

COMPUTER NETWORKS LAB ASSIGNMENT-6(CS-613)

Name: Pranit De Roll No: 39/CSE/16161 Reg No: 0000152



Q. Implement chat (send messages) between client and server using TCP & UDP model.

Implementation using TCP model.

import socket

HOST = ''

PORT = 4040

*def* create\_listen\_socket(*host*, *port*):

""" Setup the sockets our server will receive connection

requests on """

sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

sock.bind((host, port))

sock.listen(100)

return sock

*def* recv\_msg(*sock*):

""" Wait for data to arrive on the socket, then parse into

messages using b'\0' as message delimiter """

data = *bytearray*()

msg = ''

# Repeatedly read 4096 bytes off the socket, storing the bytes

# in data until we see a delimiter

while not msg:

recvd = sock.recv(4096)

if not recvd:

# Socket has been closed prematurely

raise *ConnectionError*()

data = data + recvd

if *b*'\0' in recvd:

# we know from our protocol rules that we only send

# one message per connection, so b'\0' will always be

# the last character

msg = data.rstrip(*b*'\0')

msg = msg.decode('utf-8')

return msg

*def* prep\_msg(*msg*):

""" Prepare a string to be sent as a message """

msg += '\0'

return msg.encode('utf-8')

*def* send\_msg(*sock*, *msg*):

""" Send a string over a socket, preparing it first """

data = prep\_msg(msg)

sock.sendall(data)

*def* parse\_recvd\_data(*data*):

""" Break up raw received data into messages, delimited

by null byte """

parts = data.split(*b*'\0')

msgs = parts[:-1]

rest = parts[-1]

return (msgs, rest)

*def* recv\_msgs(*sock*, *data*=*bytes*()):

""" Receive data and break into complete messages on null byte

delimiter. Block until at least one message received, then

return received messages """

msgs = []

while not msgs:

recvd = sock.recv(4096)

if not recvd:

raise *ConnectionError*()

data = data + recvd

(msgs, rest) = parse\_recvd\_data(data)

msgs = [msg.decode('utf-8') for msg in msgs]

return (msgs, rest)

Client Program:

import sys, socket, threading

import helper

HOST = sys.argv[-1] if len(sys.argv) > 1 else '127.0.0.1'

PORT = helper.PORT

*def* handle\_input(*sock*):

print("Type messages, enter to send. 'q' to quit")

while True:

msg = input() # Blocks

if msg == 'q':

sock.shutdown(socket.SHUT\_RDWR)

sock.close()

break

try:

helper.send\_msg(sock, msg) # Blocks until sent

except (*BrokenPipeError*, *ConnectionError*):

break

if \_\_name\_\_ == '\_\_main\_\_':

sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

sock.connect((HOST, PORT))

print('Connected to {}:{}'.format(HOST, PORT))

thread = threading.Thread(*target*=handle\_input, *args*=[sock], *daemon*=True)

thread.start()

rest = *bytes*()

addr = sock.getsockname()

while True:

try:

(msgs, rest) = helper.recv\_msgs(sock, rest)

for msg in msgs:

print(msg)

except *ConnectionError*:

print('Connection to server closed')

sock.close()

break

Server Program:

import threading, queue

import helper

HOST = helper.HOST

PORT = helper.PORT

send\_queues = {}

lock = threading.Lock()

*def* handle\_client\_recv(*sock*, *addr*):

rest = *bytes*()

while True:

try:

(msgs, rest) = helper.recv\_msgs(sock, rest)

except (*EOFError*, *ConnectionError*):

handle\_disconnect(sock, addr)

break

for msg in msgs:

msg = '{}: {}'.format(addr, msg)

print(msg)

broadcast\_msg(msg)

*def* handle\_client\_send(*sock*, *q*, *addr*):

while True:

msg = q.get()

if msg == None: break

try:

helper.send\_msg(sock, msg)

except (*ConnectionError*, BrokenPipe):

handle\_disconnect(sock, addr)

break

*def* broadcast\_msg(*msg*):

with lock:

for q in send\_queues.values():

q.put(msg)

*def* handle\_disconnect(*sock*, *addr*):

fd = sock.fileno()

with lock:

q = send\_queues.get(fd, None)

if q:

q.put(None)

del send\_queues[fd]

addr = sock.getpeername()

print('Client {} disconnected'.format(addr))

sock.close()

if \_\_name\_\_ == '\_\_main\_\_':

listen\_sock = helper.create\_listen\_socket(HOST, PORT)

addr = listen\_sock.getsockname()

print('Listening on {}'.format(addr))

while True:

client\_sock,addr = listen\_sock.accept()

q = queue.Queue()

with lock:

send\_queues[client\_sock.fileno()] = q

recv\_thread = threading.Thread(*target*=handle\_client\_recv, *args*=[client\_sock, addr], *daemon*=True)

send\_thread = threading.Thread(*target*=handle\_client\_send, *args*=[client\_sock, q, addr], *daemon*=True)

recv\_thread.start()

send\_thread.start()

print('Connection from {}'.format(addr))

Implementation using UDP model.

Client Program:

import socket

UDP\_IP\_ADDRESS = "127.0.0.1"

UDP\_PORT\_NO = 5000

clientSock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

while True:

message = input(*str*("Enter your message : "))

if message == "[e]":

message = "Left chat room!"

clientSock.sendto(message.encode(), (UDP\_IP\_ADDRESS, UDP\_PORT\_NO))

print("\n")

break

clientSock.sendto(message.encode(), (UDP\_IP\_ADDRESS, UDP\_PORT\_NO))

Server Program:

import socket

UDP\_IP\_ADDRESS = "127.0.0.1"

UDP\_PORT\_NO = 5000

serverSock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

serverSock.bind((UDP\_IP\_ADDRESS, UDP\_PORT\_NO))

print("Waiting to recieve messages from client")

while True:

data, addr = serverSock.recvfrom(1024)

print(*f*"Message recieved from client => {data.decode()}")