# **Distributed Computing**

# **Lab#2**

# **Client Server Communication**

# Server:

import select

import socket

HEADER\_SIZE = 15

IP = "127.0.0.1" # same machine

PORT = 3982

# TCP Server Socket

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# For multiple connections

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

server\_socket.bind((IP, PORT))

server\_socket.listen()

# A list of sockets called socket\_list, including server

sockets\_list = [server\_socket]

# Connected clients - key: socket, data: user header and name

clients = {}

utoc ={}

print(f'Listening for connections on {IP}:{PORT}...')

# Handles message receiving

def receive\_message(client\_socket):

try:

# Receive our "header" containing message length

message\_header = client\_socket.recv(HEADER\_SIZE)

# No data recieved, client gracefully closed a connection

if not len(message\_header):

return False

message\_length = int(message\_header.decode('utf-8').strip())

# Return an object of message header and message data

return {'header': message\_header, 'data': client\_socket.recv(message\_length)}

except:

# Client closed connection violently, or lost it.

return False

while True:

# - read\_sockets - sockets on which data is recieved

# - \_ - sockets ready to pass data

# - exception\_socket - sockets with some exceptions

read\_sockets, \_, exception\_sockets = select.select(sockets\_list, [], sockets\_list)

# Iterate over notified sockets

for notified\_socket in read\_sockets:

# New connection, accept it

if notified\_socket == server\_socket:

# Accept new connection

client\_socket, client\_address = server\_socket.accept()

# Receive client name

user = receive\_message(client\_socket)

# Client disconnected

if user is False:

continue

# Add accepted socket to select.select() list

sockets\_list.append(client\_socket)

# Save username and username header

clients[client\_socket] = user

utoc[user['data']] = client\_socket

print('Accepted new connection from {}:{}, username: {}'.format(\*client\_address, user['data'].decode('utf-8')))

# Existing socket is sending a message

else:

# Receive message

message = receive\_message(notified\_socket)

# Client disconnected

if message is False:

print('Closed connection from: {}'.format(clients[notified\_socket]['data'].decode('utf-8')))

# Remove from list for socket.socket()

sockets\_list.remove(notified\_socket)

# Remove from our list of users

del clients[notified\_socket]

continue

# Who sent the message

user = clients[notified\_socket]

print(f'Received message from {user["data"].decode("utf-8")}: {message["data"].decode("utf-8")}')

msg = message["data"].decode('utf-8')

if msg.startswith("@"):

u = msg[1:msg.index(':')]

client\_socket = utoc[u.encode('utf-8')]

#client\_socket = key\_list[uss.index(u.encode('utf-8'))]

spl\_word = ':'

res = msg.partition(spl\_word)[2]

message['data']= res.encode('utf-8')

client\_socket.send(user['header'] + user['data'] + message['header'] + message['data'])

else:

# Broadcast message

for client\_socket in clients:

# Not to send to sender

if client\_socket != notified\_socket:

# Send user and message

client\_socket.send(user['header'] + user['data'] + message['header'] + message['data'])

# Handle some socket exceptions just in case

for notified\_socket in exception\_sockets:

# Remove from list for socket.socket()

sockets\_list.remove(notified\_socket)

# Remove from our list of users

del clients[notified\_socket]

## Client ( send a private msg to say Bob enter msg as @Mali:(your msg here))

import socket

import select

import errno

import sys

HEADER\_SIZE = 15

IP = "127.0.0.1" # same machine

PORT = 1234

my\_username = input(“Enter your username: ")

# Create a TCP socket

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# Connect to a given ip and port

client\_socket.connect((IP, PORT))

# Set connection to non-blocking state, so .recv() call won't block

client\_socket.setblocking(False)

# Prepare username and header and send them

username = my\_username.encode('utf-8')

username\_header = f"{len(username):<{HEADER\_LENGTH}}".encode('utf-8')

client\_socket.send(username\_header + username)

while True:

# Wait for user to input a message

message = input(f'{my\_username} > ')

# Non empty message - send it

if message:

# Encode message to bytes, prepare header and convert to bytes, then send

message = message.encode('utf-8')

message\_header = f"{len(message):<{HEADER\_SIZE }}".encode('utf-8')

client\_socket.send(message\_header + message)

try:

# Now we want to loop over received messages and print them

while True:

# Receive "header" containing username length

username\_header = client\_socket.recv(HEADER\_LENGTH)

# No data received , server gracefully closed a connection

if not len(username\_header):

print('Connection closed by the server')

sys.exit()

# Convert header to int value

username\_length = int(username\_header.decode('utf-8').strip())

# Receive and decode username

username = client\_socket.recv(username\_length).decode('utf-8')

# Do the same for message

message\_header = client\_socket.recv(HEADER\_LENGTH)

message\_length = int(message\_header.decode('utf-8').strip())

message = client\_socket.recv(message\_length).decode('utf-8')

# Print message

print(f'{username} > {message}')

except IOError as e:

# When there are no incoming data error is going to be raised

if e.errno != errno.EAGAIN and e.errno != errno.EWOULDBLOCK:

print('Reading error: {}'.format(str(e)))

sys.exit()

# Received Nothing

continue

except Exception as e:

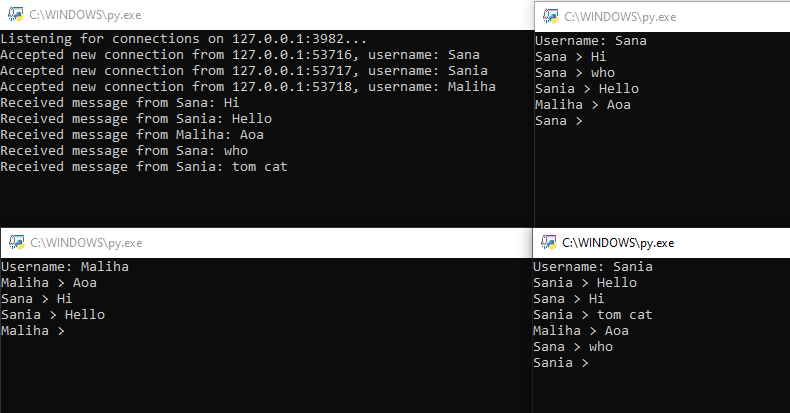
# Any other exception - something happened, exit

print('Reading error: '.format(str(e)))

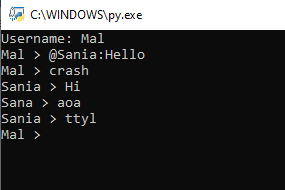
sys.exit()

# Output:

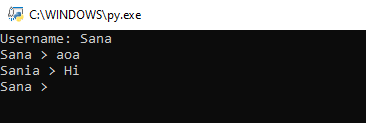
## Broadcasting:



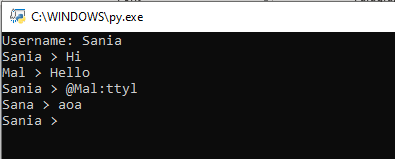
## Server\_Side



## Client\_1 (Sana)



## Client\_2 (Sania)



## Client\_3 (Mal)

