Network

Network Transport

Socket Programming

Mercy College
CYBERSECURITY
Dr. John Yoon

IASP 505

Network

Reference

- Learn the structure or the format of network packets
  - Visit: www.networksorcery.com for IP
- Coding for unpacking the packets according to the packet format
  - Refer to: https://docs.python.org/3.6/library/struct.html#

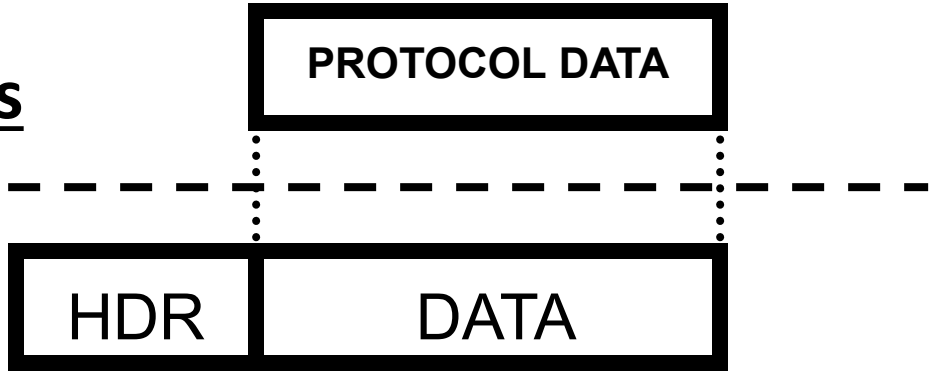
Inter-Layer Relationships

Packing for packets

- Each layer uses the layer below
  - The lower layer adds headers to the data from the upper layer
  - The data from the upper layer can also be a header on data from the layer above …

Unpacking for packets

- Backward3



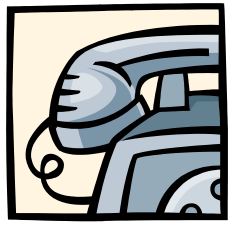
IP Characteristics

- Datagram-based
  - Connectionless
- Unreliable
  - Best efforts delivery
  - No delivery guarantees
- Logical (32-bit) addresses
  - Unrelated to physical addressing
  - Leading bits determine network membership

UDP Characteristics

- Also datagram-based
  - Connectionless, unreliable, can broadcast
- Applications usually message-based
  - No transport-layer retries
  - Applications handle (or ignore) errors
- Processes identified by port number
- Services live at specific ports
  - Usually below 1024, requiring privilege

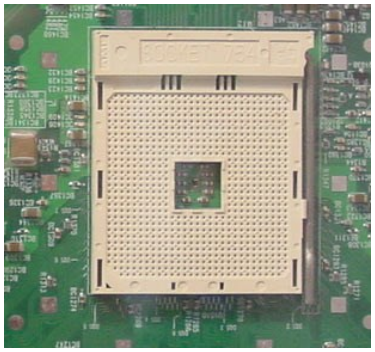
TCP Characteristics



- Connection-oriented
  - Two endpoints of a virtual circuit
- Reliable
  - Application needs no error checking
- Stream-based
  - No predefined blocksize
- Processes identified by port numbers
- Services live at specific ports

Sockets

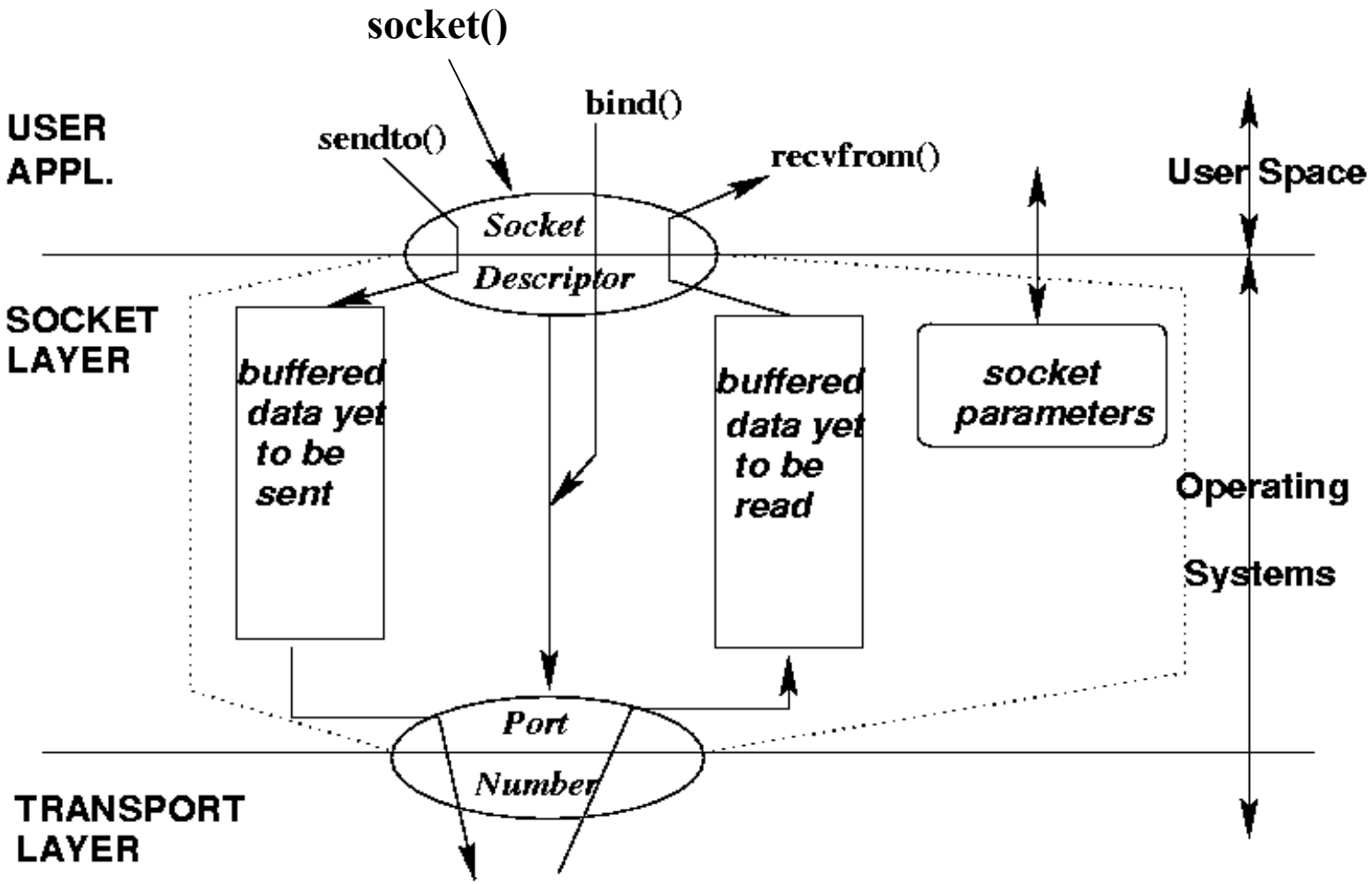
- Various sockets… Any similarity?



- Endpoint of a connection
  - Identified by IP address and Port number

- Primitive to implement high-level networking interfaces
  - e.g., Remote procedure call (RPC)

Socket: Conceptual View



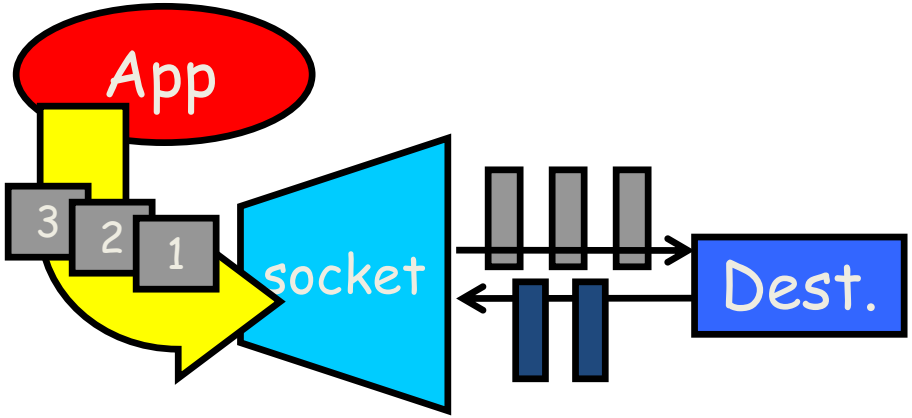
What is a socket?

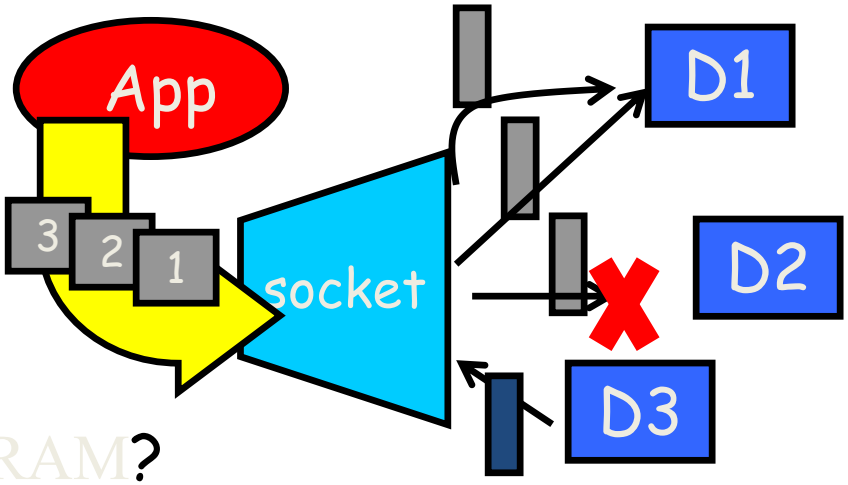
- An interface between application and network
  - The application creates a socket
  - The socket *type* dictates the style of communication
    - reliable vs. best effort
    - connection-oriented vs. connectionless
- Once configured the application can
  - pass data to the socket for network transmission
  - receive data from the socket (transmitted through the network by some other host)

Two essential types of sockets

- SOCK_STREAM
  - a.k.a. TCP
  - reliable delivery
  - in-order guaranteed
  - connection-oriented
  - bidirectional

- SOCK_DGRAM
  - a.k.a. UDP
  - unreliable delivery
  - no order guarantees
  - no notion of "connection" – app indicates dest. for each packet
  - can send or receive





Q: why have type SOCK_DGRAM?

A Socket-eye view of the Internet

cysecure.org

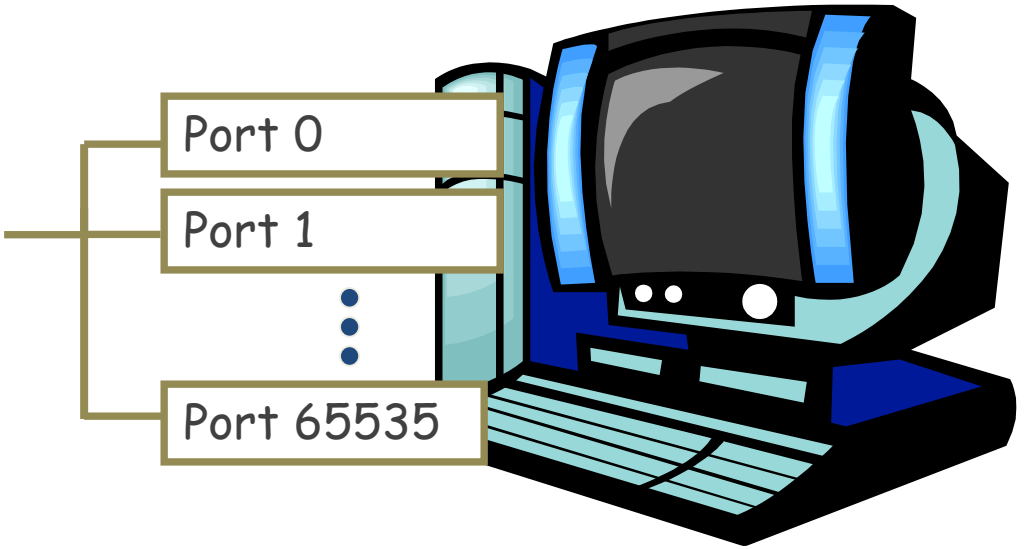
www.cnn.com

mercy.edu

- Each host machine has an IP address
- When a packet arrives at a host

Ports

- Each host has 65,536 ports
- Some ports are *reserved for specific apps*
  - 20,21: FTP
  - 23: Telnet
  - 80: HTTP
  - 443: HTTPS
  - see RFC 1700
    - about 2000 ports are reserved



A socket provides an interface to send data to/from the network through a port

Addresses, Ports and Sockets

- Like apartments and mailboxes
  - You are the application
  - Your apartment building address is the address
  - Your mailbox is the port
  - The post-office is the network
  - The socket is the key that gives you access to the right mailbox (one difference: assume outgoing mail is placed by you in your mailbox)

- Q: How do you choose which port a socket connects to?

The bind function

- associates and (can exclusively) reserves a port for use by the socket
- int status = bind(sockid, &addrport, size);
  - status: error status, = -1 if bind failed
  - sockid: integer, socket descriptor
  - addrport: struct sockaddr, the (IP) address and port of the machine (address usually set to INADDR_ANY – chooses a local address)
  - size: the size (in bytes) of the addrport structure
- bind can be skipped for both types of sockets. When and why?

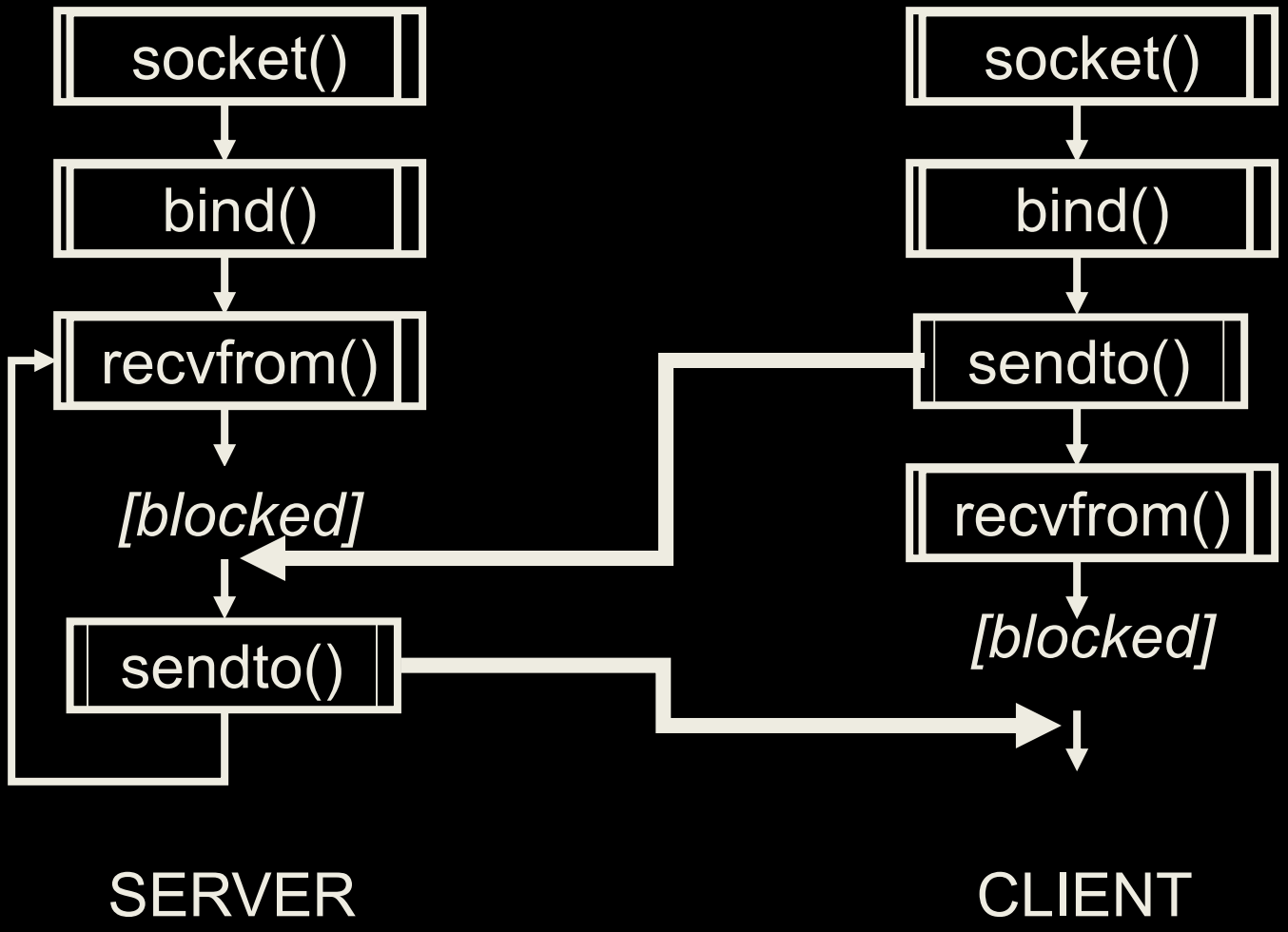
Skipping the bind

- SOCK_DGRAM:
  - if only sending, no need to bind. The OS finds a port each time the socket sends a pkt
  - if receiving, need to bind
- SOCK_STREAM:
  - destination determined during conn. setup
  - don’t need to know port sending from (during connection setup, receiving end is informed of port)

Connection Setup (SOCK_STREAM)

- Recall: no connection setup for SOCK_DGRAM
- A connection occurs between two kinds of participants
  - passive: waits for an active participant to request connection
  - active: initiates connection request to passive side
- Once connection is established, passive and active participants are “similar”
  - both can send & receive data
  - either can terminate the connection

Connectionless Services



Simple Connectionless Server

```
import random
from socket import *

def main():
    s = socket(AF_INET, SOCK_DGRAM)
    s.bind(('', 54321)) # port number

    while 1:
        rand = random.randint(0,10)
        message, address = s.recvfrom(1024)
        print( 'Server recieved ', message)
        message = message.upper()
        if rand >= 4:
            s.sendto(message, address)

main()
```

Note that the *bind()* argument is a two-element tuple of address and port number

Simple Connectionless Client

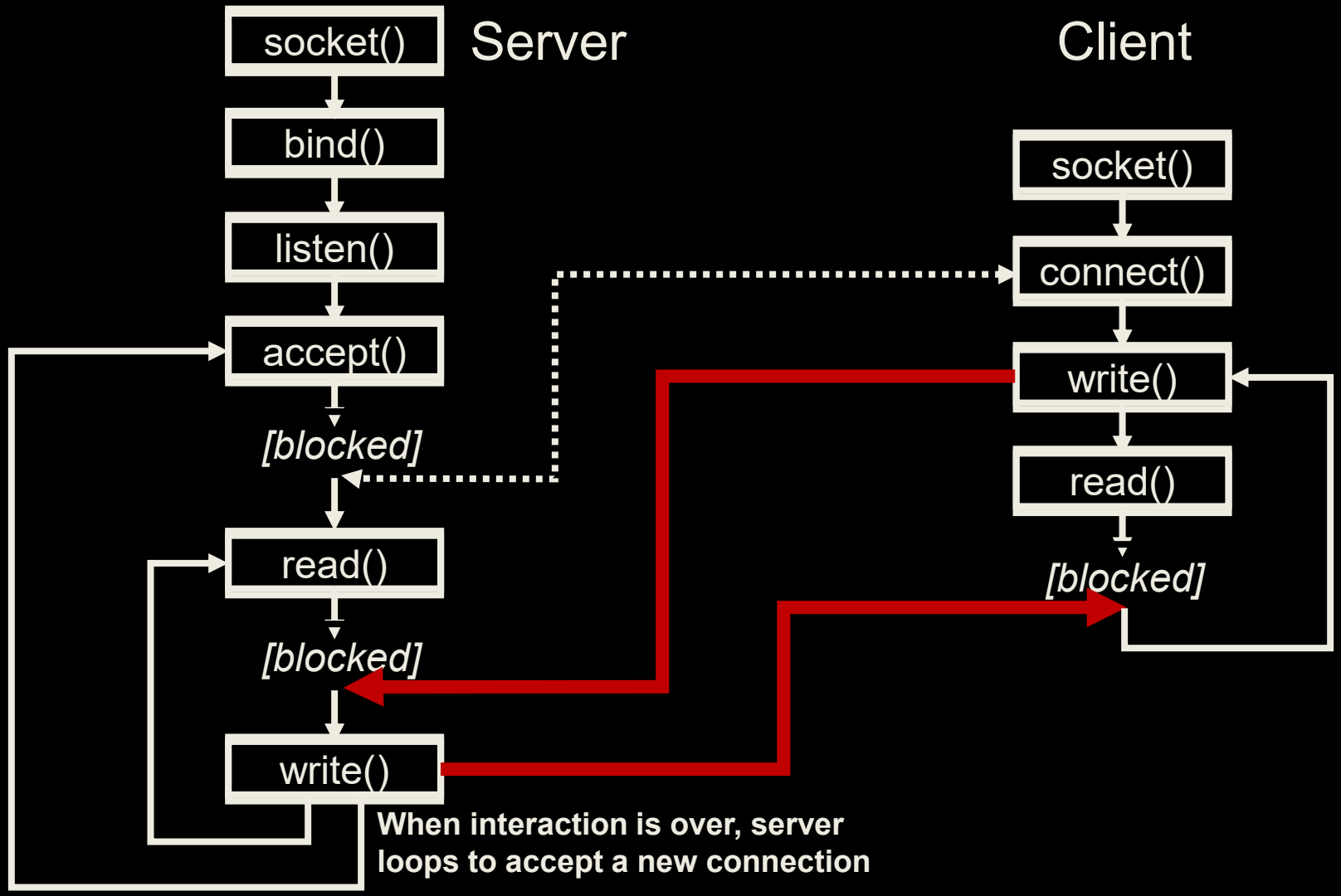
```
import time
from socket import *

for ping in range(10):
    c = socket(AF_INET, SOCK_DGRAM)
    c.settimeout(1)
    message = b"hello, mercy"
    address = ('localhost', 54321) # port number

    start = time.time()
    c.sendto(message, address)

    try:
        data, server = c.recvfrom(1024)
        end = time.time()
        elapsed = end - start
        print("%s of %d took %d" %(data, ping, elapsed))
    except timeout:
        print ("Request timed out")
```

Connection-Oriented Services



Simple Connection Server

```
from socket import *

def main():
  s=socket(AF_INET, SOCK_STREAM)
  s.bind((' ',10530))
  s.listen(1)
  conn, (rmip, rmpt) = s.accept()

  while 1:
    print ("connected by ", str(rmip)+": " + str(rmpt))
    data = conn.recv(1024)
    #print ("What was delivered: ", data.decode())

    #if not data:
       #break
  conn.close()
main()
```

Note that the *bind()* argument is a two-element tuple of address and port number

Simple Connection Client

```
from socket import *

def main():
  s = socket(AF_INET, SOCK_STREAM)
  s.connect(('localhost',10530))
  sendme = "Give me the data"
  s.send(sendme.encode())
main()
```