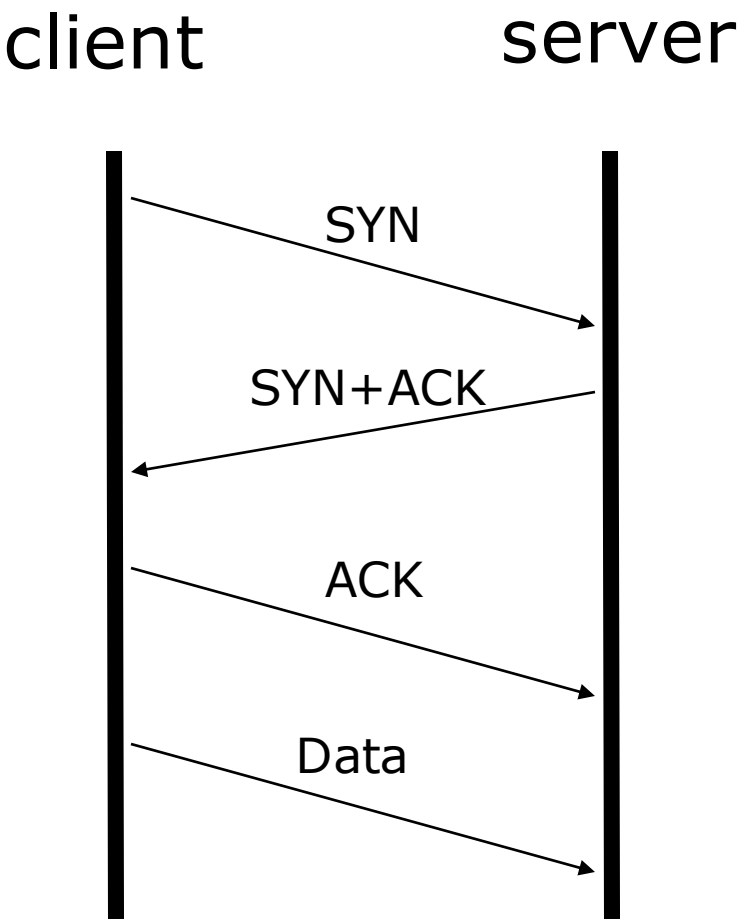
# TCP SOCKETS

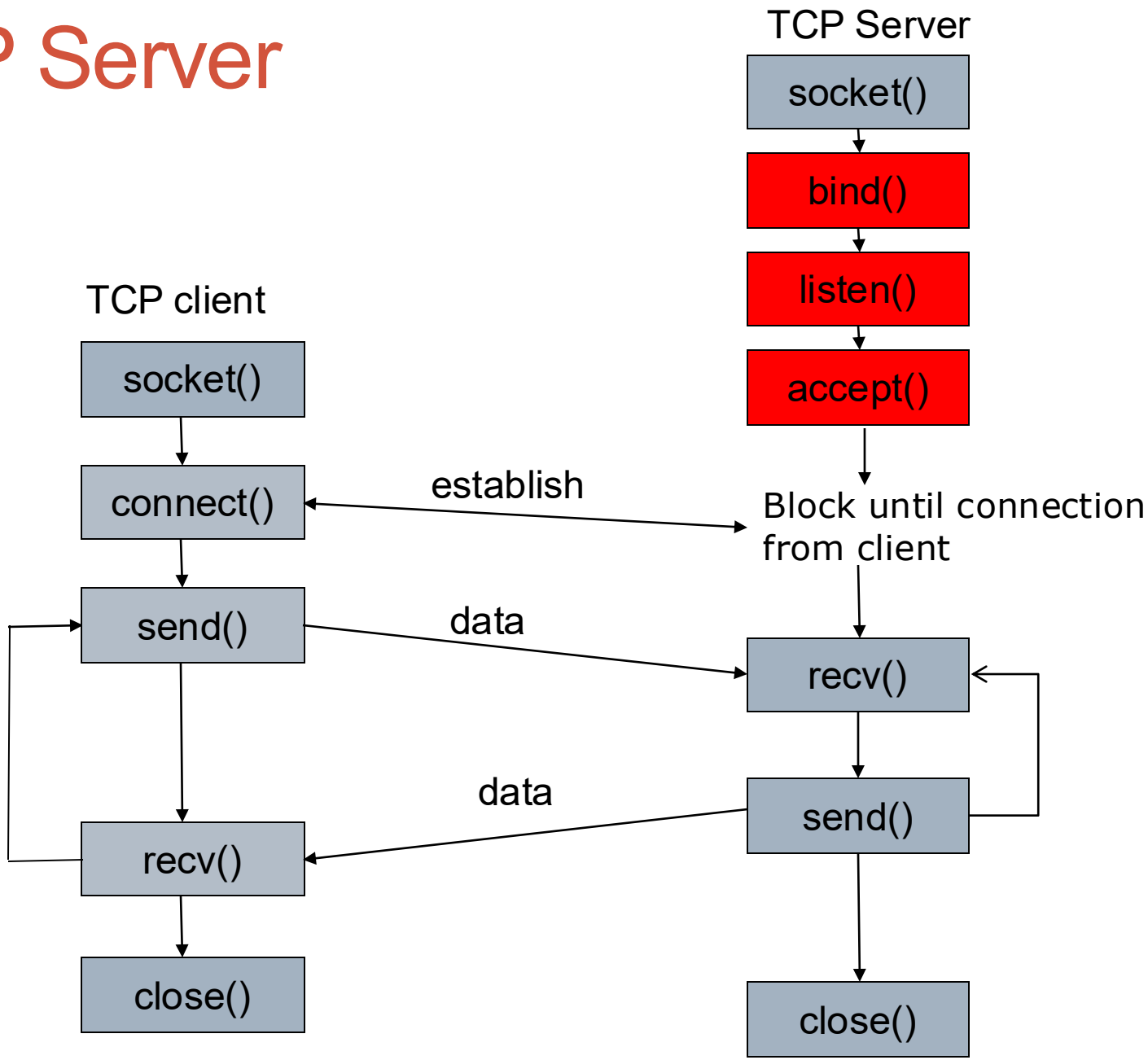
# TCP (Transmission Control Protocol)

- Provide reliable communication
- Data rate control
- Example
  - Mail
  - WEB
  - Image



# TCP SERVER

# TCP Server


TCP Server
socket()
bind()
listen()
accept()
Block until connection from client
TCP client
socket()
connect()
establish
send()
data
recv()
data
send()
recv()
close()
close()

# TCP server side

1. Create a socket – socket().
2. Bind the socket – bind().
3. Listen on the socket – listen().
4. Accept a connection – accept().
5. Send and receive data – recv(), send().
6. Disconnect connection– close()
7. Close LISTENING socket

repeatedly

# Socket Mode

- Types of server sockets
  - *Iterating server*: Only one socket is opened at a time.
  - *Concurent server*: After an accept, a child process/thread is spawned to handle the connection.
  - *Multiplexing server*: use select to simultaneously wait on all open socketIds, and waking up the process only when new data arrives

# socket()

```
#include <sys/types.h>
#include <sys/socket.h>
int socket(int domain, int type, int protocol);
```

- Creates an endpoint for communication
- [IN] `domain`: AF_INET, AF_INET6, …
- [IN] `type` argument can be:
  - SOCK_STREAM: Provides sequenced, reliable, two-way, connection-based byte streams
  - SOCK_DGRAM: Supports datagrams
  - SOCK_RAW: Provides raw network protocol access
- [IN] `protocol` is usually 0
- Returns value
  - A new socket descriptor that you can use to do sockety things with
  - If error occurs, return -1

# bind()

```
#include <sys/types.h>
#include <sys/socket.h>
int bind(int sockfd, const struct sockaddr *addr,
                                  socklen_t addrlen);
```

- Associate a socket with an IP address and port number
- Where
  - [IN] `sockfd` : socket descriptor
  - [IN] `addr` : pointer to a `sockaddr` structure assigned to `sockfd`
  - [IN] `addrlen` : specifies the size, in bytes of address structure pointed to by `addr`
- Return value
  - Returns 0 if no error occurs.
  - Otherwise, return -1 (and **errno** will be set accordingly)

# listen()

```
#include <sys/socket.h>
int listen(int sockfd, int backlog);
```

- Establish a socket to LISTENING for incoming connection.
- Parameters:
  - [IN] `sockfd`: a descriptor identifying a bound, unconnected socket
  - [IN] `backlog`: the number of pending connections the queue will hold
- Return value
  - On success, 0 is returned
  - On error, -1 is returned

# accept()

```
#include <sys/types.h>
#include <sys/socket.h>
int accept(int sockfd, struct sockaddr *addr,socklen_t *addrlen);
```

- Accept an incoming connection on a LISTENING socket
- Parameters:
  - `sockfd`: A descriptor identifying a socket which is listening for connections after a `listen()`.
  - `addr`: pointer to a `sockaddr` structure filled in with the address of the peer socket
  - `addrlen`: return the actual size of the peer address.
- Return value
  - Newly connected socket descriptor if no errors
  - -1 if has errors

# send()

```
#include <sys/types.h>
#include <sys/socket.h>
ssize_t send(int sockfd, const void *buf, size_t len,
             int flags);
```

- Send data on a connected socket
- Parameter:
  - [IN] `sockfd`: a descriptor identifying a connected socket.
  - [IN] `buf`: points to the buffer containing the message to send.
  - [IN] `len`: specifies the length of the message
  - [IN] `flags`: specifies the type of message transmission, usually 0
- Return value:
  - If no error occurs, send() returns the total number of characters sent
  - Otherwise, return -1

# `send()` - Flags

- `MSG_OOB`: Send as “out of band” data. The receiver will receive the signal SIGURG and it can then receive this data without first receiving all the rest of the normal data in the queue.
- `MSG_DONTROUTE` :Don't send this data over a router, just keep it local.
- `MSG_DONTWAIT`: If **send()** would block because outbound traffic is clogged, have it return EAGAIN.
- `MSG_NOSIGNAL`: If you **send()** to a remote host which is no longer **recv()**, you'll typically get the signal SIGPIPE. Adding this flag prevents that signal from being raised.

# send()

```
char sendBuff[2048];
int  dataLength, nLeft, idx;

// Fill sendbuff with 2048 bytes of data
nLeft = dataLength;
idx = 0;

while (nLeft > 0){
    // Assume s is a valid, connected stream socket
    ret = send(s, &sendBuff[idx], nLeft, 0);
    if (ret == -1)
    {
        // Error handler
    }
    nLeft -= ret;
    idx += ret;
}
```

# recv()

```
#include <sys/types.h>
#include <sys/socket.h>
ssize_t recv(int sockfd, void *buf, size_t len, int flags);
```

- Receive data on a socket
- Parameter:
  - [IN] `sockfd`: a descriptor identifying a connected socket.
  - [IN, OUT] `buf`: points to a buffer where the message should be stored
  - [IN] `len`: specifies the length in bytes of the buffer
  - [IN] `flags`: specifies the type of message reception, usually 0
- Return value:
  - If no error occurs, returns the length of received message in bytes
  - If peer has performed an orderly shutdown, return 0
  - Otherwise, return -1

# receive()

```
char     recvBuff[1024];
int      ret, nLeft, idx;
nLeft = dataLength; //length of the data needs to be
                                    //received
idx = 0;

while (nLeft > 0)
{
    ret = recv(s, &recvBuff[idx], nLeft, 0);
    if (ret == -1)
    {
        // Error handler
    }
    idx += ret;
    nLeft -= ret;
}
```

# `recv()` - Flags

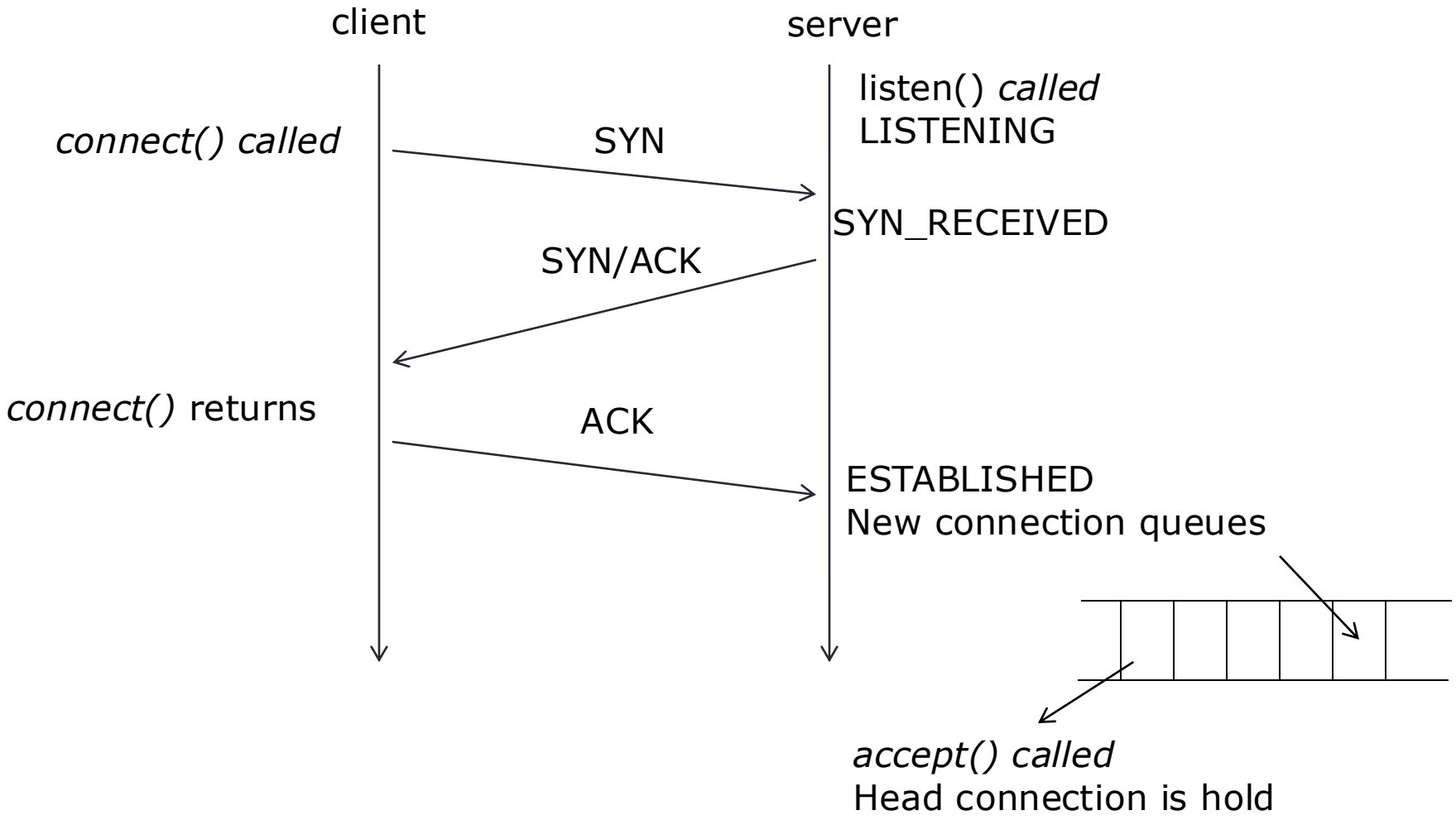
- `MSG_PEEK`: Peeks at an incoming message. The data is treated as unread and the next *recvfrom*() or similar function shall still return this data.
- `MSG_OOB`: Requests out-of-band data. The significance and semantics of out-of-band data are protocol-specific.
- `MSG_WAITALL`: On SOCK_STREAM sockets this requests that the function block until the full amount of data can be returned, excepting:
  - the connection is terminated
  - MSG_PEEK was specified
  - an error is pending for the socket
  - a signal is caught
- Use bitwise OR operator (|) to combine more than one flag

# close()

```
#include <unistd.h>
int close(int sockfd);
```

- Close a socket descriptor
- [IN] `sockfd`: a descriptor identifying a socket.
- Return value
  - Returns 0 if no error occurs.
  - Otherwise, return -1 (and **errno** will be set accordingly)
- `close()` **vs** `shutdown()`
  - `close()` tries to complete this transmission before closing, frees the socket descriptor
  - `shutdown()`: immediately stops receiving and transmitting data, don't releases the socket descriptor

# Process connections



# Example

```
int listenfd, connfd, n;
pid_t childpid;
 socklen_t clilen;
 char buf[MAXLINE];
 struct sockaddr_in cliaddr, servaddr;
listenfd = socket (AF_INET, SOCK_STREAM, 0);

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

bind (listenfd, (struct sockaddr *) &servaddr, sizeof(servaddr));

listen (listenfd, LISTENQ);
printf("%s\n","Server running...waiting for connections.");
```

*creation of server socket*

*Preparation of the socket address struct*

*Bind the socket to the port in address*

*Listen for connection to the socket*

# Example (Cont.)

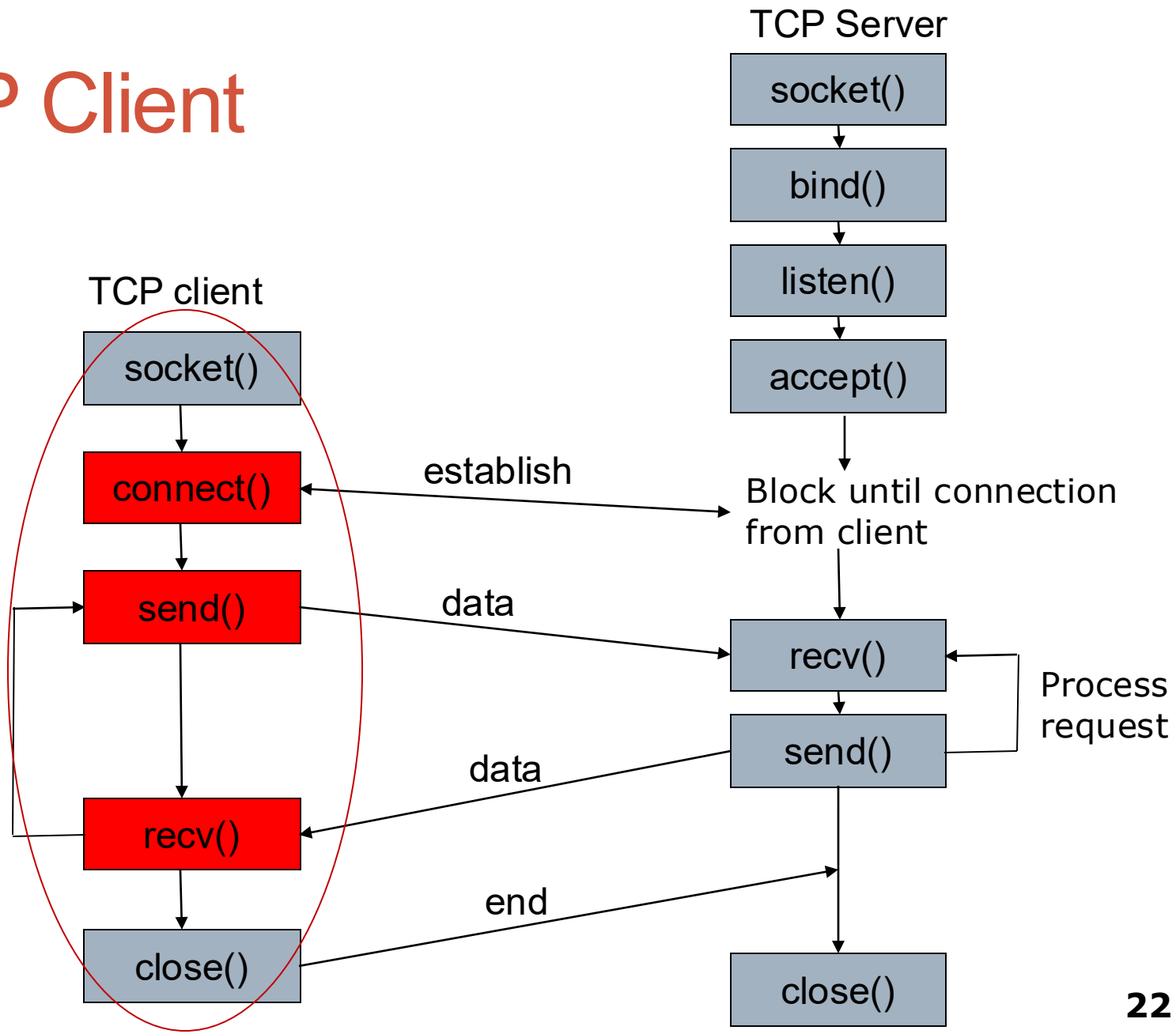
```
for ( ; ; ) {
 clilen = sizeof(cliaddr);
 connfd = accept (listenfd, (struct sockaddr *) &cliaddr, &clilen);
 printf("%s\n","Received request...");
 while ( (n = recv(connfd, buf, MAXLINE,0)) > 0)  {
  printf("%s","String received from and resent to the client:");
  puts(buf);
  send(connfd, buf, n, 0);
 }
 if (n < 0) {
  perror("Read error");
  exit(1);
 }
 close(connfd); // close the file descriptor.
}
close (listenfd); //close listening socket
```

*Accept a connection request → return a File Descriptor (FD)*

*Send and receive data from the FD*

# TCP CLIENT

# TCP Client


TCP Server
socket()
bind()
listen()
accept()
Block until connection from client
recv()
send()
Process request
TCP client
socket()
connect()
send()
recv()
close()
establish
data
data
end
close()

# TCP client side

- The typical TCP client's communication involves four basic steps:
  - Create a TCP socket using socket().
  - Establish a connection to the server using connect().
  - Communicate using send() and recv().
  - Close the connection with close().
- Why "clients" doesn't need bind() ?

# connect()

```
#include <sys/types.h>
#include <sys/socket.h>
int connect(int sockfd, const struct sockaddr *serv_addr,
            socklen_t addrlen);
```

- Connect a socket to a server
- Parameters:
  - [IN] `sockfd`: A descriptor identifying an unconnected socket.
  - [IN] `serv_addr`: The address of the server to which the socket is to be connected.
  - [IN] `addrlen`: The length of the name.
- Return value
  - If no error occurs, returns 0.
  - Otherwise, it returns -1

# send(), receive()

- Similar in TCP server

# Example

```
int main(int argc, char **argv)
{ int sockfd;
  struct sockaddr_in servaddr;
 char sendline[MAXLINE], recvline[MAXLINE];
//Create a socket for the client
if ((sockfd = socket (AF_INET, SOCK_STREAM, 0)) <0) {
  perror("Problem in creating the socket");
  exit(2);
 }
 //Creation of the remote server socket information structure
 memset(&servaddr, 0, sizeof(servaddr));
 servaddr.sin_family = AF_INET;
 servaddr.sin_addr.s_addr= inet_addr(argv[1]);
 servaddr.sin_port =  htons(SERV_PORT); //convert to big-endian order
```

create a client socket

create a socket addr info pointing to server socket

# Example (Cont.)

```
// Connect the client to the server socket
if (connect(sockfd, (struct sockaddr *) &servaddr, sizeof(servaddr))<0) {
 perror("Problem in connecting to the server");
 exit(3);
}

while (fgets(sendline, MAXLINE, stdin) != NULL) {
send(sockfd, sendline, strlen(sendline), 0);

 if (recv(sockfd, recvline, MAXLINE,0) == 0){
  //error: server terminated prematurely
  perror("The server terminated prematurely");
  exit(4);
 }
```

Connect the client socket with remote server

Send and receive data from client socket

# OTHERS

# shutdown()

```
#include <sys/socket.h>
int shutdown(int socket, int how);
```

- Shut down socket send and receive operations
- Where
  - [IN] `sockfd`: a descriptor identifying a socket.
  - [IN] `how`: SHUT_RD, SHUT_WR, SHUT_RDWR
- Return value
  - Returns 0 if no error occurs.
  - Otherwise, return -1

# Socket options

```
#include <sys/socket.h>
int setsockopt (int sockfd, int level, int optname,
                void *optval, int optlen);
```

- Set the options that control the transfering data on a socket
- Parameters:
  - [IN] `sockfd`: refer to an open socket descriptor
  - [IN] `level`: specifies the protocol level at which the option resides
  - [IN] `optname`: specifies a single option to set
  - [IN] `optval`: points to the setted option value
  - [IN] `optlen`: the size of option value pointed by `optval`
- Return:
  - Returns 0 if no error occurs.
  - Otherwise, return -1 (and **errno** will be set accordingly)

# Socket options(cont)

```
#include <sys/socket.h>
int getsockopt (int sockfd, int level, int optname,
                void *optval, int *optlen);
```

- Set the options that control the transfering data on a socket
- Parameters:
  - [IN] `sockfd`: refer to an open socket descriptor
  - [IN] `level`: specifies the protocol level at which the option resides
  - [IN] `optname`: specifies a single option to set
  - [OUT] `optval`: points to the setted option value
  - [IN, OUT] `optlen`: the size of option value pointed by `optval`
- Return:
  - Returns 0 if no error occurs.
  - Otherwise, return -1 (and **errno** will be set accordingly)

# level = SOL_SOCKET

| Value name | Type | Description |
|---|---|---|
| SO_BROADCAST | int | Configures a socket for sending broadcast data.(Only UDP socket) |
| SO_DONTROUTE | int | Sets whether outgoing data should be sent on interface the socket is bound to and not a routed on some other interface |
| SO_KEEPALIVE | int | TCP automatically sends a keep-alive probe to the peer |
| SO_LINGER | linger | specifies how the close function operates for a connection-oriented protocol |
| SO_REUSEADDR | int | Allows the socket to be bound to an address that is already in use |
| SO_RCVTIMEO | timeval | Sets the timeout for blocking receive calls |
| SO_SNDTIMEO | timeval | Sets the timeout for blocking send calls |