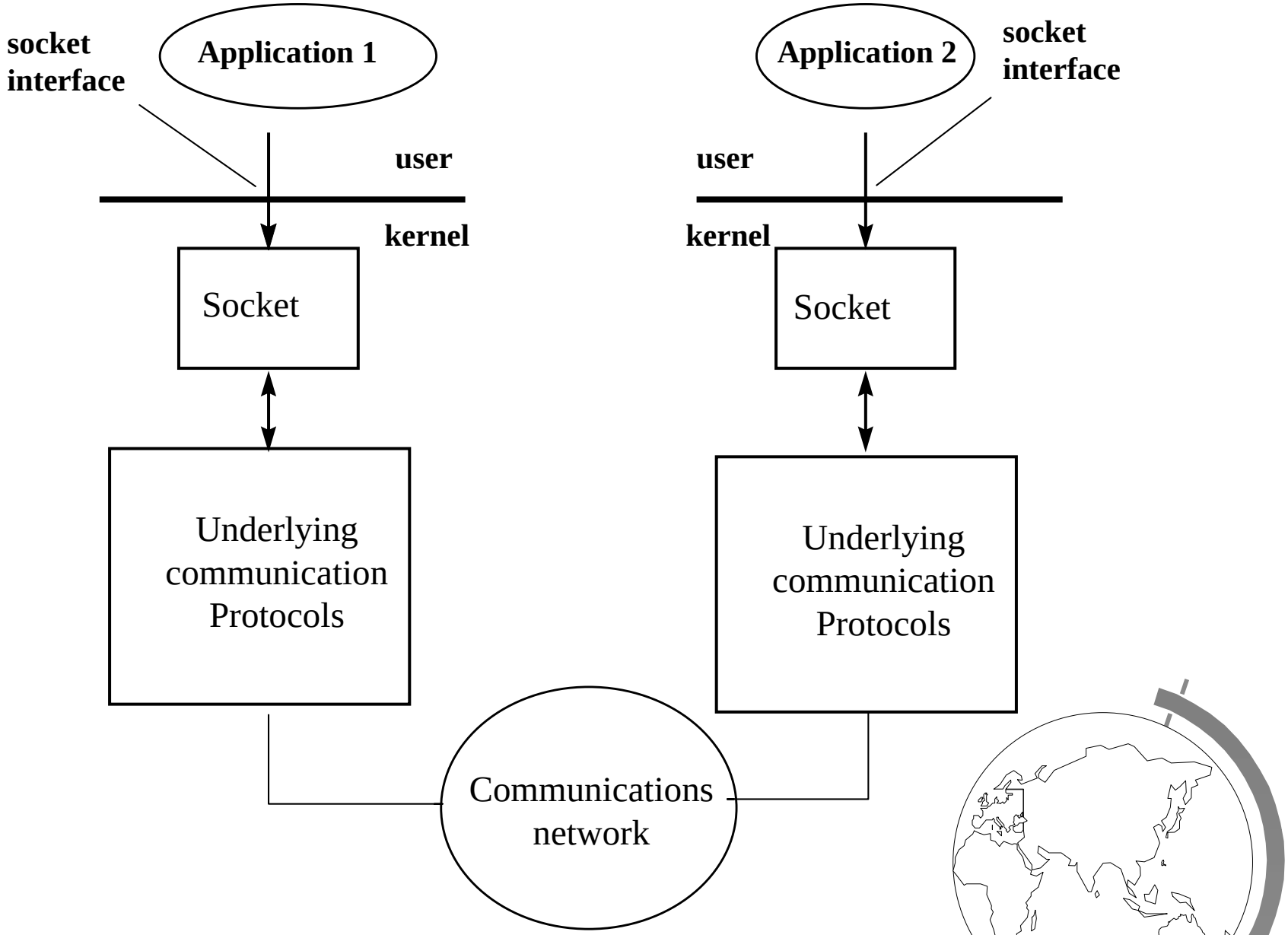
Introduction to LAN/WAN

Sockets

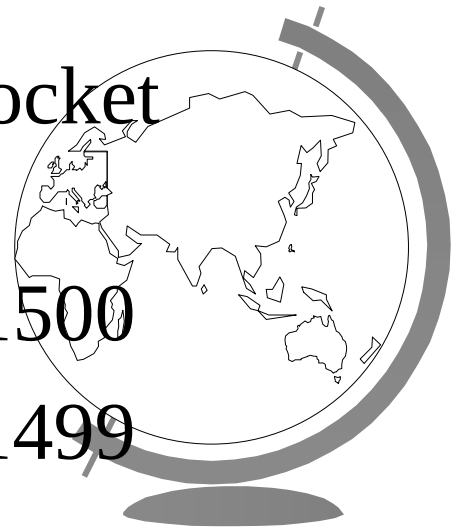
Outline

- Socket basics
- TCP sockets
- *Socket details*
- Socket options
- Final notes



Socket Basics

- An *end-point* for a IP network connection
  - what the application layer “plugs into”
  - programmer cares about Application Programming Interface (API)
- End point determined by two things:
  - Host address: IP address is *Network Layer*
  - Port number: is *Transport Layer*
- Two end-points determine a connection: socket pair
  - ex: 206.62.226.35,p21 + 198.69.10.2,p1500
  - ex: 206.62.226.35,p21 + 198.69.10.2,p1499



Ports

- Numbers (vary in BSD, Solaris):
  - 0-1023 “reserved”, must be root
  - 1024 – 49151 (registered with IANA)
  - 49152 – 65535 “ephemeral”
- /etc/services:

```
ftp 21/tcp
telnet 23/tcp
finger 79/tcp
snmp 161/udp
```

Sockets and the OS

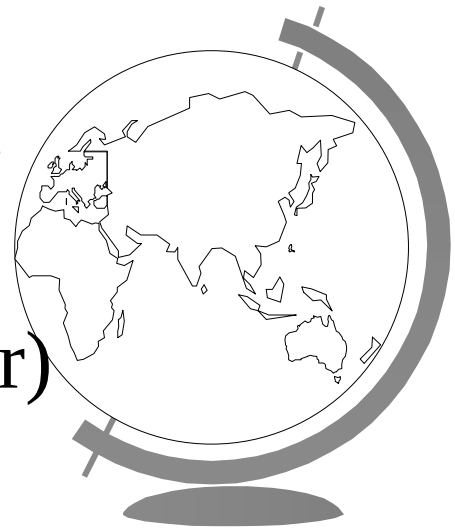
User

Socket

Operating System

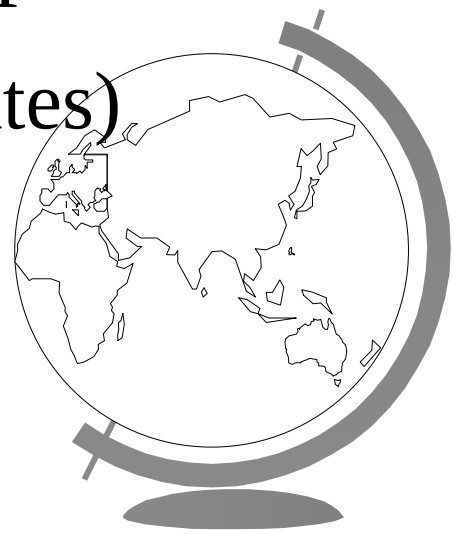
(Transport Layer)

- User sees “descriptor”, integer index
  - like: **`FILE *`**, or file index
  - returned by **`socket()`** call (more later)



Transport Layer

- UDP: User Datagram Protocol
  - no acknowledgements
  - no retransmissions
  - out of order, duplicate possible
  - connectionless
- TCP: Transmission Control Protocol
  - reliable (in order, all arrive, no duplicates)
  - flow control
  - connection
  - duplex



Socket Details

TCP/IP Sockets in C: Practical Guide for Programmers,
M J Donahoo and K Calvert ©2001 Morgan Kaufmann

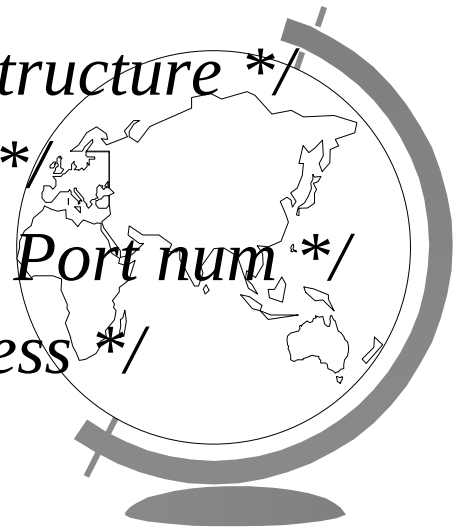
Sections 6.1.3-6.1.4 of text

- Socket address structure
- TCP client-server
- UDP client server
- Misc stuff
  - **`setsockopt(), getsockopt()`**

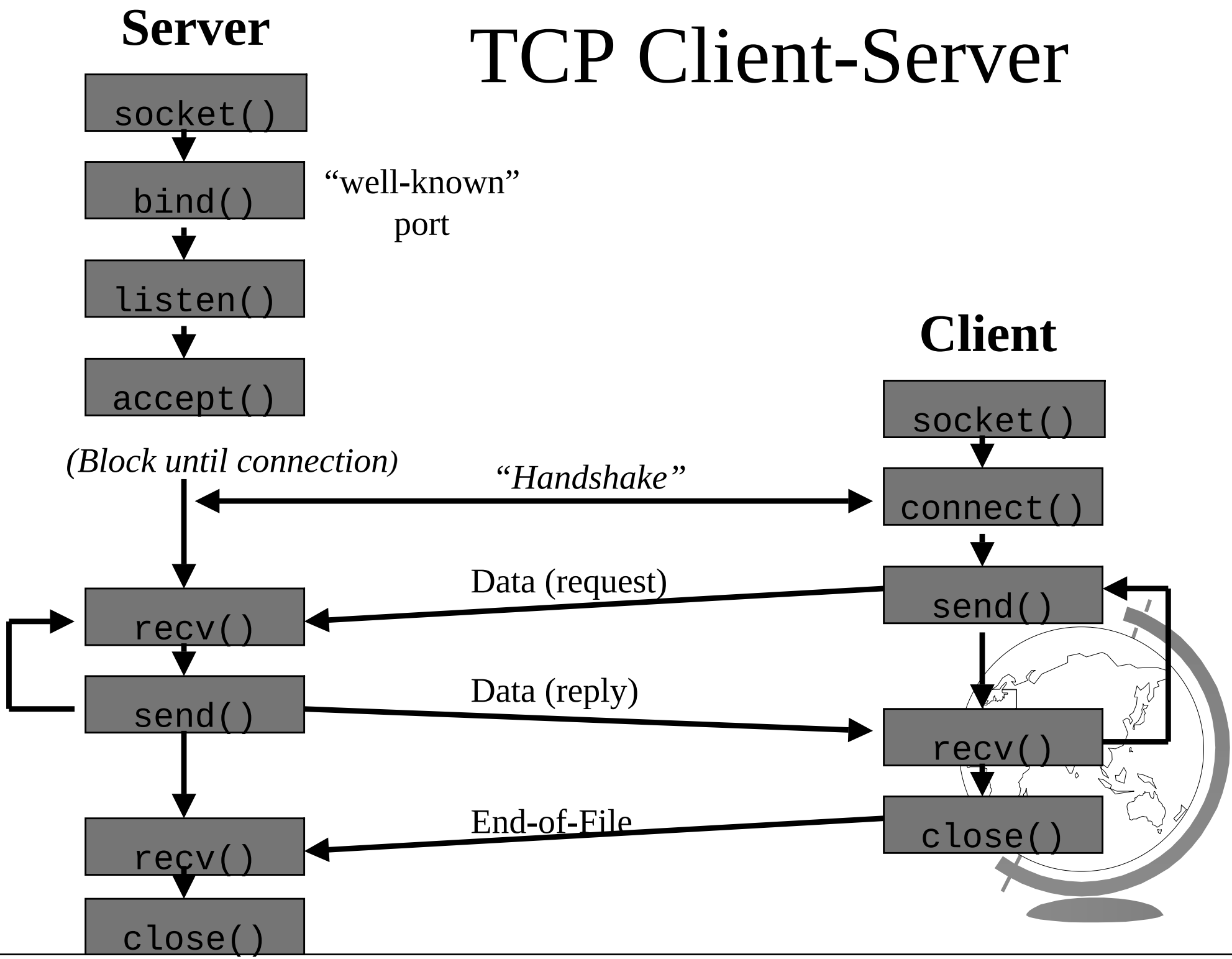
Socket Address Structure

```
#include  <sys/types.h>
#include  <sys/socket.h>

struct in_addr {
  in_addr_t s_addr;              /* 32-bit IPv4 addresses */
};
struct sock_addr_in {
  unit8_t       sin_len;         /* length of structure */
  sa_family_t   sin_family;      /* AF_INET */
  in_port_t     sin_port;        /* TCP/UDP Port num */
  struct in_addr sin_addr;       /* IPv4 address */
  char sin_zero[8];              /* unused */
}
```



TCP Client-Server



socket()

`int socket(int family, int type, int protocol);`

- Create a socket, giving access to transport layer service.

- *family* is one of
  - AF_INET (IPv4), AF_INET6 (IPv6), AF_LOCAL (local Unix),
  - AF_ROUTE (access to routing tables), AF_KEY (new, for encryption)
- *type* is one of
  - SOCK_STREAM (TCP), SOCK_DGRAM (UDP)
  - SOCK_RAW (for special IP packets, PING, etc. Must be root)
- *protocol* is 0 (used for some raw socket options)
- upon success returns socket descriptor
  - like file descriptor => -1 if failure
- Example:

```
If (( sockfd = socket (AF_INET, SOCK_STREAM, 0)) < 0)
    err_sys ("socket call error");
```

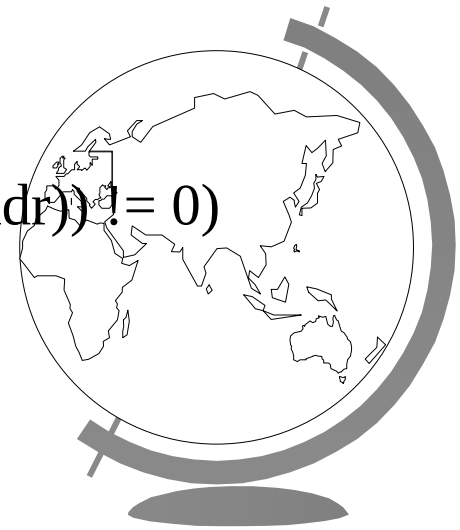
bind()

```
int bind(int sockfd, const struct sockaddr *myaddr,
         socklen_t addrlen);
```

Assign a local protocol address (“name”) to a socket.

- *sockfd* is socket descriptor from `socket()`
- *myaddr* is a pointer to address struct with:
  - *port number* and *IP address*
- *addrlen* is length of structure
- returns 0 if ok, -1 on error
  - EADDRINUSE (“Address already in use”)
- Example:

```
If (bind (sd, (struct sockaddr *) &servaddr, sizeof (servaddr)) != 0)
    errsys (“bind call error”);
```



`listen()`

`int listen(int sockfd, int backlog);`

Announce willingness to accept connections, give queue size, change socket state for TCP server.

- *sockfd* is socket descriptor from **`socket()`**
- *backlog* is maximum number of *incomplete* connections
  - historically 5
  - rarely above 15 on a even moderate web server!
- Sockets default to active (for client)
- Example:

```
If (listen (sd, 2) != 0)
    errsys (“listen call error”);
```

accept()

```
int accept(int sockfd, struct sockaddr cliaddr,
      socklen_t *addrlen);
```

Return next completed connection.

- *sockfd* is socket descriptor from **`socket()`**
- *cliaddr* and *addrlen* return protocol address from client
- returns brand new descriptor, created by OS
- if used with **`fork()`**, can create concurrent server (more later)
- Example:

```
sfd = accept (s, NULL, NULL);
if (sfd == -1) err_sys ("accept error");
```

`close()`

`int close(int sockfd);`

Close socket for use.

- *sockfd* is socket descriptor from **`socket()`**
- closes socket for reading/writing
  - returns (doesn't block)
  - attempts to send any unsent data
  - -1 if error

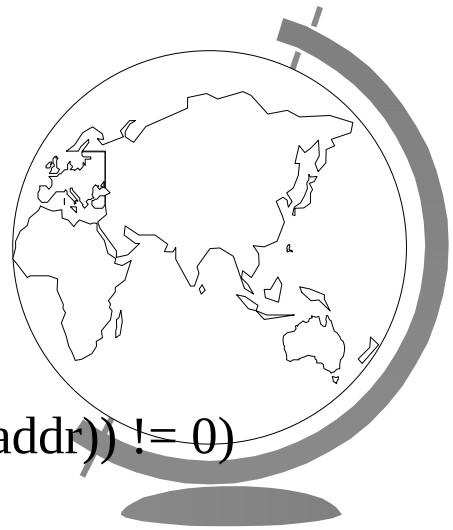
connect()

int connect(int *sockfd*, const struct sockaddr **servaddr*, socklen_t *addrlen*);

Connect to server.

- *sockfd* is socket descriptor from **socket()**
- *servaddr* is a pointer to a structure with:
  - Server *port number* and *IP address*
  - must be specified (unlike **bind()**)
- *addrlen* is length of structure
- client doesn't need **bind()**
  - OS will pick ephemeral port
- returns socket descriptor if ok, -1 on error
- Example:

```
if ( connect (sockfd, (struct sockaddr *) &servaddr, sizeof (servaddr)) != 0)
    err_sys("connect call error");
```



Sending and Receiving

```
int recv(int sockfd, void *buff,
  size_t mbytes, int flags);
int send(int sockfd, void *buff,
  size_t mbytes, int flags);
```

- Same as **`read()`** and **`write()`** but for *flags*
- *flags* examples (see man pages)
  - MSG_DONTWAIT (this send non-blocking)
  - MSG_OOB (out of band data, 1 byte sent ahead)
  - MSG_WAITALL (don't give me less than max)
  - MSG_DONTROUTE (bypass routing table)

Socket Options

- Many socket( ) options
- Set/lookup using
  - **`setsockopt(), getsockopt()`**
- Examples:
  - SO_LINGER
  - SO_RCVBUF, SO_SNDBUF (modify buffer sizes)
  - SO_RCVLOWAT, SO_SNDLOWAT
  - SO_RCVTIMEO, SO_SNDTIMEO (Timeouts)
  - TCP_KEEPALIVE (idle time before close (2 hours, default))
  - TCP_MAXRT (set timeout value)
  - TCP_NODELAY (disable *Nagle Algorithm)*
- See man pages for details
- > **`man socket`** on any Unix machine

Concurrent TCP Server

Text segment

```
sock = socket()
/* setup socket */
while (1) {
      newsock = accept(sock)
      fork()
      if child
            read(newsock)
            until exit

}
```

Parent

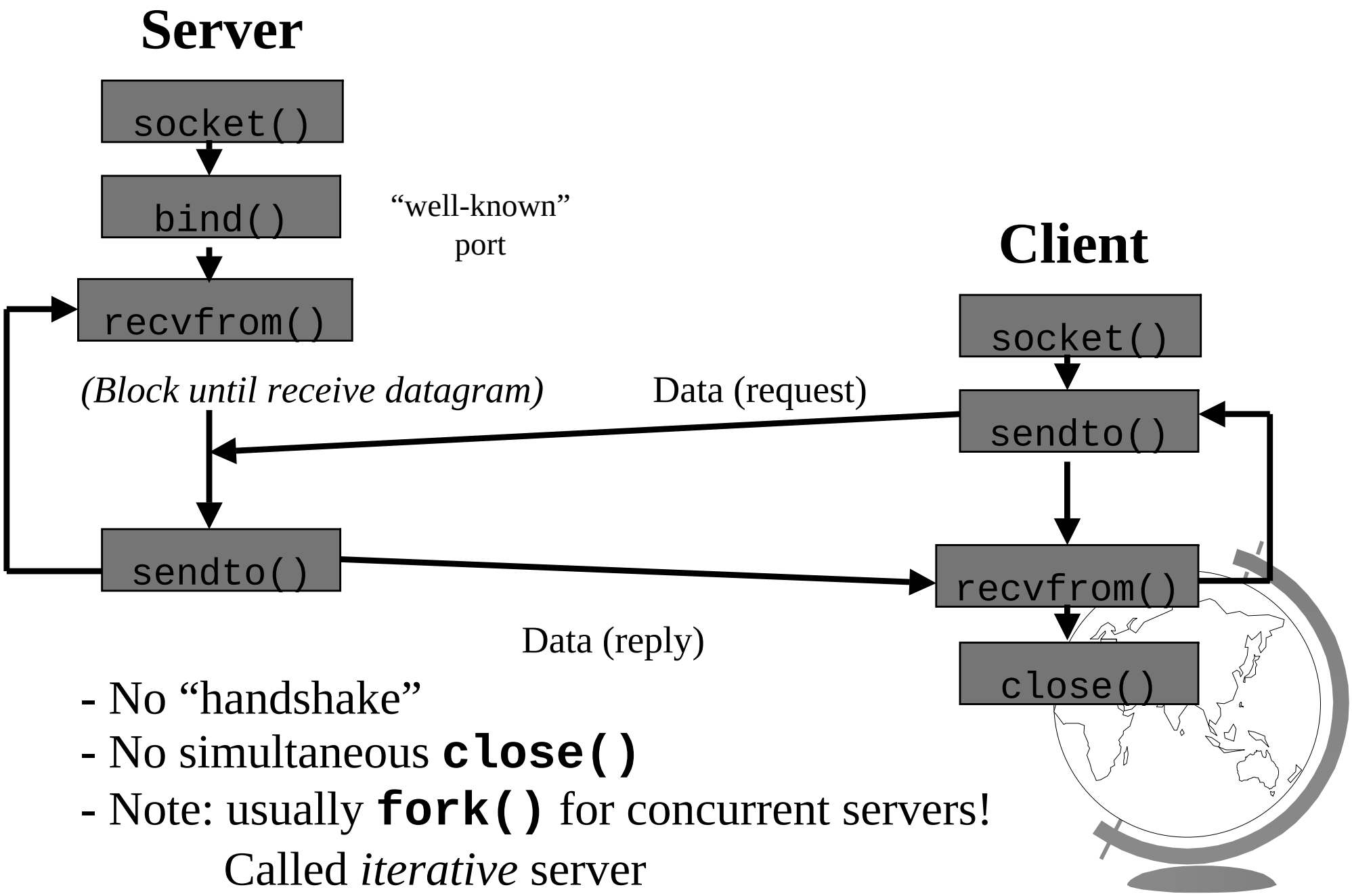
```
int sock;
int newsock;
```

Child

```
int sock;
int newsock;
```

- Close **sock** in child, **newsock** in parent
- Reference count for socket descriptor

UDP Client-Server



- No "handshake"
- No simultaneous **`close()`**
- Note: usually **`fork()`** for concurrent servers!

  Called *iterative* server

Sending and Receiving (UDP)

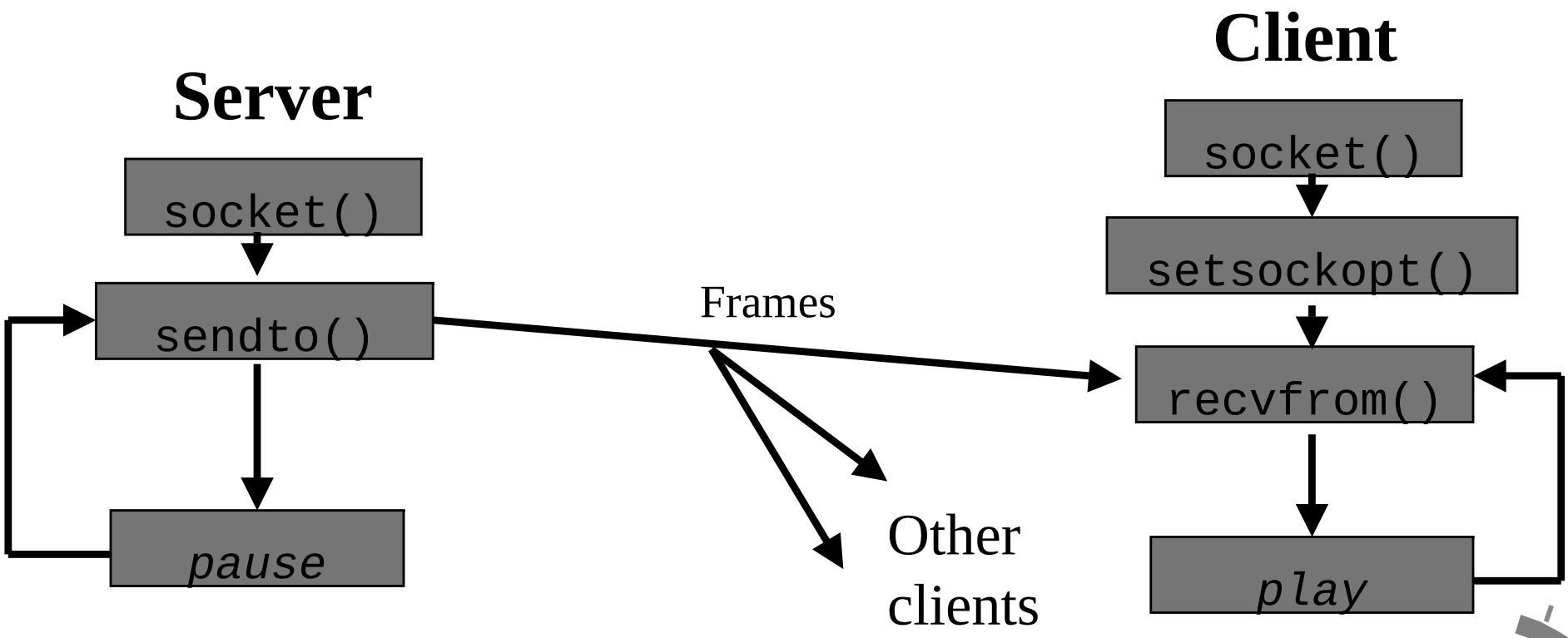
```
int recvfrom(int sockfd, void *buff, size_t mbytes, int
   flags, struct sockaddr *from, socklen_t *addrlen);
int sendto(int sockfd, void *buff, size_t mbytes, int
   flags, const struct sockaddr *to, socklen_t addrlen);
```

- Same as **`recv()`** and **`send()`** but for *addr*
  - **`recvfrom`** fills in address of where packet came from
  - **`sendto`** requires address of where sending packet to

`connect()` with UDP

- Record address and port of peer
  - datagrams to/from others are not allowed
  - does not do three way handshake, or connection
  - connect a misnomer, here. Should be **`setpeername()`**
- Use **`send()`** instead of **`sendto()`**
- Use **`recv()`** instead of **`recvfrom()`**
- Can change connect (or unconnect) by repeating **`connect()`** call

Mcast Extensions to UDP



- **`sendto()`** in server to multicast group
- **`setsockopt()`** in client to join group
  `IP_ADD_MEMBERSHIP`