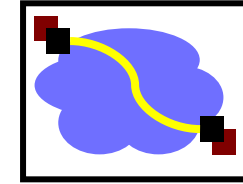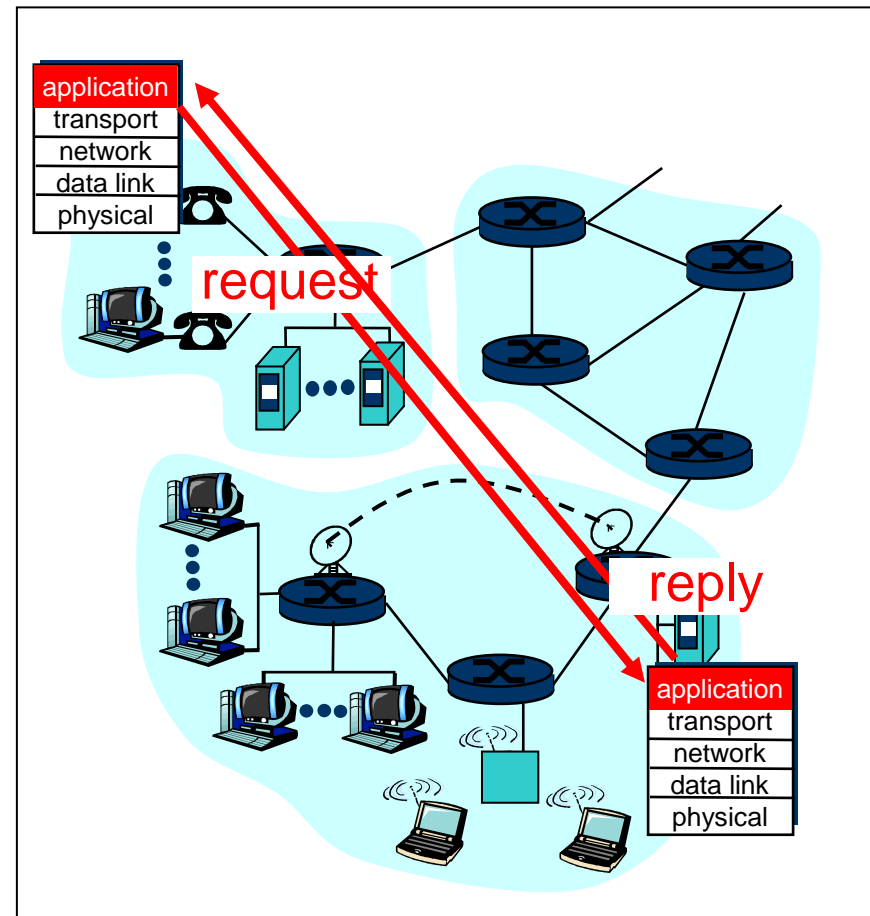# Socket Programming

Client-Server Paradigm

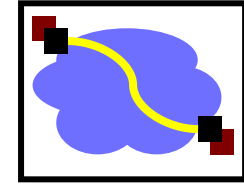Typical network app has two pieces: *client* and *server*

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
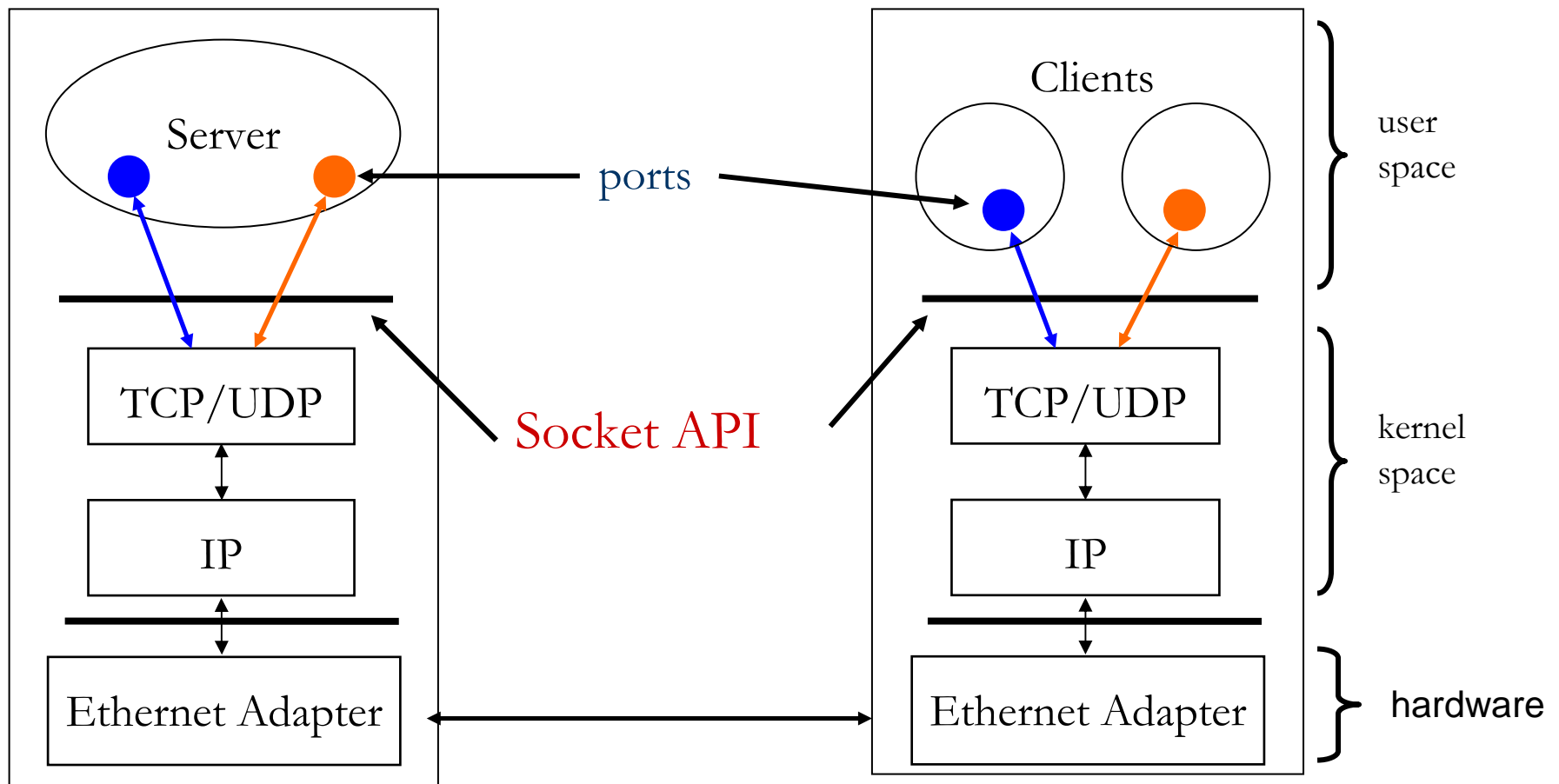
Server:

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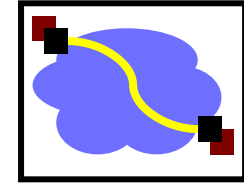

application
transport
network
data link
physical

request

reply

application
transport
network
data link
physical

Server and Client

Server and Client exchange messages over the network
through a common Socket API

UDP and TCP

UDP

- Single socket to receive messages
- No guarantee of delivery
- Not necessarily in-order delivery
- Datagram – independent packets
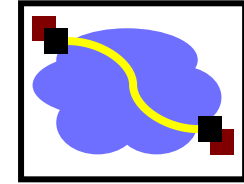- Must address each packet

TCP

- Reliable – guarantee delivery
- Byte stream – in-order delivery
- Connection-oriented – single socket per connection
- Setup connection followed by data transfer

Example UDP applications
Multimedia, voice over IP

Example TCP applications
Web, Email, Telnet

Internet Addressing Data Structure

```c
#include <netinet/in.h>

/* Internet address structure */
struct in_addr {
        u_long s_addr;            /* 32-bit IPv4 address */
};                                /* network byte ordered */


/* Socket address, Internet style. */
struct sockaddr_in {
      u_char   sin_family;        /* Address Family */
      u_short  sin_port;          /* UDP or TCP Port# */
                                  /* network byte ordered */
      struct in_addr sin_addr;    /* Internet Address */
      char     sin_zero[8];       /* unused */
};
```
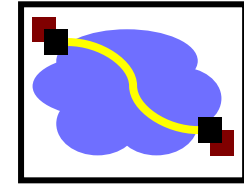
Byte Ordering

128.2.194.95

	c[0]	c[1]	c[2]	c[3]

- Big Endian
  - Sun Solaris, PowerPC, ...

128	2	194	95

- Little Endian
  - i386, alpha, ...

95	194	2	128

- Network byte order = Big Endian

Byte Ordering Functions

- Converts between **host byte order** and **network byte order**
  - 'h' = host byte order
  - 'n' = network byte order
  - 'l' = long (4 bytes), converts IP addresses
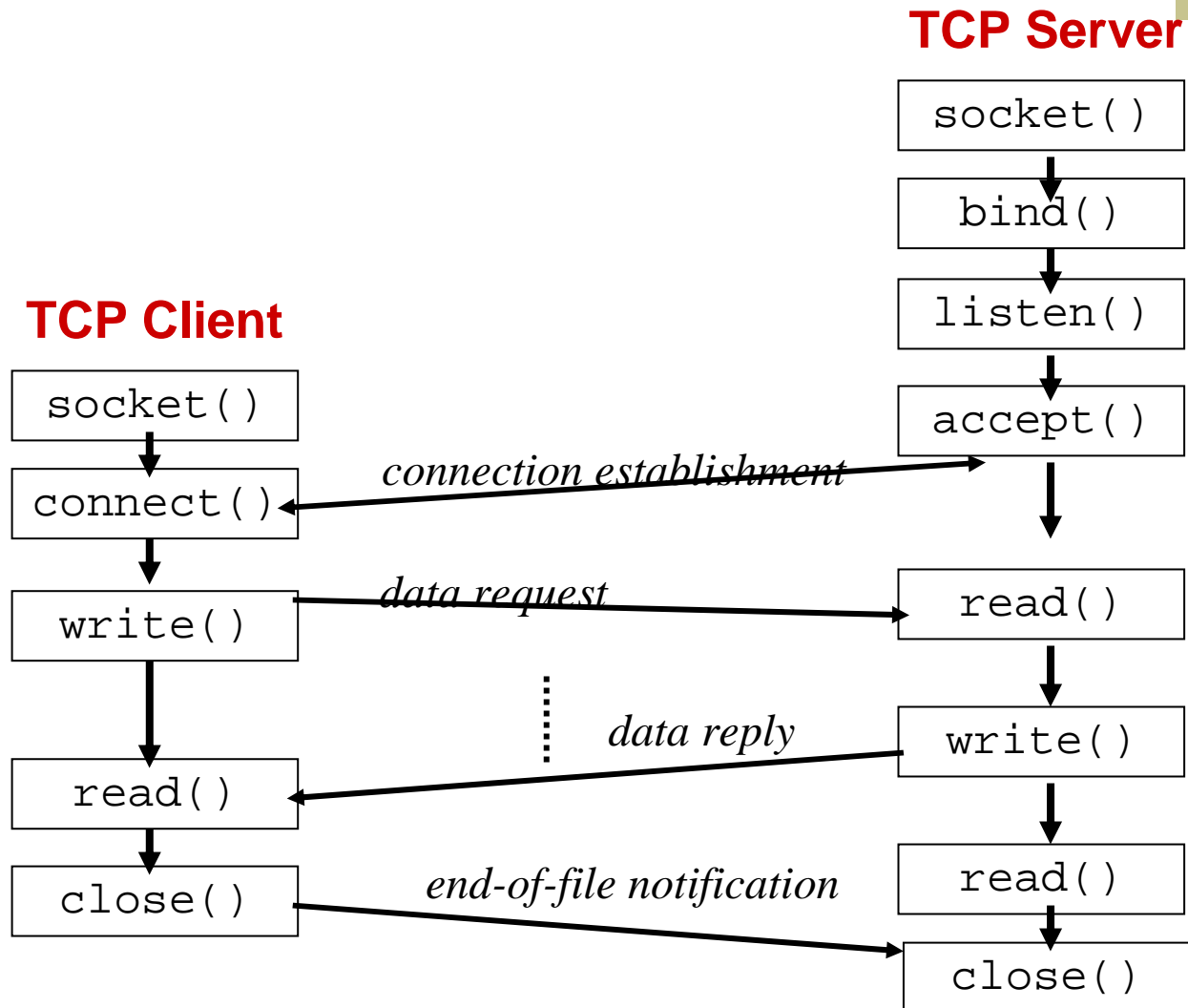 - 's' = short (2 bytes), converts port numbers

```
#include <netinet/in.h>

unsigned long int htonl(unsigned long int hostlong);
unsigned short int htons(unsigned short int
hostshort);
unsigned long int ntohl(unsigned long int netlong);
unsigned short int ntohs(unsigned short int
netshort);
```
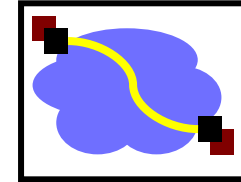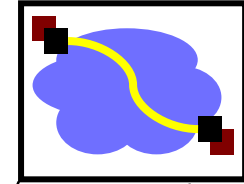
TCP Client-Server Interaction

TCP Server

```
socket()
```
↓
```
bind()
```
↓
```
listen()
```
↓
```
accept()
```

TCP Client

```
socket()
```
↓
```
connect()
```

connection establishment

↓
```
write()
```

data request

↓
```
read()
```

data reply

↓
```
close()
```

end-of-file notification

```
read()
```
↓
```
write()
```
↓
```
read()
```
↓
```
close()
```

Socket Programming

8

What is a Socket?

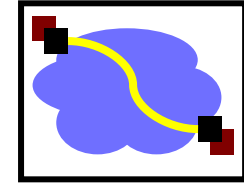- A socket is a file descriptor that lets an application read/write data from/to the network

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

  - **Domain / family** : integer, communication domain
    - AF_INET   IPv4 protocol
    - AF_INET6   IPv6 protocol
    - AF_LOCAL   Unix Domain Protocols
    - AF_ROUTE Routing Sockets
    - AF_KEY   Key Socket
  - **type:** communication type
    - SOCK_STREAM: reliable, 2-way, connection-based service
    - SOCK_DGRAM: unreliable, connectionless,
    - SOCK_RAW: Security
  - **protocol:** specifies protocol - usually set to 0
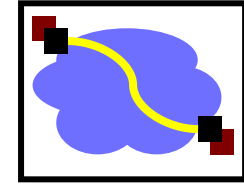
What is a Socket?

```
int sd;            /* socket descriptor */
if ((sd = socket(AF_INET, SOCK_STREAM, 0)) < 0) }
     perror("socket");
     exit(1);
}
```

socket returns an integer (socket descriptor)

- sd < 0 indicates that an error occurred
- socket descriptors are similar to file descriptors

NOTE: socket call does not specify where data will be coming from, nor where it will be going to – it just creates the interface!
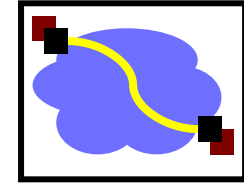
Binding a Socket to a Port

- A *socket* can be bound to a *port*
- ie. reserves a port for use by the socket

int status = bind(int sockfd, struct sockaddr *address, int addrlen);

- **sockfd:** integer, socket descriptor
- **address:** struct sockaddr, the (IP) address and port of the machine (address usually set to INADDR_ANY – chooses a local address)
- **addrlen:** the size (in bytes) of the address structure
- **status:** Successful completion returns 0
  if bind failed = -1

Binding a Socket to a Port

```
int sd;                              /* socket descriptor */
struct sockaddr_in server;/* used by bind() */


/* 1) create the socket */
server.sin_family = AF_INET; /* use the Internet addr family */


server.sin_port = htons(80); /* bind socket 'sd' to port 80*/


/* bind: a client may connect to any of my addresses */
server.sin_addr.s_addr = htonl(INADDR_ANY);


if(bind(sd, (struct sockaddr*) & server, sizeof(server)) < 0) {
        perror("bind"); exit(1);
}
```
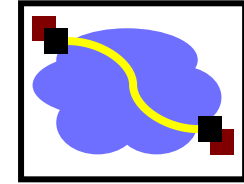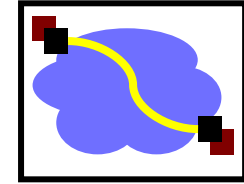
Listening For Connections

- **listen** indicates that the server will accept a connection

int status = listen(int sockfd, int backlog);

- **sockfd:** socket descriptor
- **backlog:** maximum # of active participants that can "wait" for a connection
- **status:** 0 if listening, -1 if error

Listening For Connections
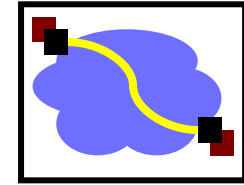
```
int sd;                       /* socket descriptor */
struct sockaddr_in srv;    /* used by bind() */


/* 1) create the socket */
/* 2) bind the socket to a port */

if(listen(sd, 5) < 0) {
        perror("listen");
        exit(1);
}
```

Accepting a connection

- Takes the first connection request on the queue, creates another socket with the same properties of sockfd.
- If no connection request pending, blocks the server until it receives a connection request from client
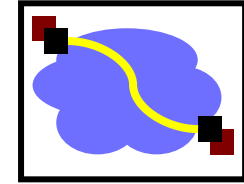
int status =accept(int sockfd, struct sockaddr *addr, int *addrlen);

- **sockfd:** the orig. socket (being listened on)
- **addr:** address of the active participant
- **addrlen:** value/result parameter
- **status:** the new socket (used for data-transfer)
  - Successful completion returns 0
  - Error -1

Accepting a connection

```c
int sd;                                 /* socket descriptor */
struct sockaddr_in server;      /* used by bind() */
struct sockaddr_in client;       /* used by accept() */
int newfd;                              /* returned by accept() */
int clientlen = sizeof(client);        /* used by accept() */

/* 1) create the socket */
/* 2) bind the socket to a port */
/* 3) listen on the socket */

newfd = accept(sd, (struct sockaddr*) &client &clientlen);
if(newfd < 0) {
        perror("accept");      exit(1);
}
```
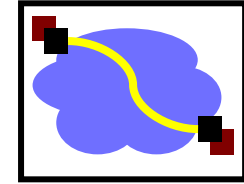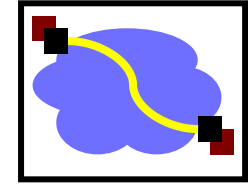
Accepting a connection

- How does the server know which client it is?
  - **cli.sin_addr.s_addr** contains the client's **IP address**
  - **cli.sin_port** contains the client's **port number**
- Now the server can exchange data with the client by using **read** and **write** on the descriptor **newfd**.
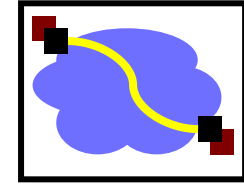
Connecting a Socket

- Attempts to make a connection on a socket

int status = connect(int sockfd, struct sockaddr *addr, int addrlen);

- **sockfd:** socket to be used in connection
- **addr:** address of passive participant
- **addrlen:** size of addr
- **status:** 0 if successful connect, -1 otherwise

Connecting a Socket
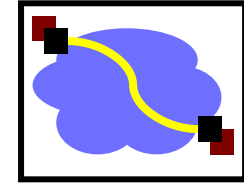
```
int sd;                              /* socket descriptor */
struct sockaddr_in server;           /* used by connect() */
/* 1)create the socket */


server.sin_family = AF_INET; /* connect: use the Internet address
family */
server.sin_port = htons(80); /* connect: socket 'sd' to port 80 */


/* connect: connect to IP Address "128.2.35.50" */
server.sin_addr.s_addr = inet_addr("128.2.35.50");


if(connect(sd, (struct sockaddr*) &server, sizeof(server)) < 0) {
        perror("connect"); exit(1);
}
```

Sending data to a Socket
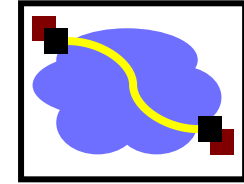
- *write* can be used with a socket

```
int sd;                        /* socket descriptor */
struct sockaddr_in server;   /* used by connect() */
char buf[512];                        /* used by write() */
int nbytes;                           /* used by write() */

/* 1) create the socket */
/* 2) connect() to the server */

/* Example: A client could "write" a request to a server */
if((nbytes = write(sd, buf, sizeof(buf))) < 0) {
        perror("write");        exit(1);
}
```

Receiving data from a socket

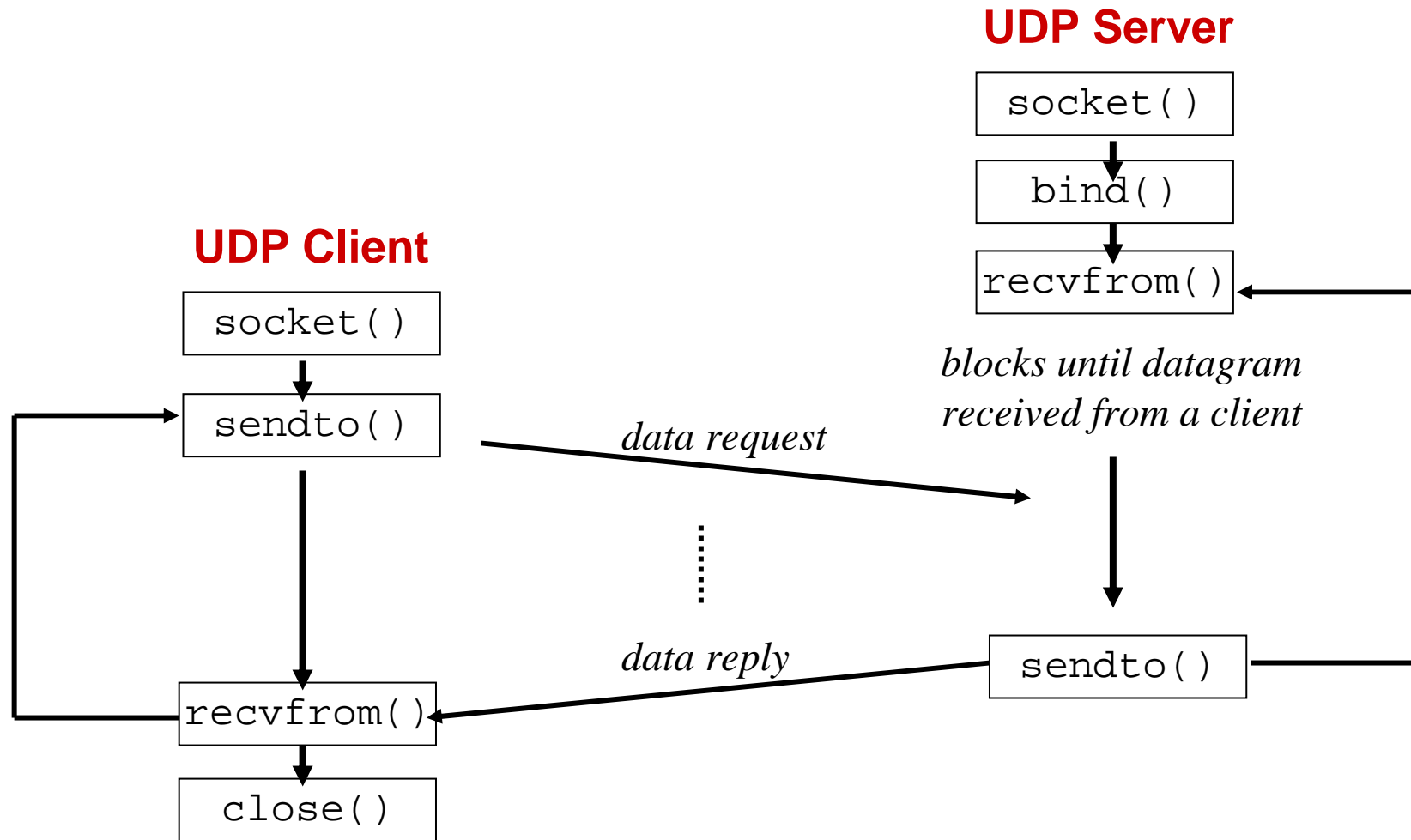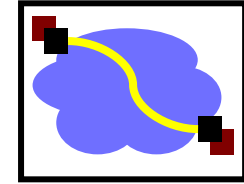- **read** can be used with a socket

```
int sd;                        /* socket descriptor */
char buf[512];                   /* used by read() */
int nbytes;                      /* used by read() */

/* 1) create the socket */
/* 2) bind the socket to a port */
/* 3) listen on the socket */
/* 4) accept the incoming connection */

if((nbytes = read(sd, buf, sizeof(buf))) < 0) {
        perror("read"); exit(1);
}
```
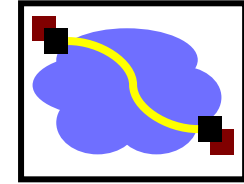
UDP Client-Server Interaction

UDP Server

```
socket()
```
↓
```
bind()
```
↓
```
recvfrom()
```

blocks until datagram received from a client

↓

UDP Client

```
socket()
```
↓
```
sendto()
```
→ *data request*

↓

```
recvfrom()
```
← *data reply*

```
sendto()
```

↓
```
close()
```

Simple Server Program

```c
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>

int main(int argc,char **argv)
{

int len;
int sockfd,newfd,n;
struct sockaddr_in servaddr,cliaddr;
char buff[1024];
char str[1000];
```
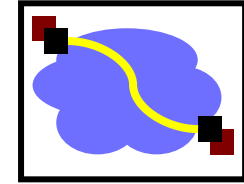
Simple Server Program

```
sockfd=socket(AF_INET,SOCK_STREAM,0);
if(sockfd<0)
  perror("cannot create socket");

bzero(&servaddr,sizeof(servaddr));

servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=INADDR_ANY;
servaddr.sin_port=htons(7228);

if(bind(sockfd,(struct sockaddr*)&servaddr,sizeof(servaddr))<0)
  perror("Bind error");

listen(sockfd,2);
```
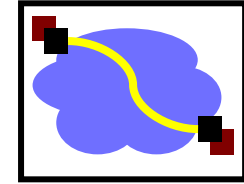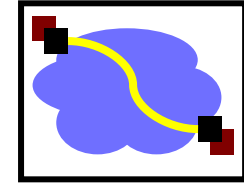
Simple Server Program

```
len=sizeof(cliaddr);
 newfd=accept(sockfd,(struct sockaddr*)&cliaddr,&len);
// printf("hi");
//Receiving the message

 n=read(newfd,buff,sizeof(buff));
 printf("\nReceived Message is \t%s",buff);

 close(sockfd);
 close(newfd);
 return 0;
}
```
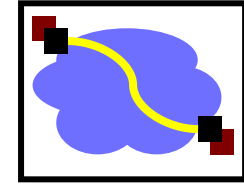
Simple Client Program

```c
#include<stdio.h>
#include<sys/types.h>
#include<sys/socket.h>
#include<netinet/in.h>
#include<string.h>

int main(int argc,char **argv)
{
int len;
int sockfd,n;
struct sockaddr_in servaddr,cliaddr;
```
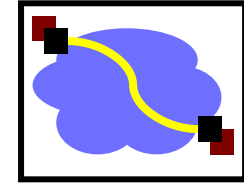
Simple Client Program

```
char str[1000];
char buff[1024];

sockfd=socket(AF_INET,SOCK_STREAM,0);
if(sockfd<0)
  perror("cannot create socket");

bzero(&servaddr,sizeof(servaddr));

servaddr.sin_family=AF_INET;
servaddr.sin_addr.s_addr=inet_addr(argv[1]);
servaddr.sin_port=htons(7228);
```

Simple Client Program

```
connect(sockfd,(struct sockaddr*)&servaddr,sizeof(servaddr));

//Sending Message
printf("Enter the message");
scanf("%s",buff);
n=write(sockfd,buff,sizeof(buff));


close(sockfd);
return 0;
}
```