，而 accept 返回的是一个新的 socket 描述符 new_sd，利用这个新的描述符处理当前连接请求

3、可以创建一个子进程去处理 new_sd，也可以创建线程处理

TCP API: send()

功能：经套接字向远程端点的进程发送信息

```
int nsent, sd, bytes_no, flags;
const char msg[SOME_SIZE];
nsent = send(sd, &msg, bytes_no, flags);
```

sd：套接字描述符
&msg：缓冲区地址
bytes_no：想要发送数据的字节数
falgs：一般置 0.

TCP API: recv()

功能：接收远程端点的进程发送来的信息

```
int nrecv, sd, bytes_no, flags;
char buffer[SOME_SIZE];
nrecv = recv(sd, &buffer, bytes_no, flags)
```

flag:

MSG_DONTROUTE：	不查找路由表（用于 send 函数）
MSG_PEEK：	查看数据，并不从系统缓冲区移走数据
MSG_WAITALL：	等待所有数据

...

TCP API: read(), write()

read() 与 recv(x, x, x, 0) 作用相同

write() 与 send(x, x, x, 0) 作用相同

read()/write() 优点：程序员对之熟悉，与操作普通文件很类似

recv()/send() 优点：程序中的语义明显

TCP API (CLIENT): connect()

功能：客户端通过套接字 sd 向服务器端发起 tcp 连接

```
struct sockaddr remote_addr;
connect(sd, remote_addr, size);
```

Note:

1、自动完成了 bind 函数的功能，这就是为什么 client 端不需要调用 bind 函数得原因

2、完成了三路握手的过程

UDP API: sendto(), recvfrom()

功能：向 / 由目的主机发送 / 接收 udp 数据报

```
int nsent, nrecv, sd, bytes_no, flags;
int to_size, from_size;
const char msg;
char buffer;
const struct sockaddr to;
struct sockaddr from;

nsent = sendto(sd, &msg, bytes_no, flags,
   &to, to_size);
nrecv = recvfrom(sd, &buffer, bytes_no, flags,
   &from, &from_size);
```

note:

1、无须握手过程， server 端直接调用 recvfrom 等待 client 端发送数据报

2、前四个参数和 send/recv 相同，后两个参数：

to/from ： 目的主机地址结构，包括目的主机 IP 及端口号

to_size/from_size:to/from 结构体的大小

END !

Thank you !