INDEX

WHAT IS A SOCKET?

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

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

socket returns an integer (socket descriptor) fd < 0 indicates that an error occurred socket descriptors are similar to file descriptors

AF_INET: associates a socket with the Internet protocol family

SOCK_STREAM: selects the TCP protocol

SOCK_DGRAM: selects the UDP protocol

protocol specifies the specific protocol : usually 0, which means the default

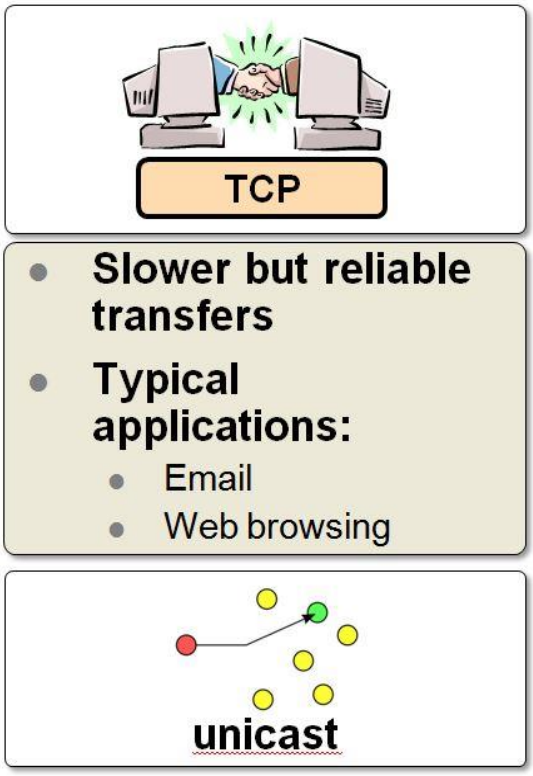
The **socket()** system call returns a socket descriptor (small integer) or -1 on error

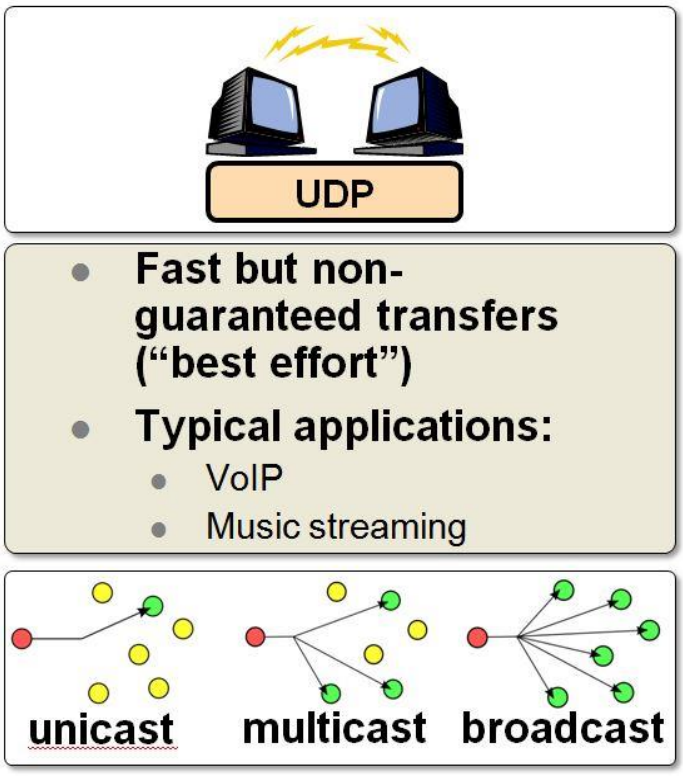
socket() allocates resources needed for a communication endpoint but it does not deal with endpoint addressing

SOCKET PROGRAMMING

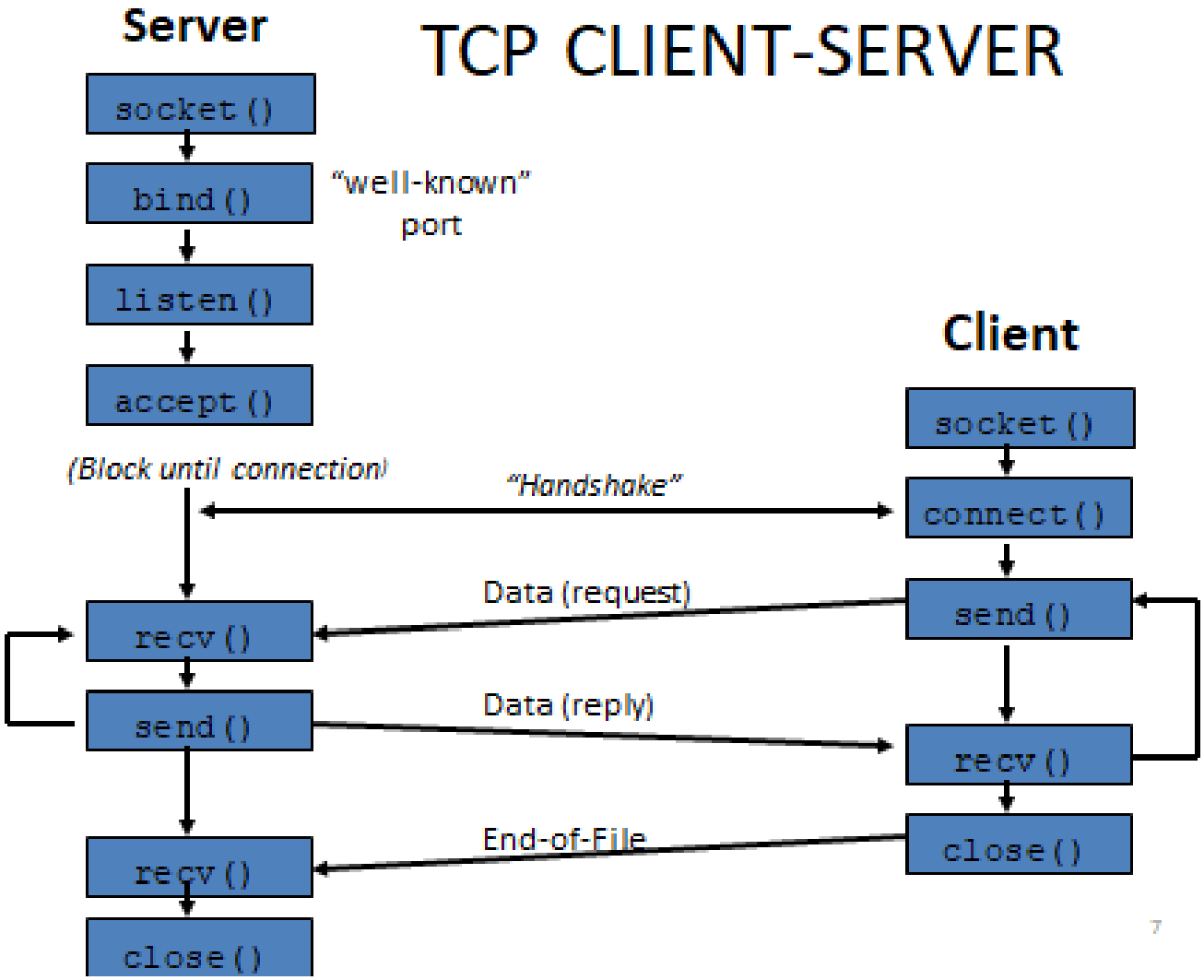
With UDP

With TCP





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) • setuid bit (-rws--x--x root 1997 /sbin/ping*)
- *protocol* is 0 (used for some raw socket options)
- upon success returns socket descriptor – Integer, like file descriptor – Return -1 if failure

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
  - if port is 0, then host will pick ephemeral port • not usually for server (exception RPC port-map)
  - IP address != INADDR_ANY (unless multiple nics)
- *addrlen is length of structure*
- *returns 0 if ok, -1 on error*
  - EADDRINUSE (“Address already in use”)

listen()

int listen(int sockfd, int backlog);

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 a client)*
  - change to passive so OS will accept connection

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
- note, if create new process or thread, can create concurrent server

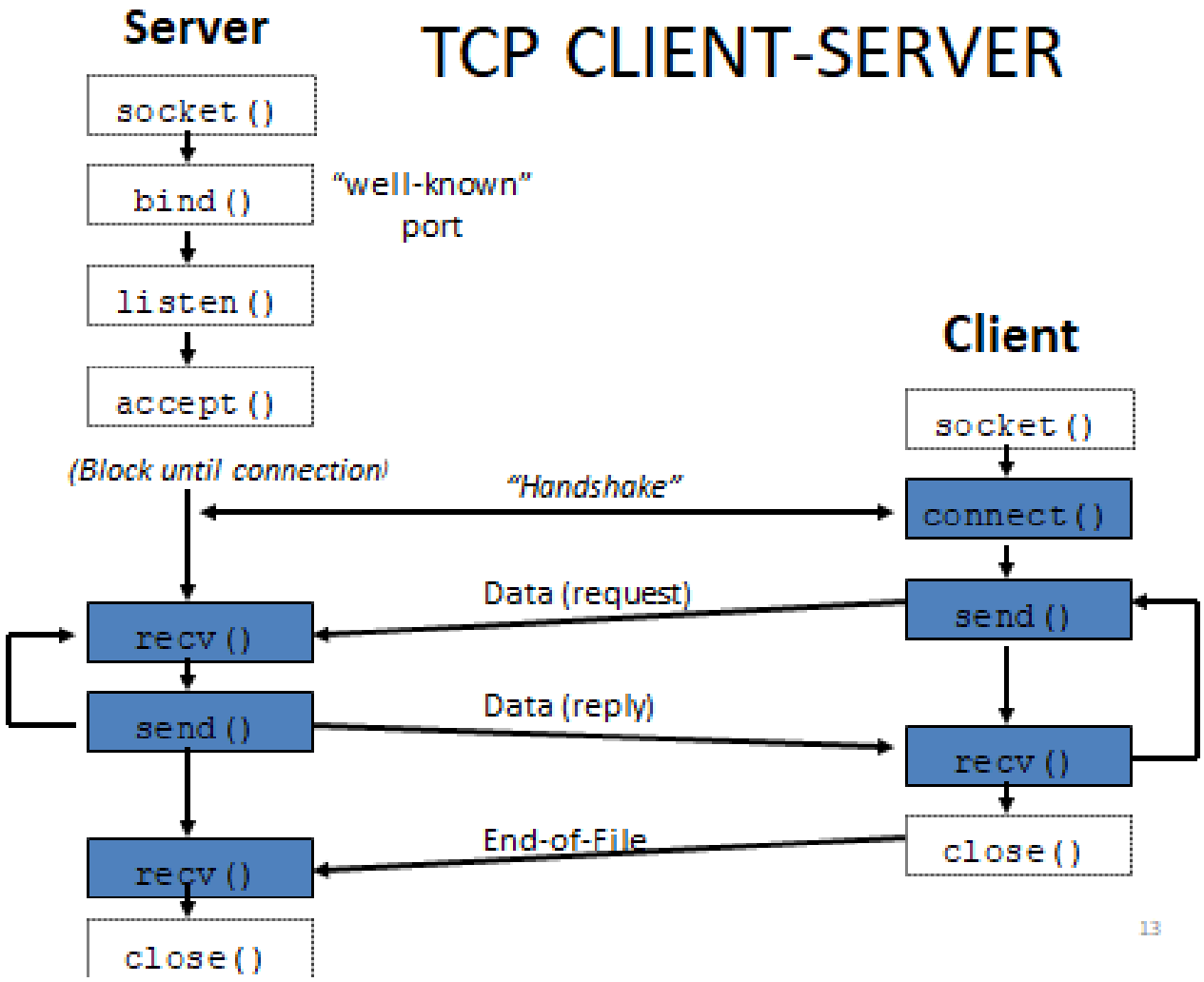
close()

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
  - socket option SO_LINGER
- *block until data sent*
- *or discard any remaining data*
  - returns -1 if error

TCP CLIENT-SERVER



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:*
  - 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*

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*
  - MSG_DONTWAIT (this send non-blocking)
  - MSG_OOB (out of band data, 1 byte sent ahead)
  - MSG_PEEK (look, but don’t remove)
  - MSG_WAITALL (don’t give me less than max)
  - MSG_DONTROUTE (bypass routing table)

IMPEMENTATION OF TCP SERVER AND CLIENT USING PYTHON:

Tcpserver.py

```
from socket import *

import thread


BUFF = 1024

HOST = '127.0.0.1'# must be input parameter @TODO

PORT = 9999 # must be input parameter @TODO

def response(key):

    return key

def handler(clientsock,addr):

    while 1:

        data = clientsock.recv(BUFF)

        if not data: break

        print repr(addr) + ' recv:' + repr(data)

        clientsock.send(response(data))

        print repr(addr) + ' sent:' + repr(response(data))

        if "close" == data.rstrip(): break # type 'close' on client console to
close connection from the server side

    clientsock.close()

    print addr, "- closed connection" #log on console
```

```
if __name__=='__main__':
    ADDR = (HOST, PORT)
    serversock = socket(AF_INET, SOCK_STREAM)
    serversock.setsockopt(SOL_SOCKET, SO_REUSEADDR, 1)
    serversock.bind(ADDR)
    serversock.listen(5)
    while 1:
        print 'waiting for connection... listening on port', PORT
        clientsock, addr = serversock.accept()
        print '...connected from:', addr
        thread.start_new_thread(handler, (clientsock, addr))
```

tcpclient.py

```
import socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
host = raw_input("Server hostname or ip? ")
port = input("Server port? ")
sock.connect((host,port))
while True:
    data = raw_input("message: ")
    sock.send(data)
    print "response: ", sock.recv(1024)
```

SERVER

```
C:\Windows\System32\cmd.exe - python  server.py
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\win8\Desktop\new>python server.c
python: can't open file 'server.c': [Errno 2] No such file or directory

C:\Users\win8\Desktop\new>python server.py
waiting for connection... listening on port 9999
...connected from: ('127.0.0.1', 49646)
waiting for connection... listening on port 9999
('127.0.0.1', 49646) recv:'I am first client.'
('127.0.0.1', 49646) sent:'I am first client.'
...connected from: ('127.0.0.1', 49647)
waiting for connection... listening on port 9999
('127.0.0.1', 49647) recv:'I am second client'
('127.0.0.1', 49647) sent:'I am second client'
('127.0.0.1', 49646) recv:'2nd message from 1st client.'
('127.0.0.1', 49646) sent:'2nd message from 1st client.'
('127.0.0.1', 49647) recv:'2nd message from 2nd client.'
('127.0.0.1', 49647) sent:'2nd message from 2nd client.'
```

CLIENT

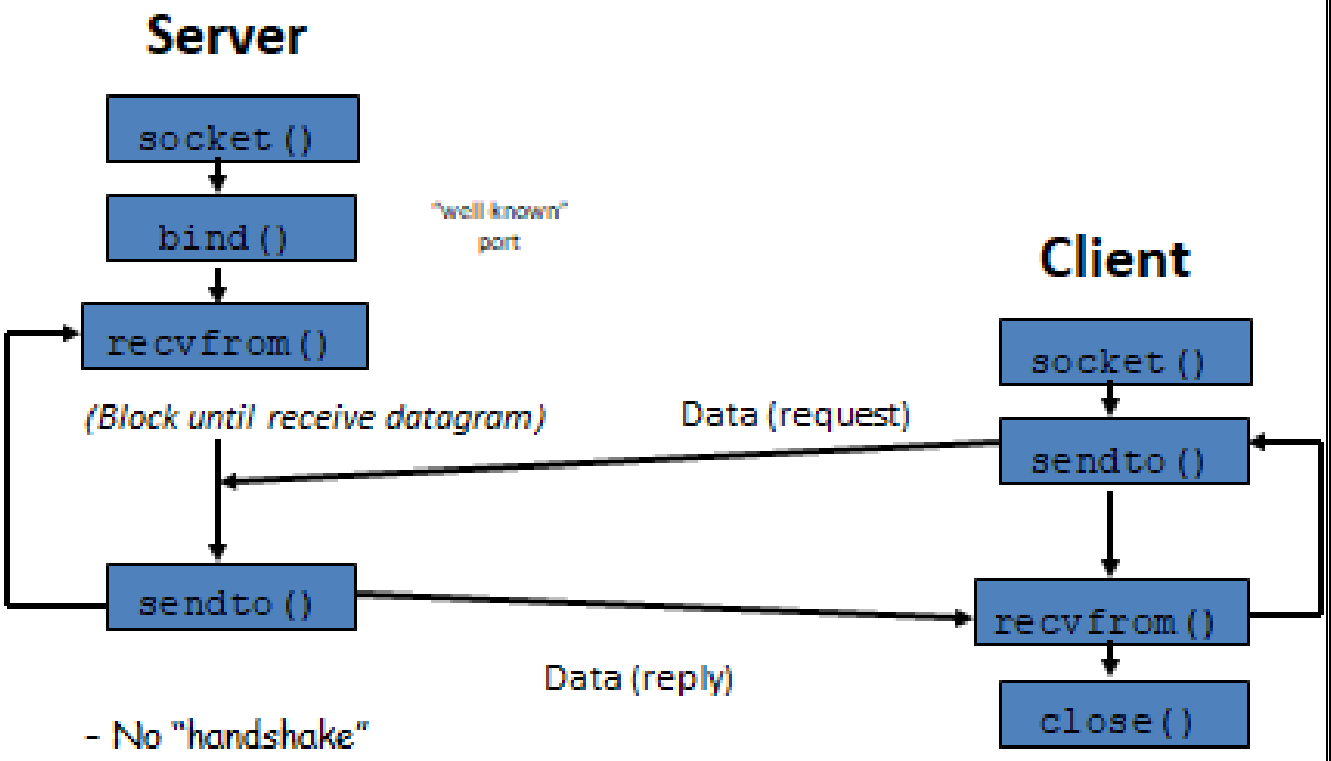
```
C:\Windows\System32\cmd.exe - python client.py
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\win8\Desktop\new>python client.py
Server hostname or ip? localhost
Server port? 9999
message: I am first client.
response:  I am first client.
message: 2nd message from 1st client.
response:  2nd message from 1st client.
message:
```

```
C:\Windows\System32\cmd.exe - python client.py
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\win8\Desktop\new>python client.py
Server hostname or ip? localhost
Server port? 9999
message: I an second client
response:  I an second client
message: 2nd message from 2nd client.
response:  2nd message from 2nd client.
message:
```

UDP CLIENT-SERVER



- No "handshake"
- No simultaneous close
- No `fork()/spawn()` for concurrent servers!

SENDING AND RECEIVING

```
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*
- *(Can do similar with bind() on receiver)*

IMPLEMENTATION OF UDP SERVER AND CLIENT ON PYTHON:

Udpserver.py

```
import socket

import sys


HOST = "   # Symbolic name meaning all available interfaces

PORT = 8888 # Arbitrary non-privileged port


# Datagram (udp) socket

try :

    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    print 'Socket created'

except socket.error, msg :

    print 'Failed to create socket. Error Code : ' + str(msg[0]) + '
Message ' + msg[1]

    sys.exit()



# Bind socket to local host and port

try:

    s.bind((HOST, PORT))
```

```
except socket.error , msg:
    print 'Bind failed. Error Code : ' + str(msg[0]) + ' Message ' + msg[1]
    sys.exit()

print 'Socket bind complete'

#now keep talking with the client
while 1:
    # receive data from client (data, addr)
    d = s.recvfrom(1024)
    data = d[0]
    addr = d[1]

    if not data:
        break

    reply = 'OK...' + data

    s.sendto(reply , addr)
    print 'Message[' + addr[0] + ':' + str(addr[1]) + '] - ' + data.strip()
```

```
s.close()
```

udpclient.py

```
import socket   #for sockets
import sys

# create dgram udp socket
try:
    #Call Socket
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
except socket.error:
    print 'Failed to create socket'
    sys.exit()

host = 'localhost';
port = 8888;

while(1) :
    msg = raw_input('Enter message to send : ')
```

```
try :
    #Send the string to the server
    s.sendto(msg, (host, port))

    # receive data from client (data, addr)
    d = s.recvfrom(1024)
    reply = d[0]
    addr = d[1]

    #display server reply
    print 'Server reply : ' + reply

except socket.error, msg:
    print 'Error Code : ' + str(msg[0]) + ' Message ' + msg[1]
    sys.exit()
```

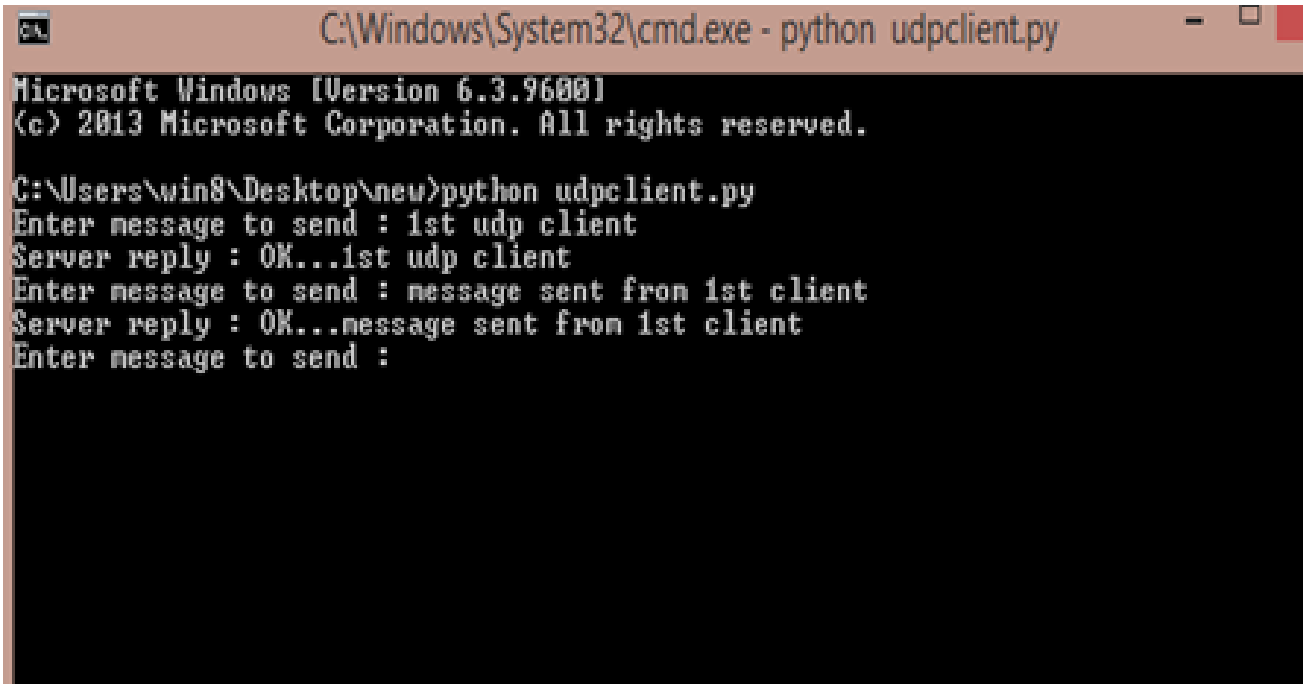
SERVER

C:\Windows\System32\cmd.exe - python udpserver.py

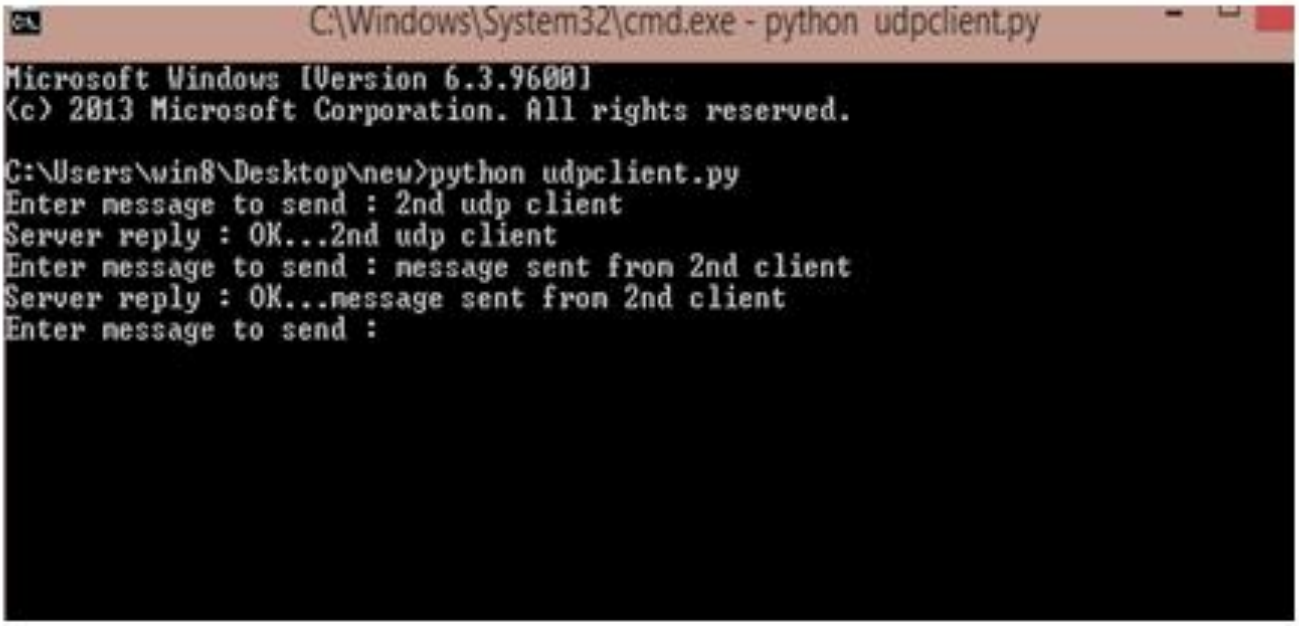
```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\win8\Desktop\new>python udpserver.py
Socket created
Socket bind complete
Message[127.0.0.1:49480] - 1st udp client
Message[127.0.0.1:53693] - 2nd udp client
Message[127.0.0.1:53693] - message sent from 2nd client
Message[127.0.0.1:49480] - message sent from 1st client
```

CLIENT 1


C:\Windows\System32\cmd.exe - python udpclient.py
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.
C:\Users\win8\Desktop\new>python udpclient.py
Enter message to send : 1st udp client
Server reply : OK...1st udp client
Enter message to send : message sent from 1st client
Server reply : OK...message sent from 1st client
Enter message to send :

CLIENT 2


C:\Windows\System32\cmd.exe - python udpclient.py
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.
C:\Users\win8\Desktop\new>python udpclient.py
Enter message to send : 2nd udp client
Server reply : OK...2nd udp client
Enter message to send : message sent from 2nd client
Server reply : OK...message sent from 2nd client
Enter message to send :