

# Chapitre VII SOCKET PROGRAMMING

---

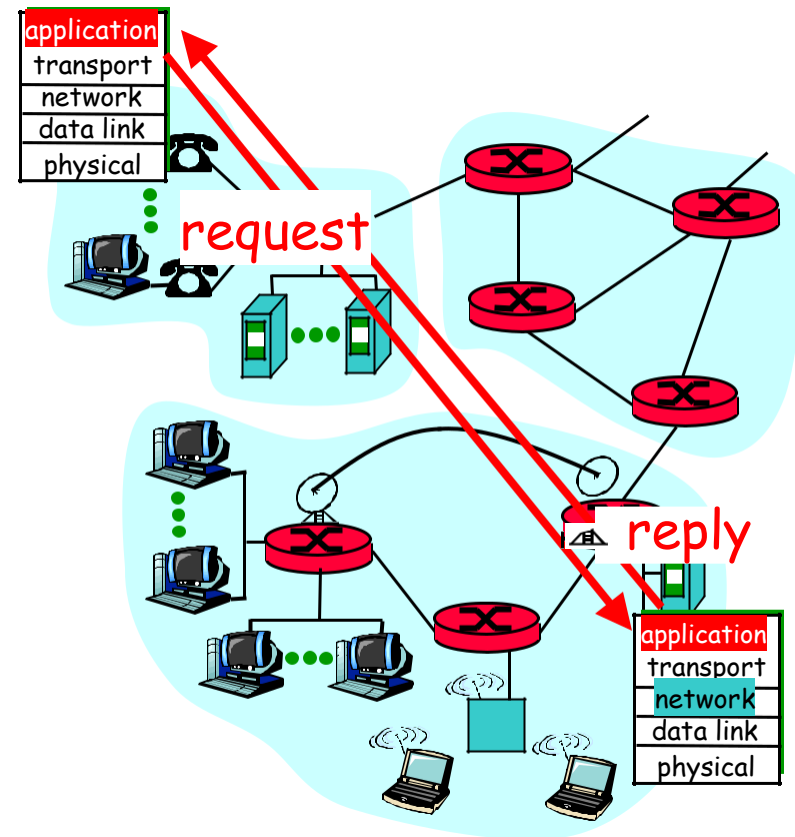
# Client-server paradigm

## Client:

initiates contact with server (“speaks first”)  
typically requests service from server,  
for Web, client is implemented in browser;  
for e-mail, in mail reader

## Server:

provides requested service to client  
e.g., Web server sends requested Web page, mail server delivers e-mail



# Application Layer Programming

**API: application programming interface**

**defines interface between application and transport layer**

**sockets: Internet API**

- two processes communicate by sending data into socket, reading data out of socket**

# Socket Interface. What is it?

**Gives a file system like abstraction to the capabilities of the network.**

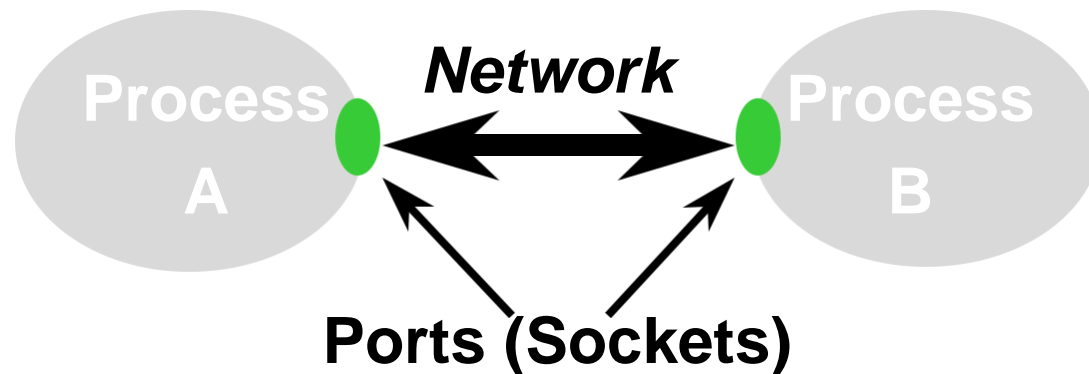
**Each transport protocol offers a set of services. The socket API provides the abstraction to access these services**

**The API defines function calls to create, close, read and write to/from a socket.**

# Socket Abstraction

**The *socket* is the basic abstraction for network communication in the socket API**

- Defines an endpoint of communication for a process
- Operating system maintains information about the socket and its connection
- Application references the socket for sends, receives, etc.



What do you need for socket communication ??

## **Basically 4 parameters**

- Source Identifier (IP address)**
- Source Port**
- Destination Identifier**
- Destination Port**

**In the socket API, this information is communicated by binding the socket.**

# Creating a socket

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

Protocol Family:  
PF\_INET or  
PF\_UNIX

Communication  
semantics:  
SOCK\_STREAM or  
SOCK\_DGRAM

Usually  
UNSPEC

The call returns a integer identifier called a  
*handle*

# Binding a socket

```
int bind (int socket, struct sockaddr *address, int addr_len)
```

**This call is executed by:**

- Server in TCP and UDP**

**It binds the socket to the specified address. The address parameter specifies the local component of the address, e.g. IP address and UDP/TCP port**



# Socket Descriptors

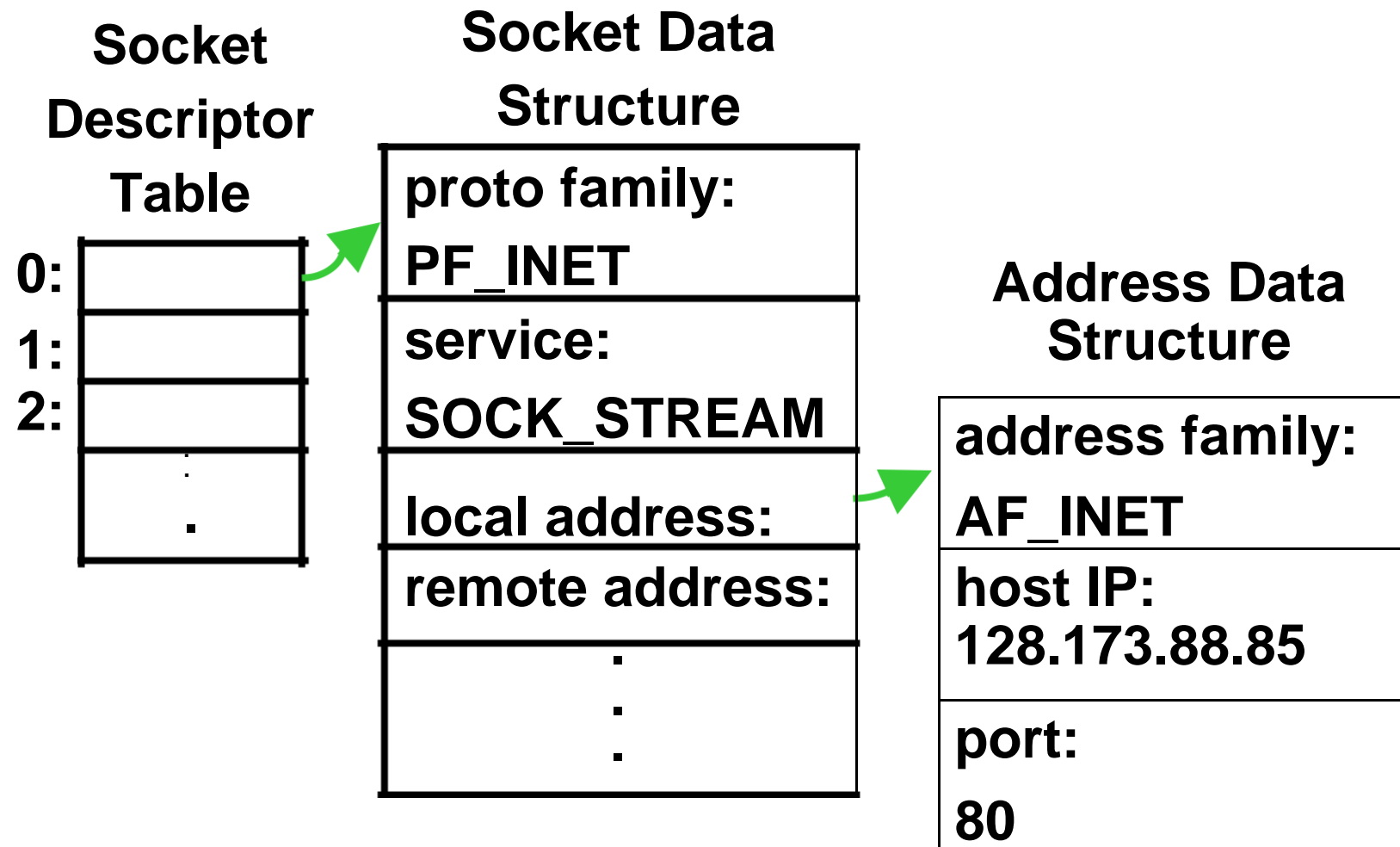
**Operating system maintains a set of socket descriptors for each process**

- Note that socket descriptors are shared by threads**

## **Three data structures**

- Socket descriptor table**
- Socket data structure**
- Address data structure**

# Socket Descriptors



# TCP Server Side: Listen

```
int listen (int socket, int backlog)
```

**This server side call specifies the number of pending connections on the given socket.**

**When the server is processing a connection, “backlog” number of connections may be pending in a queue.**

# TCP Server Side: Passive Openen

```
int accept (int socket, struct sockaddr *address, int *addr_len)
```

**This call is executed by the server.**

**The call does not return until a remote client has established a connection.**

**When it completes, it returns a new socket handle corresponding to the just-established connection**

# TCP Client Side: Active Open

```
int connect (int socket, struct sockaddr *address, int *addr_len)
```

**This call is executed by the client.**

**\*address contains the remote address.**

**The call attempts to connect the socket to a server. It does not return until a connection has been established.**

**When the call completes, the socket “socket” is connected and ready for communication.**

# Sockets: Summary

## **Client:**

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

`int connect (int socket, struct sockaddr *address, int addr_len)`

## **Server:**

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

`int bind (int socket, struct sockaddr *address, int addr_len)`

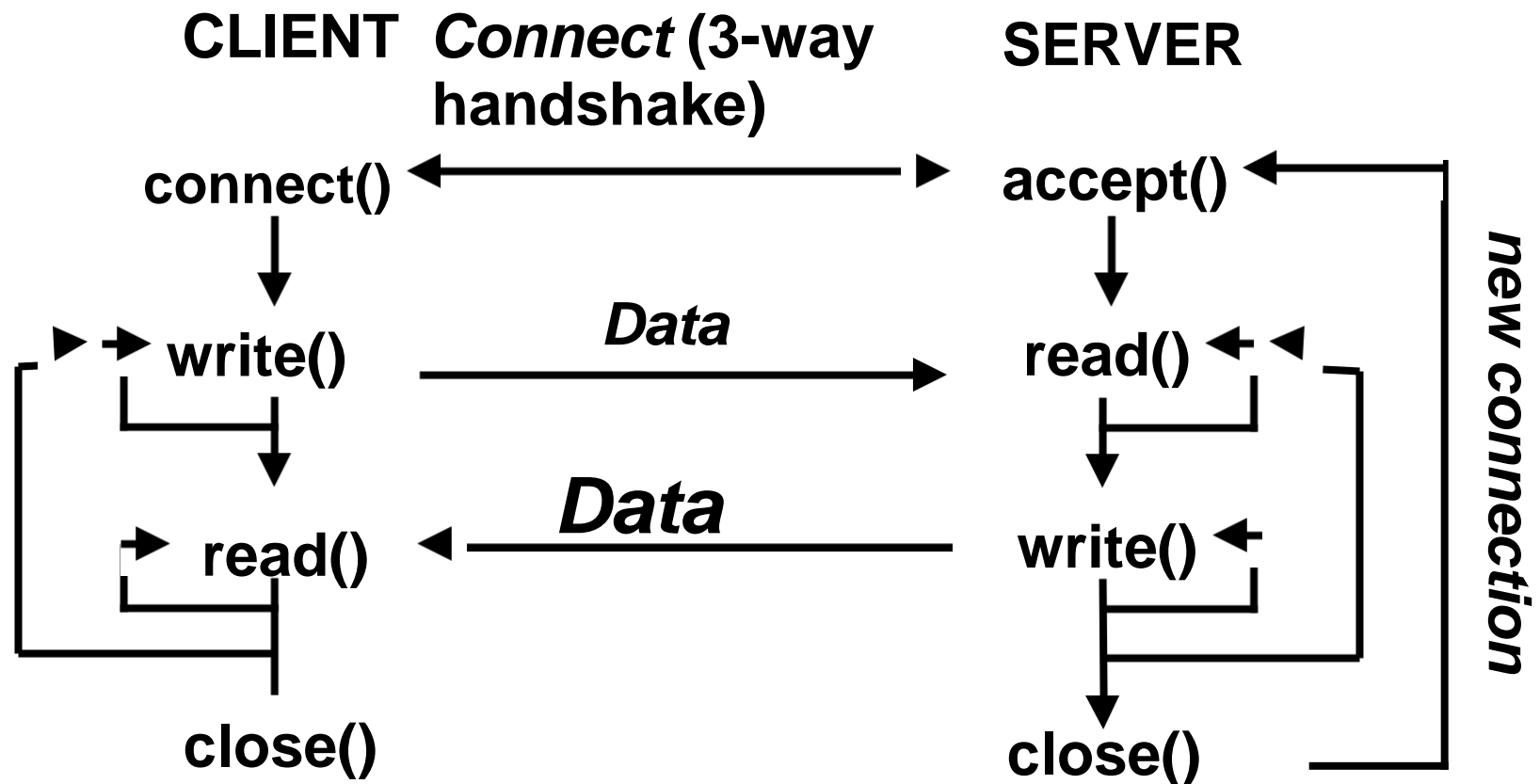
`int listen (int socket, int backlog)`

`int accept (int socket, struct sockaddr *address, int *addr_len)`

# Message Passing

```
int send (int socket, char *message, int msg_len,  
          int flags) (TCP)  
  
int sendto (int socket, void *msg, int len,  
            int flags, struct sockaddr * to,  
            int tolen ); (UDP)  
  
int write(int socket, void *msg, int len); /* TCP */  
  
  
int recv (int socket, char *buffer, int buf_len,  
          int flags) (TCP)  
  
int recvfrom(int socket, void *msg, int len,  
             int flags, struct sockaddr *from,  
             int *fromlen); (UDP)  
  
int read(int socket, void *msg, int len); (TCP)
```

# Summary of Basic Socket Calls





# Network Byte Order

**Network byte order is most-significant byte first**

**Byte ordering at a host may differ**

**Utility functions**

- **htons(): Host-to-network byte order for a short word (2 bytes)**
- **htonl(): Host-to-network byte order for a long word (4 bytes)**
- **ntohs(): Network-to-host byte order for a short word**
- **ntohl(): Network-to-host byte order for a long word**

# Some Other “Utility” Functions

**gethostname() -- get name of local host**

**getpeername() -- get address of remote host**

**getsockname() -- get local address of socket**

**getxby Y() -- get protocol, host, or service number using known number, address, or port, respectively**

**getsockopt() -- get current socket options**

**setsockopt() -- set socket options**

**ioctl() -- retrieve or set socket information**

## Some Other “Utility” Functions

**inet\_addr() -- convert “dotted”  
character string form of IP address  
to internal binary form**

**inet\_ntoa() -- convert internal  
binary form of IP address to  
“dotted” character string form**

# Address Data Structures

```
struct sockaddr {  
    u_short    sa_family;    // type of address  
    char       sa_data[14];  // value of address  
}
```

## **sockaddr is a generic address structure**

```
struct sockaddr_in {  
    u_short    sa_family;      // type of address (AF_INET)  
    u_short    sa_port;        // protocol port number  
    struct      in_addr sin_addr; // IP address  
    char        sin_zero[8];    // unused (set to zero)  
}
```

**sockaddr\_in is specific instance for the Internet address family**