**Universiteti i Prishtinës**

Fakulteti i Inxhinierisë Elektrike dhe Kompjuterike

Programi: Inxhinieri Kompjuterike

****

**Projekti 1 : Dizajnimi klient-server**

**Raporti**

Data: 04.18.2018

Studenti: Nehar Jashari Nr. Indeksit: 160714100152

**Lënda: Rrjeta Kompjuterike**

Profesori i lëndës: Blerim Rexha Assistenti i lëndes: Haxhi Lajqi

Vegla e përdorur për zhvillimin e software-it: Visual Studio 2017 Community Edition

Gjuha programuse e përdorur: Python 3.6.3

Sistemi operativ i përdorur: Windows 10 Pro (64 bit)

# Hyrje

Ky raport përshkruan në detaje punën e bërë gjatë zhvillimit të Protokollit FIEK, detyrë e dhënë në kuadër të lëndës Rrjeta Kompjuterike si Projekti 1. Ky protokoll i lejon klientit dhe serverit ti testoj lidhjet e tyre duke komunikuar nërpërmjet socket-ave. Projekti është punuar në gjuhën programuese Python, verzioni 3.6.3 dhe është zhvilluar si Console Application në Visual Studio 2017 Community Edition.

Mënyra se si funksionon ky protokoll është se programi i krijuar si server përmban disa metoda të cilat janë: IPADDR, PORTNR, ZANORE, PRINTO, HOST, TIME, LOJA, KONVERTO, FIBONACCI, QUADRATIC dhe ROLLTHEDICE, metoda të cilat do të shpjegohen më gjerësisht më tutje. Këto metoda mund të qasen nga klienti i cili dërgon një kërkesë në server duke shkruar emrin e metodës së caktuar dhe serveri iu përgjigjet kërkesës së klientit duke kthyer rezultatin e asaj metode.

Projekti përmban 4 programe të ndryshme: FIEK-TCP klienti, FIEK-TCP serveri, FIEK-UDP klienti dhe FIEK-UDP serveri.

Qëllimi i këtij projekti ka qenë që të kuptohen më mirë konceptet dhe protokollet që lidhen me arkitekturën klient-server, operimin me sockets dhe zhvillimin e programeve që punojnë me këto parime.

**Përmbajtja**

[Hyrje 2](#_Toc512113365)

[Përshkrimi i Metodave 4](#_Toc512113366)

[Programi Klient – Server 8](#_Toc512113367)

[Programimi me Sockets sipas TCP protokollit 9](#_Toc512113368)

[FIEK-TCP klienti 9](#_Toc512113369)

[FIEK-TCP serveri 11](#_Toc512113370)

[Programimi me Sockets sipas UDP protokollit 16](#_Toc512113371)

[Pamjet e Testimit 17](#_Toc512113372)

[Trajtimi i Gabimeve 21](#_Toc512113373)

[Përfundimi 22](#_Toc512113374)

[Referencat 23](#_Toc512113375)

# Përshkrimi i Metodave

Thirrja e metodave bëhet tek klienti duke shkruar emrin e metodës së caktuar. Nëse kërkohet të ipet një argument për zgjidhjen e metodës së caktuar, ajo jepet duke shkruar një hapësirë dhe pastaj argurmentin (parametrin) e funksionit. Nuk duhet të jepen më shumë argumente se sa është e përcaktuar për atë metodë sepse serveri do të lajmëroj gabim (error) në shkrimin e kërkesës nga klienti.

Programet e krijuara përmbajnë këto metoda:

* **IPADDR** – Përcakton dhe kthen IP adresën e klientit në formë dhjetore. IP adresa e klientit është marrë nga vlera e parë e fushës address[ ], e cila përmban të dhënat e klientit i cili lidhet me serverin tonë. Pastaj është kthyer si format të lexueshëm për përdoruesin, me anë të metodës str( ). Me thirrjen e kësaj metode nuk ka nevojë të shënohet ndonjë argument por vetëm emri IPADDR.

def IPADDR(address):

return str(address[0])

* **PORTNR** – Përcakton dhe kthen portin e klientit. Ngjashëm si te metoda IPADDR, por këtu porti i klientit është marrë nga vlera e dytë e fushës address[ ].

def PORTNR(address):

return str(address[1])

* **ZANORE** – Gjen numrin e zanoreve në tëkstin e dhënë dhe kthen si përgjigje numrin e tyre. Kjo metodë merr si parametër një tekst të çfarëdoshëm, pastaj krahason secilin karakter të atij stringu me fushën (array) tonë me emrin *zanoret*, ku janë ruajtur të gjitha zanoret e alfabetit shqip. Nëse karakteri i caktuar i tekstit përputhet me *zanoret,* atëherë inkrementohet për një vlerë variabla që ruan numrin e zanoreve. Me përshkimin e gjithë tekstit të caktuar kthehet nga metoda numri i zanoreve në atë tekst.

def ZANORE(teksti):

numriZanoreve = 0

zanoret = "aeiouyëAEIOUYË"

for i in range(1, len(teksti)):

if teksti[i] in zanoret:

numriZanoreve +=1

return str(numriZanoreve)

* **PRINTO** – Merr si parametër një tekst të caktuar, te i cili hiqen hapësirat në fillim dhe në fund të tekstit nëpërmes metodës strip( ), dhe pastaj kthehet teksti i caktuar si i “pastërt”.

def PRINTO(teksti):

teksti = str(teksti).strip()

return teksti

* **HOST** – Kërkon dhe kthen emrin e hostit i cili mirret nga objekti socket nëpërmes metodës *gethostname( ),* i cili ruhet në variablën e përcaktuar “hosti”. Nëse nuk mund të përcaktohet hosti i klientit, atëherë funksioni kthen një mesazh ku tregon se emri i hostit nuk mund të përcaktohet, kjo është bërë nëpërmjet një *try-except* të definuar si mëposhtë.

def HOST():

try:

hosti = socket.gethostname()

pergjigja = "Emri i hostit eshte: " + hosti

return str(pergjigja)

except:

pergjigja = "Emri i hostit nuk mund te percaktohet!"

return str(pergjigja)

* **TIME** – Kthen kohën aktuale në server, si format të lexueshëm për përdoruesit. Kjo është përfituar nga libraria e importuar nga python datetime, ku është marrë koha aktuale në system përmes metodës *datetime.now( )*.

def TIME():

return str(datetime.datetime.now())

* **LOJA** – Kthen një varg të 20 numrave random të rangut 1 deri 99. Vargu i numrave është marrë nëpërmes një unaze që zhvillon 20 cikle, ku gjendet nga një numër random i rangut 1 deri 99 dhe ruhet në listën e përcaktuar më parë me emrin “listaNumrave”, e cila pastaj sortohet dhe kthehet si format të lexueshëm për klientin.

def LOJA():

listaNumrave = []

for i in range(20):

listaNumrave.append(random.randint(1,99))

listaNumrave.sort()

return str(listaNumrave)

* **FIBONACCI** – Gjen numrin Fibonacci që paraqet si rezultat i parametrit të dhënë hyrës.

def FIBONACCI(vlera):

numri = 1

numripr = 0

numriDhene = 0

numriDhene = int(vlera)

for i in range(numriDhene-1):

numri = numri + numripr

numripr = numri - numripr

return str(numri)

* **KONVERTO** – Kthen si rezultat konvertimin e parametrit të dhënë varësisht prej opcionit të zgjedhur. Opcionet e konvertimit janë: CelsiusToKelvin, CelsiusToFahrenheit, KelvinToCelsius, KelvinToFahrenheit, FahrenheitToCelsius, FahrenheitToKelvin, PoundToKilogram, KilogramToPound. Metoda merr dy parametra nga klienti, së pari tipin e konvertimit dhe pastaj vlerën që deshiron të konvertoj. Funksioni sipas tipit të dhënë përmes disa kushtëzimeve përcakton cili tip është përzgjedhur dhe kthen rezultatin e caktuar, që paraqet vlerën e konvertuar. Nëse nuk jepet tipi i cili është përcaktuar nga serveri metoda kthen një mesazh që e lajmëron klientin se ka gabuar në shkrimin e këskesës.

def KONVERTO(tipi,vlera):

vleraKonvertuar = 0

vlera = float(vlera)

if tipi=="CELSIUSTOKELVIN":

return str(vlera+273.15)

elif tipi=="CELSIUSTOFAHRENHEIT":

return str((vlera\*9.0/5.0)+32)

elif tipi=="KELVINTOFAHRENHEIT":

return str((9.0\*(vlera-273.15)/5.0)+32)

elif tipi=="KELVINTOCELSIUS":

return str(vlera-273.15)

elif tipi=="FAHRENHEITTOCELSIUS":

return str(5.0\*(vlera-32)/9.0)

elif tipi=="FAHRENHEITTOKELVIN":

return str(5.0\*(vlera-32)/9.0+273.15)

elif tipi=="POUNDTOKILOGRAM":

return str(vlera\*0.45359237)

elif tipi=="KILOGRAMTOPOUND":

return str(vlera/0.45359237)

else:

return "Error - Keni bere ndonje gabim gjate shenimit!"

* **QUADRATIC** – Kjo metodë merr si parametra 3 numra të caktuar, që paraqesin vlera a, b dhe c të ekuacionit kuadratic: ax2 + bx + c = 0. Fillimisht zgjidhet determinant d sipas formulës së caktuar, pastaj nërpëmjes klasës cmath dhe metodës sqrt( ) gjenden dy zgjidhjet e këtij ekuacioni. Zgjidhjet kthehen te klienti në një format të lexueshëm duke shfrytëzuar funksionin e parapërcaktuar për stringje: *format( ).*

def QUADRATIC(a, b, c):

d = (b\*\*2) - (4\*a\*c)

x1 = (-b - cmath.sqrt(d))/(2\*a)

x2 = (-b + cmath.sqrt(d))/(2\*a)

pergjigja = str("Zgjidhjet e ekuacionit kuadratik jane: x1 = {:.2f} dhe x2 = {:.2f}").format(x1,x2)

return pergjigja

* **ROLLTHEDICE** – Kjo metodë simulon hudhjen e dy zareve në mënyrë virtuale, pra kthen dy numra të plotë random nga 1 deri në 6.

def ROLLTHEDICE():

min = 1

max = 6

kubi1 = str(random.randint(min, max))

kubi2 = str(random.randint(min, max))

pergjigja = str("Rolling the dice... Vlerat e qelluara jane: " + kubi1 + " dhe " + kubi2)

return pergjigja

# Programi Klient – Server

Me ekzekutimin e këto dy programeve, ku së pari duhet të ekzekutohet programi server pastaj ai i klientit, krijohen dy procese ai i serverit dhe ai i klientit. Këto procese komunikojnë me njëri-tjetrin nëpërmjet socket-ave.

Vendimi i parë të cilin duhet të marrin zhvilluesit e aplikacioneve është nëse dëshirojnë të përdorin TCP protokollin apo UDP protokollin. Përkujtojmë se TCP protokolli është një lidhje e orientuar dhe siguron një transfer të besueshëm të të dhënave, kurse UDP protokolli nuk garanton dërgimin e të gjitha të dhënave nga një host në tjetrin dhe nuk ka nevojë të vendoset një lidhje që të shkëmbehen të dhënat.

Klienti dhe serveri gjatë zhvillimit të programeve të FIEK protokollit kanë përdorur portin 11000, kurse si host është përdorur ‘localhost’-i.

Programi Server është i dizajnuar në atë mënyrë që të punoj vazhdimisht pa ndërprerje, vetëm nësë ndodh ndonjë gabim apo kërkohet nga klienti dalja nga programi. Serveri është programuar ashtu që të jetë në gjendje të pranoj një sekuencë të kërkesave nga i njëjti klient apo nga klientë të ndryshëm në vazhdimësi.

Programi Klient punon në atë mënyre që të jetë në gjendje të shkruaj një kërkesë, e cila përmban emrin e metodës së caktuar dhe ta dërgoj atë te programi i serverit i cili e merr atë kërkesë dhe kthen një përgjigje të caktuar te klienti.

# Programimi me Sockets sipas TCP protokollit

TCP protokolli gjatë komunikimit klienti-server garanton dërgimin e të gjitha të dhënave, pa asnjë humbje. Pra është një protokoll i besueshëm për shkëmbimin e të dhënave.

TCP protokolli është *connection-oriented*, pra para se të shkëmbehen të dhënat ndërmjet klientit dhe serverit fillimisht duhet të vendoset një lidhje TCP. Pasiqë që të vendoset lidhja, nëse njëra anë dëshiron të dërgoj të dhëna ajo vetëm vendos të dhënat në lidhjen TCP përmes socketit dhe dërgohen në anën tjetër.

Klenti ka për detyrë që të inicioj kontaktin me server. Në mënyrë që serveri të jetë në gjendje të reagoj ndaj kontaktit të klientit, ai duhet të jetë gati, pra fillimisht duhet të jetë duke punuar serveri. Te klienti krijohet socketi i klientit që punon sipas TCP protokollit, ku i bashkangjitet atij socketi adresa e socketit të serverit, që është IP adresa e hostit të serverit dhe portit të tij. Pasiqë që të krijohet socketi, klienti inicion një “handshake” me serverin dhe vendos një TCP lidhje me të. Me vendosjen e kësaj lidhjeje është e mundur të shkëmbehen të dhënat përmes socketave të cilët punojnë sipas parimit të TCP protokollit.

## FIEK-TCP klienti

Te programi TCP klienti fillimisht janë importuar libraritë: socket dhe sys.

from socket import \*

import sys

Pastaj janë përcaktuar hosti i serverit dhe porti i tij në variablat e përshkruara:

serverName = 'localhost'

serverPort = 11000

Kemi krijuar socketin e klientit me emrin e variablës si “s”, ku parametri i parë i socketit paraqet familjen e IP adresave të përdorura IPv4, kurse parametri i dytë tregon se tipi i socketit është SOCK\_STREAM, që do të thotë se kemi të bëjmë me TCP socket.

s = socket(AF\_INET, SOCK\_STREAM)

Vendoset lidhja TCP ndërmjet klientit dhe serverit përmes metodës connect( ), ku parametri i parë i saj duhet të jetë hosti i serverit, kurse parametri i dytë numri i portit. Në këtë mënyrë arrihet “three-way handshake” dhe vendoset lidhja TCP ndërmjet klientit dhe serverit.

s.connect((serverName,serverPort))

Në programin e krijuar do të printohet edhe një menu e krijuar që të ndihmoj klientin për përzgjedhjen e metodave:

# MENYJA e programit

print("""Jeni te lidhur me serverin!

Cilat nga keto metoda deshironi ti perdorni:

1.IPADDR - Percakton dhe kthen IP adresen e klientit

2.PORTNR - Percakton dhe kthen portin e klientit

3.ZANORE {hapesire} teksti - Merr si parameter nje tekst dhe kthen numrin e zanoreve ne ate tekst

4.PRINTO {hapesire} teksti - Merr si parameter nje tekst dhe kthen fjaline e shtypur dhe te formatuar

nga hapesirat e teperta

5.HOST - Kerkon emrin e hostit dhe e kthen ate

6.TIME - Kthen kohen aktuale ne server

7.LOJA - Kthen vargun e 20 numrave "random" te rangut [1,99]

8.FIBONACCI {hapesire} numri - Gjen numrin Fibonacci si rezultat i parametrit te dhene

9.KONVERTO {hapesire} opcioni {hapesire} vlera - Kthen si rezultat konvertimin e parametrit hyres sipas ketyre opcioneve:

CelsiusToKelvin, CelsiusToFahrenheit, KelvinToFahrenheit

KelvinToCelsius, FahrenheitToCelsius, FahrenheitToKelvin

PoundToKilogram, KilogramToPound.

10.QUADRATIC {hapesire} vleraA {hapesire} vleraB {hapesire} vleraC

- Merr si parameter vlerat e ekuacionit kuadratik (a, b, c) dhe kthen

zgjidhjet e atij ekuacioni

11.ROLLTHEDICE - Kthen dy vlera "random" te dy zareve te gjuajtura

Sheno EXIT per te dalur nga programi!

========================================================================================================================================

""")

Klienti është programuar në atë mënyrë që të punoj vazhdimisht derisa ai vet të dëshiroj ta mbyll lidhjen apo të mbyllet vetë nga ndonjë gabim eventual. Kjo është arritur përmes një unaze while 1, që teoritikisht punon deri në “pafundësi”, apo deri sa të ndërprehet nga klienti.

while 1:

try:

kerkesa = input('Shkruaj emrin e metodes dhe argumentin perkates: ')

if kerkesa!="" and kerkesa.upper()!="EXIT":

# behet enkodimi i kerkeses dhe dergimi tek TCP serveri

s.sendall(str(kerkesa).encode())

else:

break

# ketu behet pranimi i pergjigjes nga serveri dhe dekodimi i saj

# madhesia me e madhe qe mund të mirret eshte 128 byte

data = s.recv(128)

print('Te dhenat e pranuara nga serveri: ')

print(str(data.decode()).strip())

print("----------------------------------------------------------------------------------------\n")

except Exception as e:

print(str(e))

break

Në fund mbyllet lidhja e vendosur:

s.close()

## FIEK-TCP serveri

Fillimisht janë importuar libraritë që do të nevojiten për programimin e programit tonë:

from socket import \*

import random

import datetime

import sys

import threading

import cmath

Pastaj ndahet porti i serverit:

serverPort = 11000

Është krijuar socketi i serverit sipas TCP protokollit si në rastin e klientit.

serverSocket = socket(AF\_INET, SOCK\_STREAM)

Vazhdon programimi me lidhjen apo bashkangjitjen e portit të serverit me socketin e krijuar sipas kodit në vijim:

serverSocket.bind(('',serverPort))

Pasiqë të krijohet socketi i serverit te TCP protokolli presim për një klient të lidhet me server, me fjalë të tjera ndegjojmë për ndonjë lidhje me klient.

serverSocket.listen(5)

Shfaqim një mesazh në server se: “Serveri është i gatshëm të pranoj kërkesa…”.

print('Serveri eshte i gatshem te pranoj kerkesa...')

Dhe pastaj definojmë metodat sipas kodit të shfaqur më parë te Përshkrimi i metodave.

Definojmë një funksion shtesë që përdoret për shtypjen në server të të dhënave që dërgohen te klienti. Ky funksion do të thirret pas çdo dërgese të përgjigjeve nga serveri në klient dhe do të shtyp ato të dhëna në programin e serverit. Kodi për defimin e kësaj metode është:

def ShtypTeDhenat(teDhenat):

print("\n----------------------------------------------------------------------------------------------")

print("Te dhenat e derguara te klienti => ", teDhenat)

return

Bëhet definimi i një funksioni i cili thirret pas çdo krijimi të një procesi (thread) të ri, që bëhet me lidhjen e një klienti të ri me server. Ky funksion merr si parametra socketin i cili krijohet dhe i dedikohet klientit të lidhur me server, kurse parametri tjetër paraqet adresën e tij. Në këtë funksion fillimisht pranohet kërkesa nga klienti, bëhet dekodimi i saj, ajo kërkesë ndahet sipas hapësirave dhe ruhet në një Array me emrin kerkesaArray. Vlera e parë e kësaj fushe paraqet emrin e metodës e cila kërkohet nga klienti. Ky string kthehet ne upper case që të përputhet me kushtëzimet e përcaktuara nga zhvilluesi i kodit dhe pastaj bëhet krahasimi i këtij emri me emrin e metodave të parapërcaktuara nga ne. Në bazë të kërkesës përzgjedhet edhe metoda qe duhet të kthehet nga serveri dhe dërgohet te klienti si në vijim: *connectionSocket.send( \_\_\_\_\_\_\_\_\_\_\_\_\_ .encode( ) ).*

Nëse kërkesa nuk përputhet me asnjë nga metodat e përcaktuara nga ne, nga serveri dërgohet një mesazh që “Kërkesa juaj nuk është valide!”, me kod:

connectionSocket.send("Kerkesa juaj eshte invalide, ju lutem provoni perseri!".encode())

Kodi i shkruar për definimin e kësaj metode është:

def klienti\_i\_ri(connectionSocket,addr):

kerkesa = (bytes)("empty".encode())

try:

while str(kerkesa.decode()).upper()!="EXIT" and str(kerkesa.decode())!="":

# pranimi i kerkeses nga klienti

kerkesa = connectionSocket.recv(128)

# dekodimi i kerkeses

kerkesaStr = str(kerkesa.decode()).strip()

# behet ndarja e kerkeses sipas hapesirave dhe ruhet ne një Array

kerkesaArray = kerkesaStr.split(' ')

# kerkesa e derguar nga klienti kthehet se pari ne upperCase:

kerkesaArray[0] = kerkesaArray[0].upper()

# metoda IPADDR

if kerkesaArray[0]=="IPADDR":

if len(kerkesaArray) == 1:

connectionSocket.send(("IP adresa juaj eshte: "+IPADDR(addr)).encode())

ShtypTeDhenat(("IP adresa e klientit: "+IPADDR(addr)))

else:

connectionSocket.send("Kerkesa juaj eshte invalide, ju lutem provoni perseri!".encode())

ShtypTeDhenat("ERROR")

# metoda PORTNR

elif kerkesaArray[0]=="PORTNR":

if len(kerkesaArray) == 1:

connectionSocket.send(("Numri i portit tuaj eshte: "+PORTNR(addr)).encode())

ShtypTeDhenat(("Numri i portit te klientit eshte: "+PORTNR(addr)))

else:

connectionSocket.send("Kerkesa juaj eshte invalide, ju lutem provoni perseri!".encode())

ShtypTeDhenat("ERROR")

# metoda ZANORE

elif kerkesaArray[0]=="ZANORE":

if len(kerkesaArray) == 2:

rezultati = "Numri i zanoreve ne fjalen e dhene eshte: " + ZANORE(kerkesaArray[1])

connectionSocket.send(rezultati.encode())

ShtypTeDhenat(rezultati)

else:

connectionSocket.send("Kerkesa juaj eshte invalide, ju lutem provoni perseri!".encode())

ShtypTeDhenat("ERROR")

# metoda PRINTO

elif kerkesaArray[0]=="PRINTO":

kerkesaStr2 = (str(kerkesaStr)).replace("PRINTO","")

connectionSocket.send(("Fjalia e printuar: " + PRINTO(kerkesaStr2)).encode())

ShtypTeDhenat(("Fjalia e printuar: " + PRINTO(kerkesaStr2)))

# metoda HOST

elif kerkesaArray[0]=="HOST":

if len(kerkesaArray) == 1:

if HOST()=="Emri i hostit nuk mund te percaktohet!":

connectionSocket.send(("Emri i hostit nuk mund te percaktohet!").encode())

ShtypTeDhenat("Emri i hostit nuk mund te percaktohet!")

else:

connectionSocket.send(("Emri i klientit eshte: "+HOST()).encode())

ShtypTeDhenat(("Emri i klientit eshte: "+HOST()))

else:

connectionSocket.send("Kerkesa juaj eshte invalide, ju lutem provoni perseri!".encode())

ShtypTeDhenat("ERROR")

# metoda TIME

elif kerkesaArray[0]=="TIME":

if len(kerkesaArray) == 1:

connectionSocket.send(("Koha e tanishme eshte: " + TIME()).encode())

ShtypTeDhenat(("Koha e tanishme eshte: " + TIME()))

else:

connectionSocket.send("Kerkesa juaj eshte invalide, ju lutem provoni perseri!".encode())

ShtypTeDhenat("ERROR")

# metoda LOJA

elif kerkesaArray[0]=="LOJA":

if len(kerkesaArray) == 1:

connectionSocket.send(("Rezultati nga loja: " + LOJA()).encode())

ShtypTeDhenat(("Rezultati nga loja: " + LOJA()))

else:

connectionSocket.send("Kerkesa juaj eshte invalide, ju lutem provoni perseri!".encode())

ShtypTeDhenat("ERROR")

# metoda FIBONACCI

elif kerkesaArray[0]=="FIBONACCI":

for i in range(len(kerkesaArray)):

if "" in kerkesaArray:

kerkesaArray.remove("")

if len(kerkesaArray)==1 or len(kerkesaArray)>2:

connectionSocket.send(("Kerkesa juaj eshte invalide, ju lutem provoni perseri!").encode())

ShtypTeDhenat("ERROR")

else:

connectionSocket.send(("Numri i "+kerkesaArray[1]+" ne serine fibonacci eshte: "+FIBONACCI(kerkesaArray[1])).encode())

ShtypTeDhenat(("Numri i "+kerkesaArray[1]+" ne serine fibonacci eshte: "+FIBONACCI(kerkesaArray[1])))

# metoda KONVERTO

elif kerkesaArray[0]=="KONVERTO":

for i in range(len(kerkesaArray)):

if "" in kerkesaArray:

kerkesaArray.remove("")

if len(kerkesaArray)>3 or len(kerkesaArray)<3:

connectionSocket.send(("Kerkesa juaj eshte invalide, ju lutem provoni perseri!").encode())

ShtypTeDhenat("ERROR")

else:

konvertimi = str(kerkesaArray[1]).loëer().split("to")

pergjigja = kerkesaArray[2] + " " + str(konvertimi[0]) + " jane te barabarte me " + KONVERTO(str(kerkesaArray[1]).upper(),kerkesaArray[2]) + " " + str(konvertimi[1])

connectionSocket.send(str(pergjigja).encode())

ShtypTeDhenat(pergjigja)

# metoda shtese QUADRATIC - shkruan si parametra vlerat a,b,c te ekuacionti kuadratik dhe kthen zgjidhjet e tij.

elif kerkesaArray[0]=="QUADRATIC":

if len(kerkesaArray) == 4:

connectionSocket.send(str(QUADRATIC(kerkesa[1], kerkesa[2], kerkesa[3])).encode())

ShtypTeDhenat(str(QUADRATIC(kerkesa[1], kerkesa[2], kerkesa[3])))

else:

connectionSocket.send("Kerkesa juaj eshte invalide, ju lutem provoni perseri!".encode())

ShtypTeDhenat("ERROR")

# metoda shtese ROLLTHEDICE - kthen vlerat e dy kubeve te gjuajtura.

elif kerkesaArray[0]=="ROLLTHEDICE":

if len(kerkesaArray) == 1:

connectionSocket.send(str(ROLLTHEDICE()).encode())

ShtypTeDhenat(str(ROLLTHEDICE()))

else:

connectionSocket.send("Kerkesa juaj eshte invalide, ju lutem provoni perseri!".encode())

ShtypTeDhenat("ERROR")

# nqs. nuk shkruhet asnjera nga metodat e percaktuara

else:

connectionSocket.send("Kerkesa juaj eshte invalide, ju lutem provoni perseri!".encode())

ShtypTeDhenat("ERROR!")

# kur largohet klienti mbyllet lidhja me te, por serveri ende pret per lidhje me klient te tjere

connectionSocket.close()

except Exception as e:

print("\nERROR: ")

print(str(e))

connectionSocket.close()

Serveri është programuar në atë mënyrë që të jetë në gjendje të pranoj kërkesa të njëpasnjëshme dhe nga klienti të ndryshëm. Kjo është bërë përmes një unaze while 1, ku nga çdo klienti i ri që i qaset serverit krijohet një thread i ri për atë kërkesë. Pas kësaj kërkese krijohen threads të reja për klienti të ri deri sa vetë klientët e ndërprejnë lidhjen. Por serveri gjithmonë është i gatshëm për pranimin e kërkesave përderisa punon (is running).

while 1:

# ky rresht ben qe serveri te "degjoj" per kerkesa nga klienti permes lidhjes TCP

connectionSocket, addr = serverSocket.accept()

print('Klienti me IP adrese %s dhe me numrin e portit %s eshte lidhur me server' %(addr))

# krijimi i nje procesi te ri (threadi te ri), me lidhjen e nje klienti te ri

threading.\_start\_new\_thread(klienti\_i\_ri,(connectionSocket,addr))

# Programimi me Sockets sipas UDP protokollit

UDP protokolli paraqet një protokoll jo të besueshëm, pra nuk ka garancion se të dhënat do të mbërrijnë në cak gjithmonë dhe që do të ruhet rendi i dërgesës ashtu siç e kemi përcaktuar. Dallimi nga programimi sipas TCP protokollit qëndron se UDP protokolli ëhtë *connection-less,* pra nuk vendoset një lidhje ndërmjet klientit dhe serverit që ata të komunikojnë mes vete.

Gjatë krijimit të socketit si të klientit ashtu edhe te serveri, parametri i dytë i socketit do të ketë një tip tjetër të tij, pra do të jetë SOCK\_DGRAM, që do të thotë se kemi të bëjmë me UDP socket.

s = socket(AF\_INET, SOCK\_DGRAM)

Si rrjedhim që nuk kemi nevojë të vendosim një lidhje në mes klientit dhe serverit si te TCP protokolli atëherë nuk na duhet metoda connect( ) te ana e klientit.

Dallimi tjetër qëndron në metodën e dërgimit të të dhënave që këtu përdoret metoda sendto( ), ku si parametra iu bashkangjitet mesazhit adresa e plote e destinacionit, pra emri i hostit dhe numri i portit të serverit. Pasiqë të dërgohet paketa, klienti pret të marr përgjigje nga serveri.

s.sendto(str(kerkesa).encode(), (serverName, serverPort))

Te ana e serverit dallimi qëndron në metodën e pranimit të kërkesës nga klienti që është: recvfrom(128).

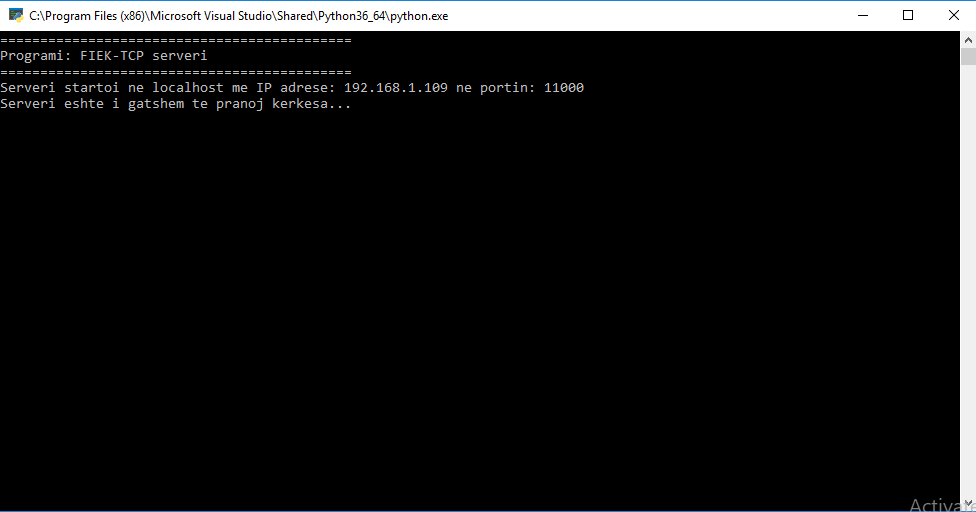
data, serverAddress = s.recvfrom(128)

Te UDP serveri nuk kemi nevojë të krijojmë threads të ri pas çdo kërkese të klientit si te rasti i TCP, pra nuk kemi pasur nevojë të definojmë metodën klienti\_i\_ri( ), por validimi i të dhënave, mënyra e qasjes ndaj kërkesës dhe kthimi i përgjigjes në bazë të asaj kërkese është bërë përafërsisht në të njëjtën mënyrë. Njëjtë realizohet edhe UDP serveri përmes një unaze while 1, ku janë vendosur kushtëzimet për përzgjedhjen e metodës dhe dërgimin e përgjigjes të klienti. Serveri është i gatshëm të pranoj kërkesa të vazhdueshme dhe të mos ndërpres lidhjen me klientin.

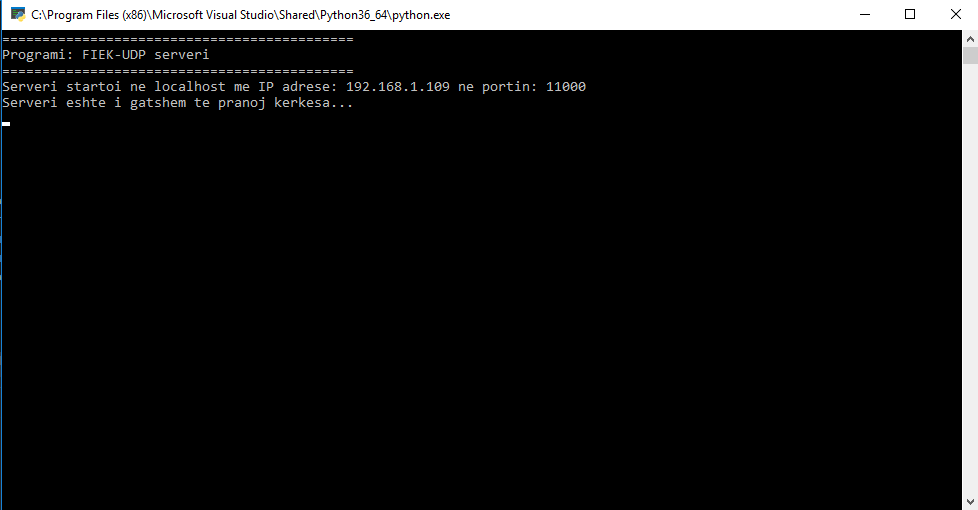
Te UDP serveri kemi edhe një dallim se nuk duhet të mbyllin lidhjen me klientin pas një kërkese, pra nuk duhet të kemi s.close( ).

# Pamjet e Testimit

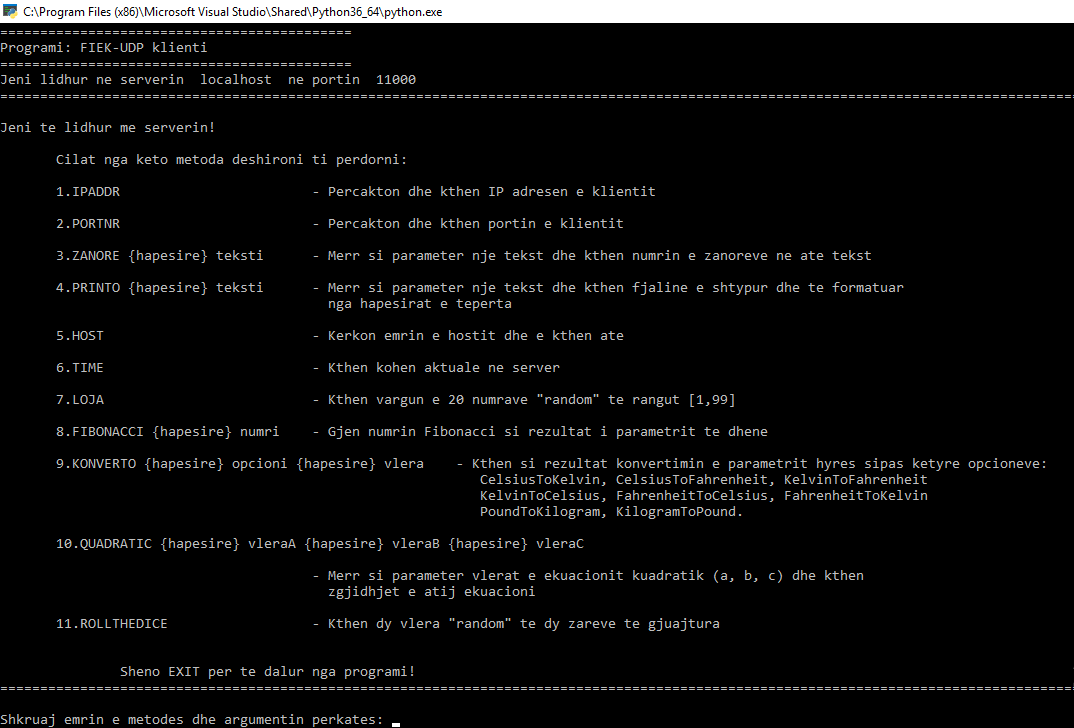
Me hapjen e programit FIEK-TCP serverit do të shfaqet kjo dritare:

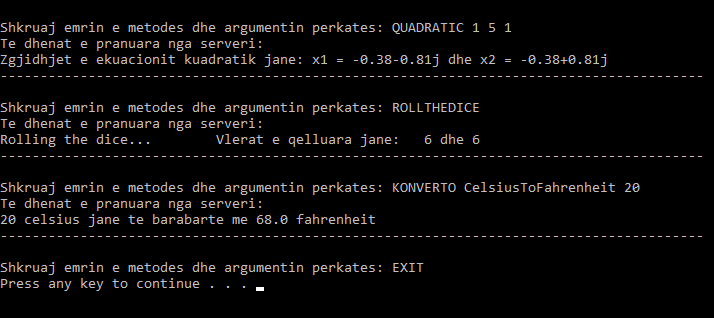


Gjithashtu edhe te UDP serveri do të shfaqet një dritare e ngjashme:

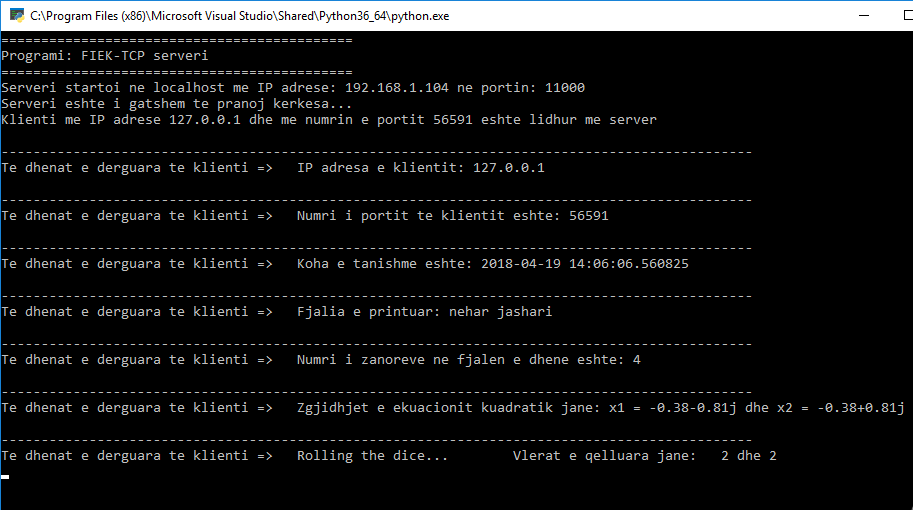


Menjëher pas hapjes së programit të klientit edhe në UDP klientin edhe në TCP klientin do të shfaqet një meny, ku tregohen metodat të cilat janë të qasshme dhe përdorimi i tyre:



Kurse këto janë pamjet (screenshots) të bëra gjatë testimit të metodave në programet e klientit:

Gjatë komunikimit klient-server apo gjatë dërgimit të përgjigjeve nga serveri tek klienti, në anën e serverit vazhdimisht do të shtypen të gjitha të dhënat që dërgohen te klienti. Këto janë pamjet që shfaqen te programi i serverit (e njëjta vlen për të dy programet TCP serveri dhe UDP serveri) :



# Trajtimi i Gabimeve

Gjatë shënimit të kërkesës që përmban emrin e metodës që dëshirojmë mund të bëhen gabime të ndryshme. Programet Server janë programuar në atë mënyrë që të trajtojnë këto gabime.

Trajtimi i gabimeve është bërë ashtu që nëse shënojmë emrin e metodës gabim do të paraqitet një mesazh nga serveri se kemi gabuar gjatë shënimit të këkesës. Ky gabim është evituar duke bërë krahasimin e stringut të kërkesës dhe emrit të metodës, nëse nuk përputhet me asnjë nga metodat paraqitet mesazhi:

connectionSocket.send("Kerkesa juaj eshte invalide, ju lutem provoni perseri!".encode())

Tjetri gabim që mund të ndodhë është nëse klienti jep më shumë argumente për metodën e caktuar se sa është e nevojshme. Kjo është bërë përmes një kushtëzimi që kërkon një numër të caktuar të argumenteve. Pasiqë është bërë ndarja e kërkesës sipas hapësirave “ “, ajo është ruajtur si array në variablën *kerkesaArray* që përmban emrin e metodës dhe argumentet përkatëse (ndodh që mos të kërkohet asnjë argument për ndonjë metodë të caktuar), atëherë bëhet pyetja se ashtë numri i argumenteve sa është kërkuar, nëse kemi dhënë më shumë apo më pak paraqitet një mesazh ku tregohet se kërkesa e klientit është invalide.

Si shembull kemi rastin e metodës QUADRATIC:

elif kerkesaArray[0]=="QUADRATIC":

if len(kerkesaArray) == 4:

connectionSocket.send(str(QUADRATIC(kerkesa[1], kerkesa[2], kerkesa[3])).encode())

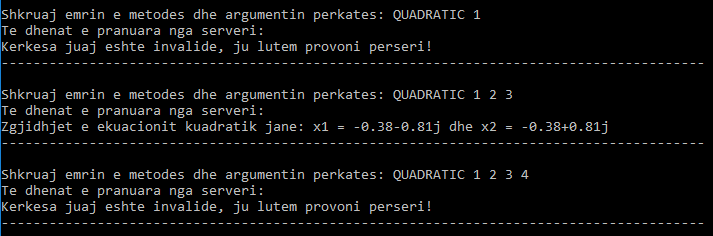
ShtypTeDhenat(str(QUADRATIC(kerkesa[1], kerkesa[2], kerkesa[3])))

else:

connectionSocket.send("Kerkesa juaj eshte invalide, ju lutem provoni perseri!".encode())

ShtypTeDhenat("ERROR")

Siç shihet kemi kërkuar vetëm 3 argumente, ku duke përfshirë edhe emrin e metodës së caktuar len(kerkesaArray) do të jenë katër. Si shembull kemi rastin e gabimit gjatë shkruarjes së metodës QUADRATIC, atëherë te klienti do të paraqitet në këtë mënyrë mesazhi:

Njëjtë funksionon edhe trajtimi i gabimeve për metodat tjera.

# Përfundimi

Gjatë punimit të këtij projekti, ku qëllimi kryesorë ka qenë të mësojmë se si shkon dizajnimi i një arkitekture klient-server, kam arritur të përforcoj aftësitë dhe njohuritë e mia për zhvillimin e aplikacioneve që funksionojnë me këtë arkitekturë dhe kam fituar përvojë në zhvillimin e aplikacioneve që punojnë përmes socket programming për komunikimin e programeve të ndryshme në rrjetë. Gjithashtu gjatë programimit të programeve klient dhe server kam kuptuar më mirë se si funksionojnë protokollet e shtresës së transportit TCP dhe ai UDP si dhe funksionimin e tyre në praktikë.

Janë zhvilluar nga dy programe klient dhe server, së pari sipas TCP protokollit pastaj sipas UDP protokollit. Komunikimi i programit të klientit dhe serverit është bërë përmes socket programming, ku socketat kanë shërbyer si “derë” në mes të serverit dhe klientit për shkëmbimin e të dhënave. Komunikimi i këtyre dy programeve ka qenë i mundur të bëhet sipas dy protokolleve të njohura: TCP ose UDP. Ku kanë ekzistuar dallime në mënyrën e qasjes në programim dhe në vendosjen e lidhjes ndërmjet socketit të klientit dhe atij të serverit.

Në këtë mënyrë ka qenë i mundur testimi i lidhjeve ndërmjet klientit dhe serverit në praktikë, duke krijuar metoda të ndryshme në server të cilat janë të qasshme nga klienti. Klienti ka pasur mundësi të dërgoj kërkesa të cilat përmbajnë emrin e metodave të përcaktuara dhe serveri në bazë të asaj kërkese nëpërmjet disa kushtëzimeve ka kthyer përgjigjen te klienti, ku përgjigja ka përfshirë zgjidhjen e asaj metode. Në këtë mënyrë funksionojnë programet e krijuara gjatë këtij projekti.

Pas zhvillimit të këtyre aplikacioneve dhe pas kësaj përvoje të fituar unë do të jem në gjendje të zhvilloj aplikacione të ndryshme me socket programming dhe jam më i përgaditur për tregun e punës jasht fakultetit.

# Referencat

* Programet e punuara gjatë ushtrimeve me asistentin Haxhi Lajqi.
* Libri: Computer Networking – A Top-Down Approach, by Jim Kurose and Keith Ross.
* <http://www.cs.dartmouth.edu/~campbell/cs60/socketprogramming.html>