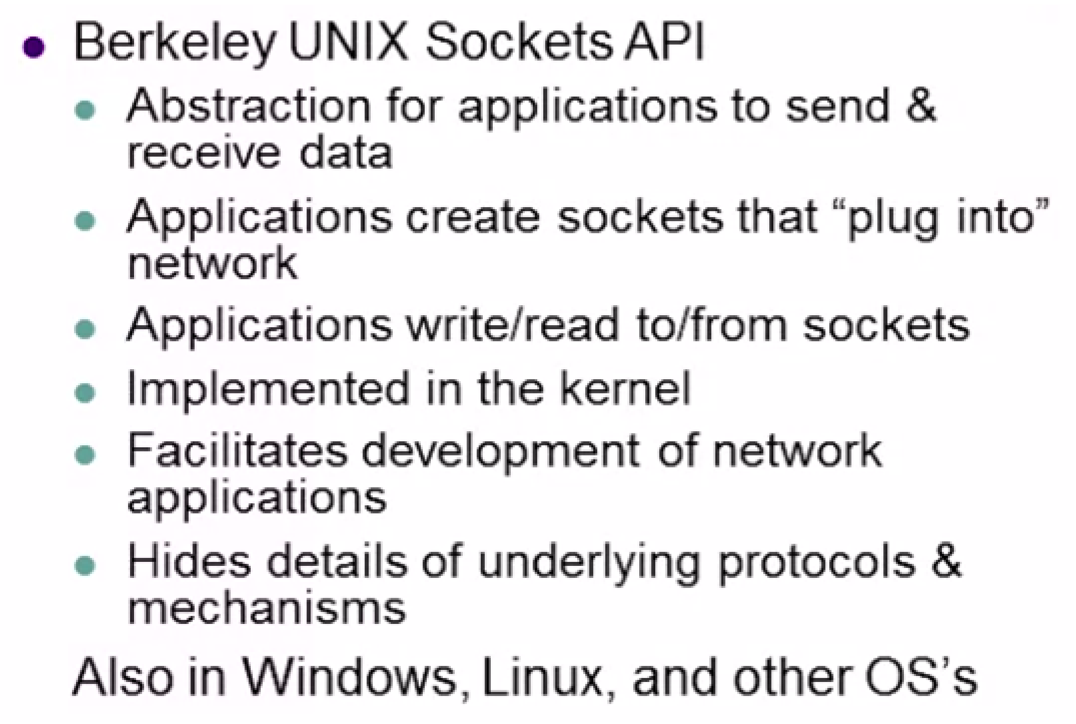
Socket API

Go Back

Socket API

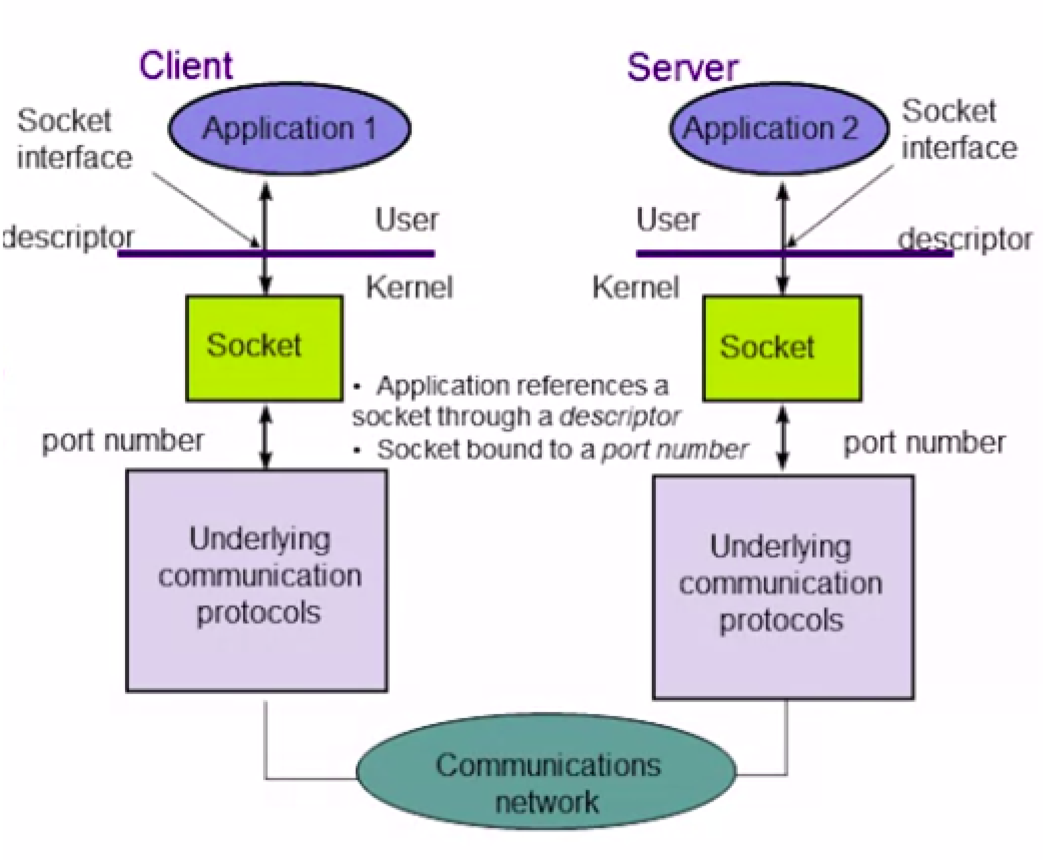
Berkeley Socket API



- Berkeley UNIX Sockets API
  - Abstraction for applications to send & receive data
  - Applications create sockets that "plug into" network
  - Applications write/read to/from sockets
  - Implemented in the kernel
  - Facilitates development of network applications
  - Hides details of underlying protocols & mechanisms

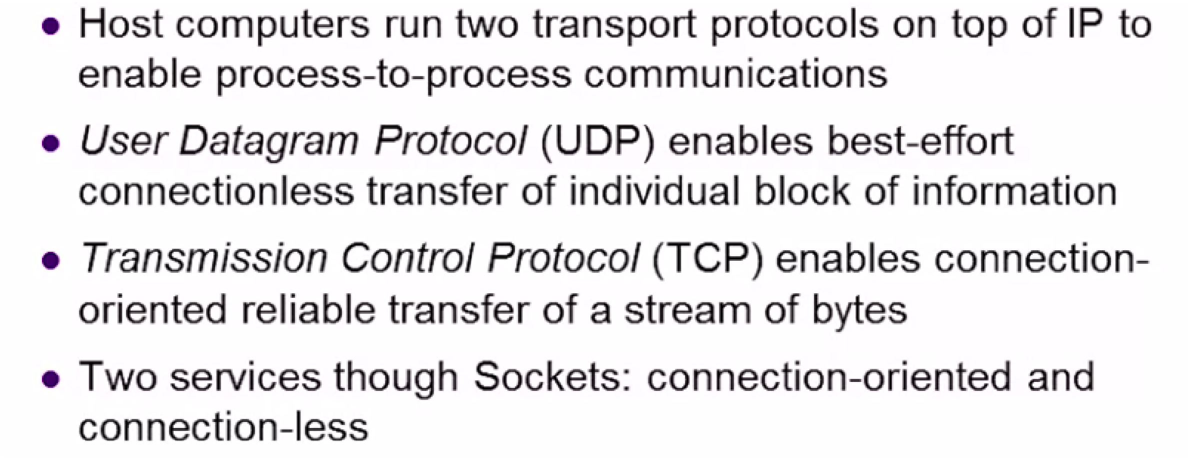
  Also in Windows, Linux, and other OS's

Communication through Socket Layer



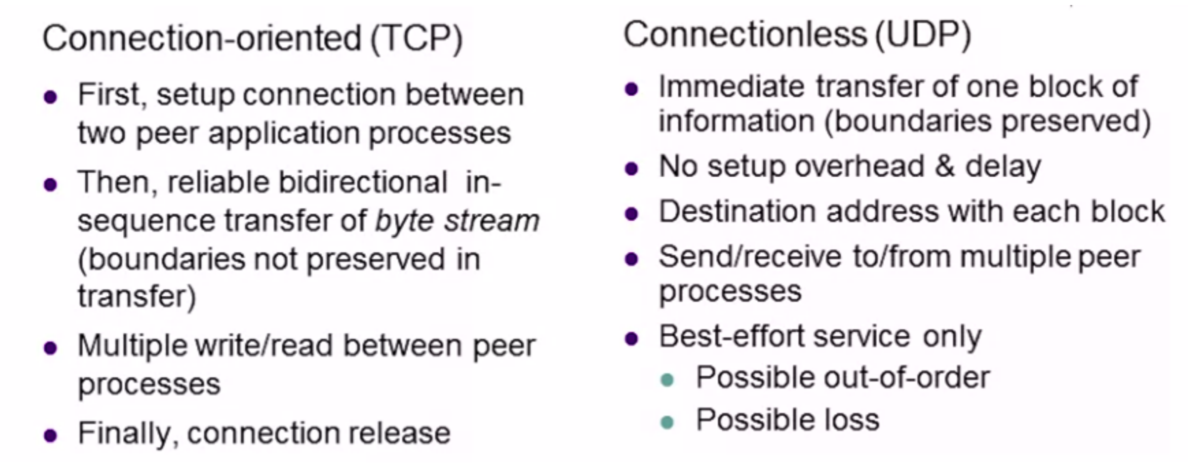
- Application references a socket through a *descriptor*
- Socket bound to a *port number*

Transport Protocols



- Host computers run two transport protocols on top of IP to enable process-to-process communications
- *User Datagram Protocol* (UDP) enables best-effort connectionless transfer of individual block of information
- *Transmission Control Protocol* (TCP) enables connection-oriented reliable transfer of a stream of bytes
- Two services though Sockets: connection-oriented and connection-less

Stream Mode of Service



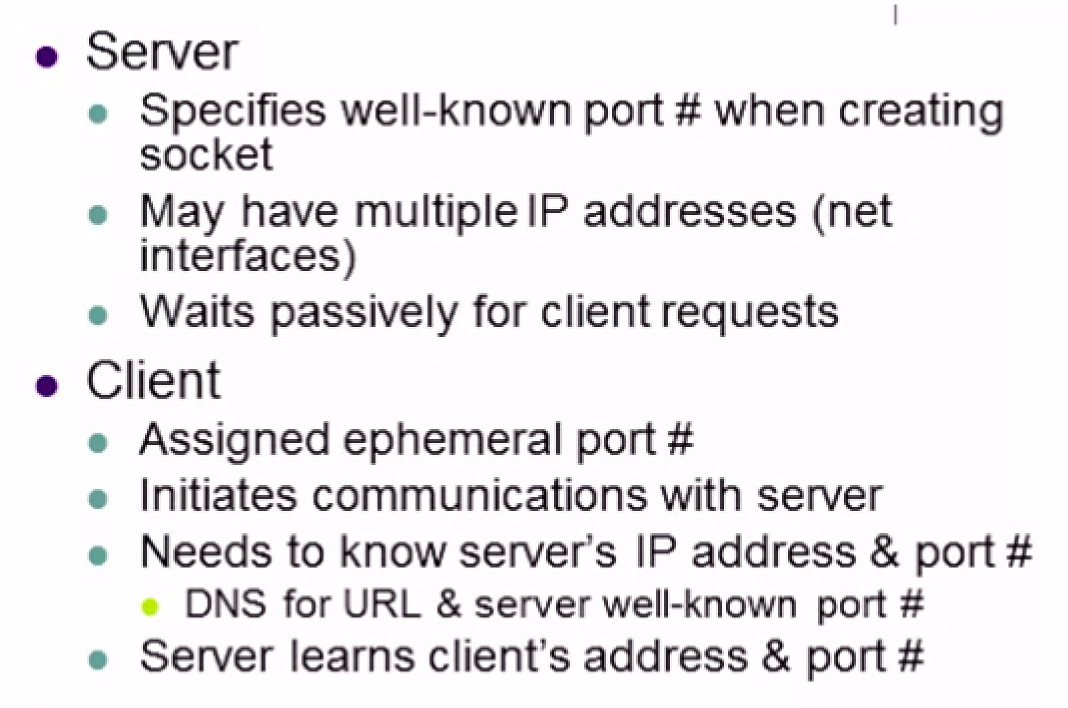
Connection-oriented (TCP)

- First, setup connection between two peer application processes
- Then, reliable bidirectional in-sequence transfer of *byte stream* (boundaries not preserved in transfer)
- Multiple write/read between peer processes
- Finally, connection release

Connectionless (UDP)

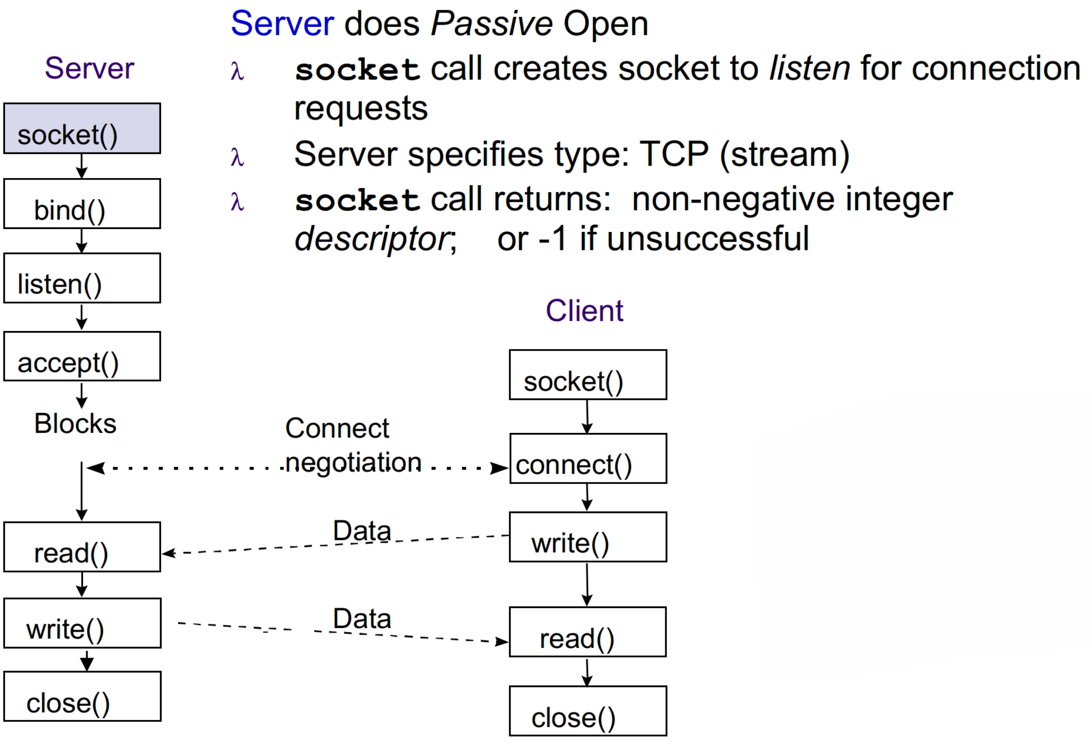
- Immediate transfer of one block of information (boundaries preserved)
- No setup overhead & delay
- Destination address with each block
- Send/receive to/from multiple peer processes
- Best-effort service only
  - Possible out-of-order
  - Possible loss

Client & Server Differences



- Server
  - Specifies well-known port # when creating socket
  - May have multiple IP addresses (net interfaces)
  - Waits passively for client requests
- Client
  - Assigned ephemeral port #
  - Initiates communications with server
  - Needs to know server's IP address & port #
    - DNS for URL & server well-known port #
  - Server learns client's address & port #

Inner Working of Socket Calls for Connection-Oriented Mode



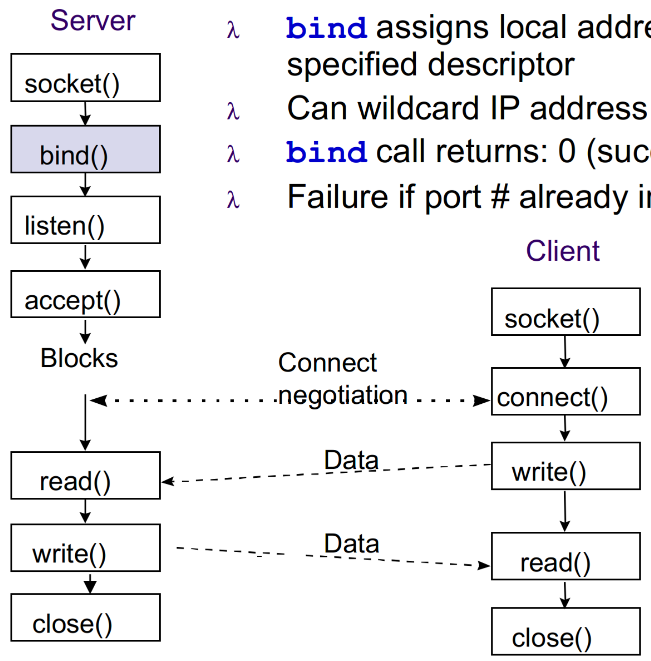
Server does *Passive* Open

λ **`socket`** call creates socket to *listen* for connection requests

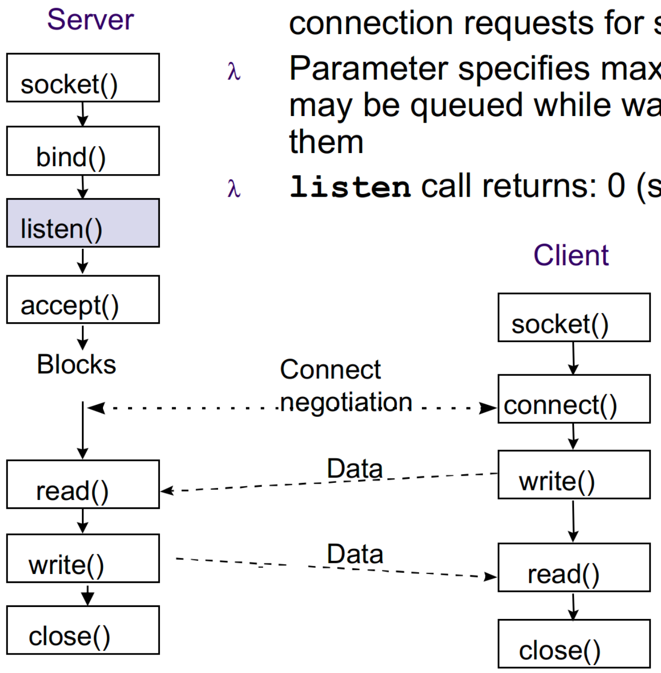
λ Server specifies type: TCP (stream)

λ **`socket`** call returns: non-negative integer *descriptor*; or -1 if unsuccessful

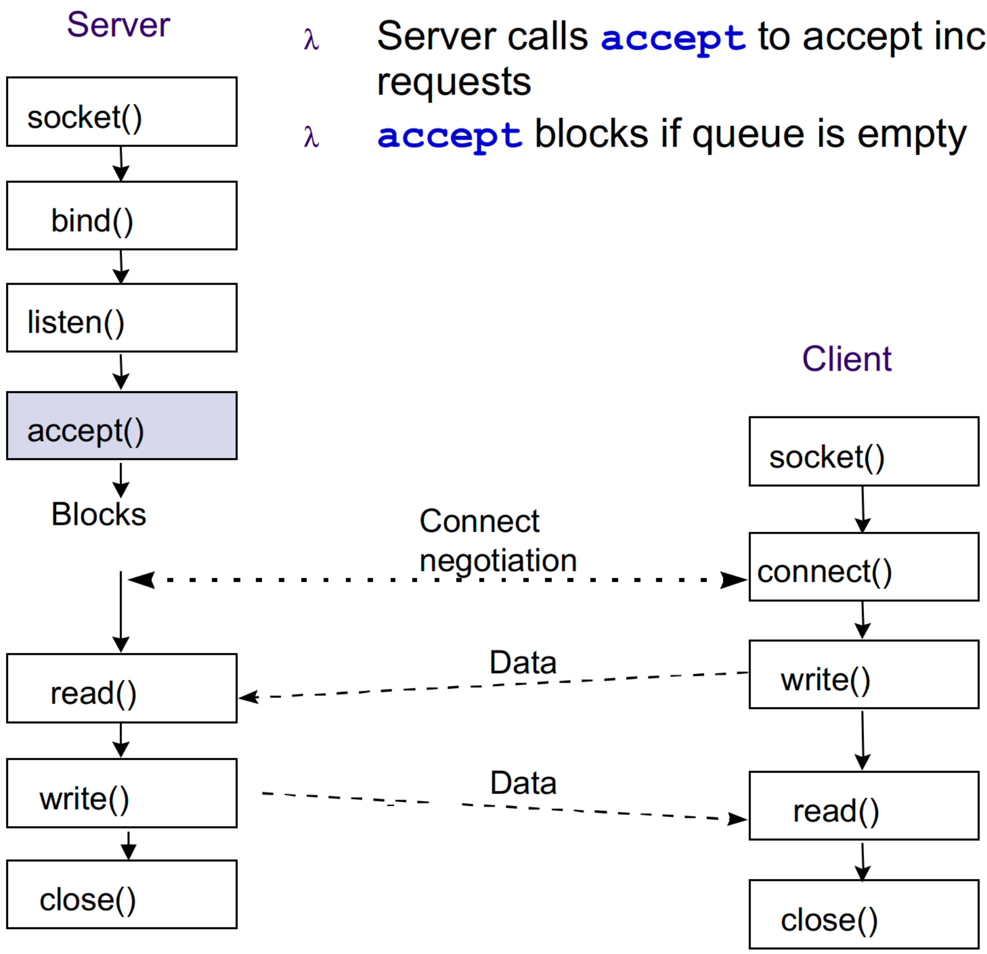
- **bind** assigns local address & port # to socket with specified descriptor
- Can wildcard IP address for multiple net interfaces
- **bind** call returns: 0 (success); or -1 (failure)
- Failure if port # already in use or if reuse option not set



- **listen** indicates to TCP readiness to receive connection requests for socket with given descriptor
- Parameter specifies max number of requests that may be queued while waiting for server to accept them
- **listen** call returns: 0 (success); or -1 (failure)



- Server calls `accept` to accept incoming requests
- `accept` blocks if queue is empty



Server

socket()

bind()

listen()

accept()

Blocks

Connect negotiation

read()

write()

close()

Client

socket()

connect()

write()

Data

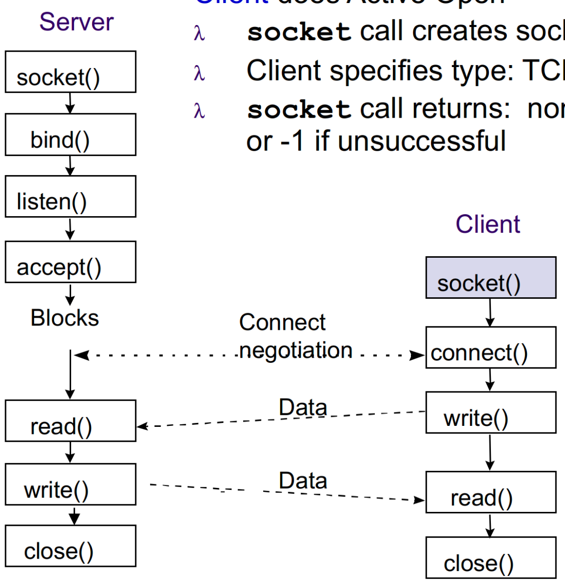
read()

Data

close()

Client does Active Open

- `socket` call creates socket to connect to server
- Client specifies type: TCP (stream)
- `socket` call returns: non-negative integer *descriptor*; or -1 if unsuccessful



Server

socket()

bind()

listen()

accept()

Blocks

Connect negotiation

read()

write()

close()

Client

socket()

connect()

write()

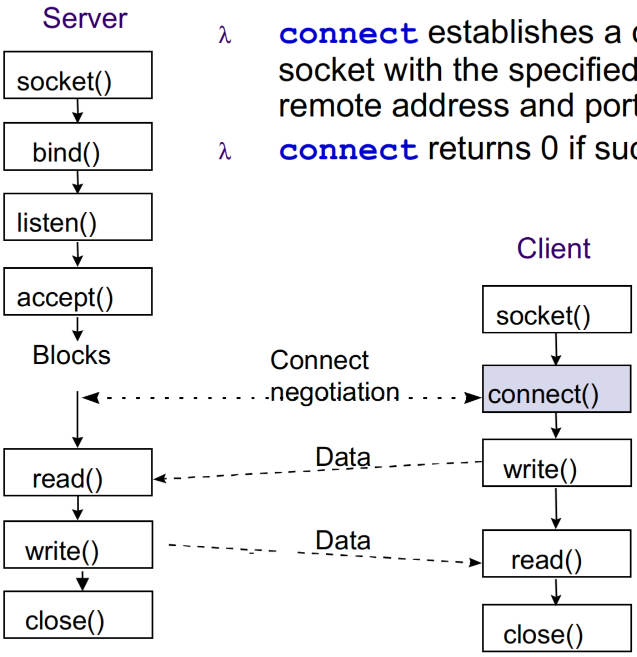
Data

read()

Data

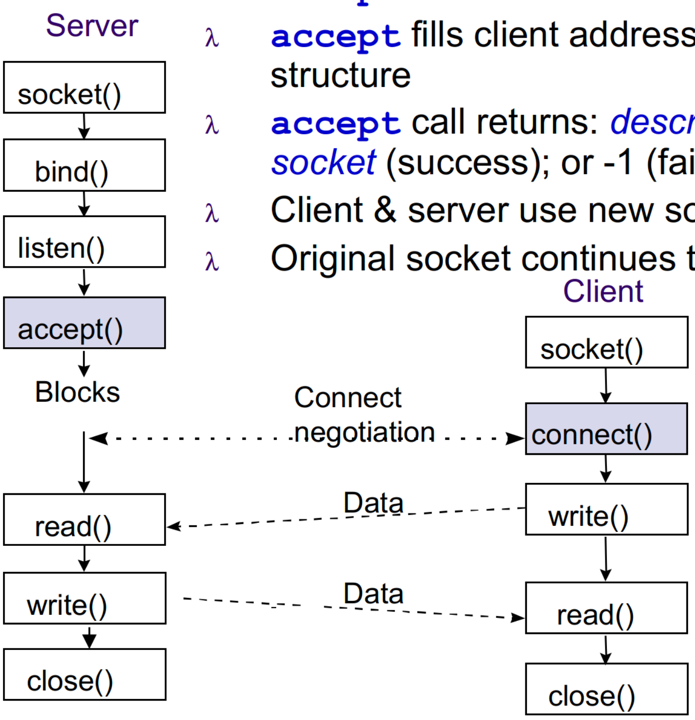
close()

- **connect** establishes a connection on the local socket with the specified descriptor to the specified remote address and port #
- **connect** returns 0 if successful; -1 if unsuccessful



Note: **connect** initiates TCP three-way handshake

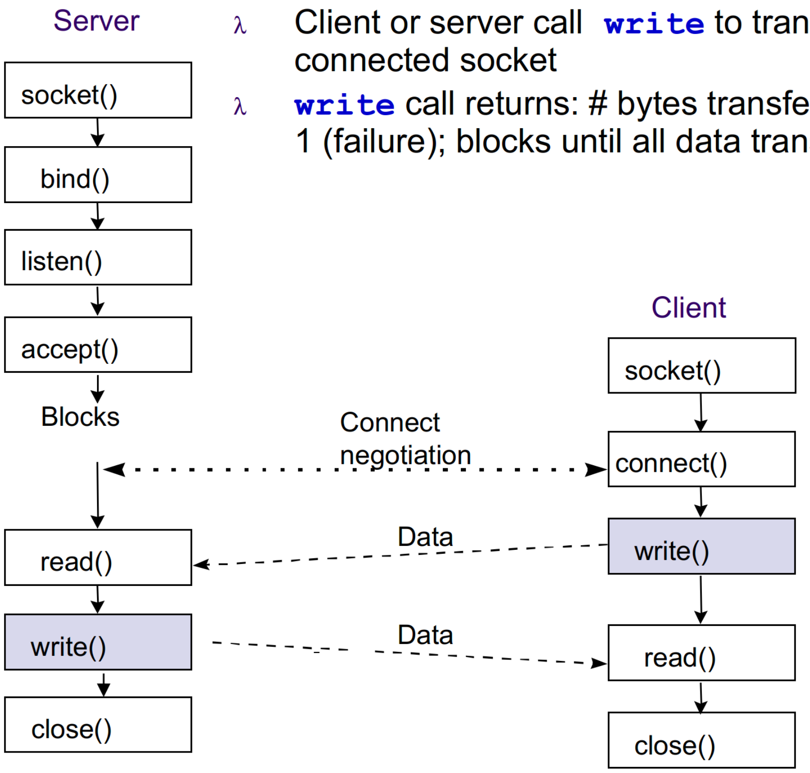
- **accept** wakes with incoming connection request
- **accept** fills client address & port # into address structure
- **accept** call returns: *descriptor of* ***new*** *connection socket* (success); or -1 (failure)
- Client & server use new socket for data transfer
- Original socket continues to listen for new requests



Data Transfer

λ Client or server call `write` to transmit data into a connected socket

λ `write` call returns: # bytes transferred (success); or -1 (failure); blocks until all data transferred

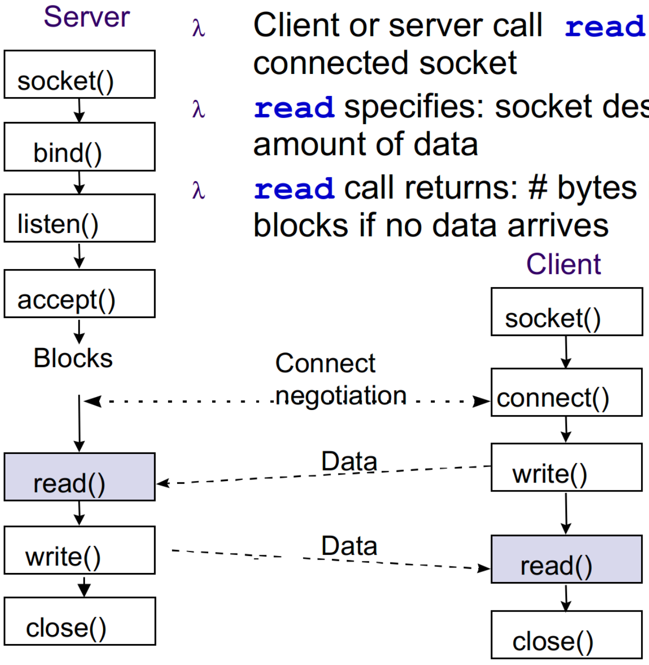


Data Transfer

λ Client or server call `read` to receive data from a connected socket

λ `read` specifies: socket descriptor; pointer to a buffer; amount of data

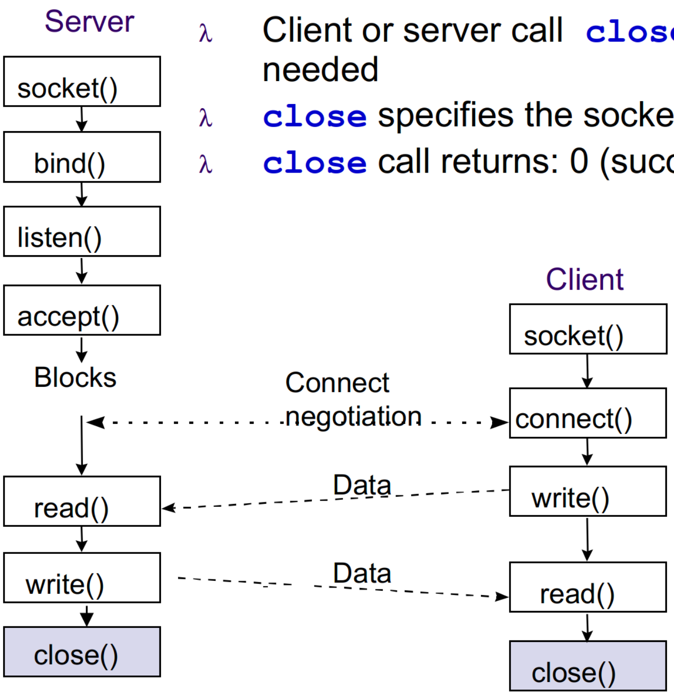
λ `read` call returns: # bytes read (success); or -1 (failure); blocks if no data arrives



Note: `write` and `read` can be called multiple times to transfer byte streams in both directions

Connection Termination

- Client or server call `close` when socket is no longer needed
- `close` specifies the socket descriptor
- `close` call returns: 0 (success); or -1 (failure)



Note: `close` initiates TCP graceful close sequence