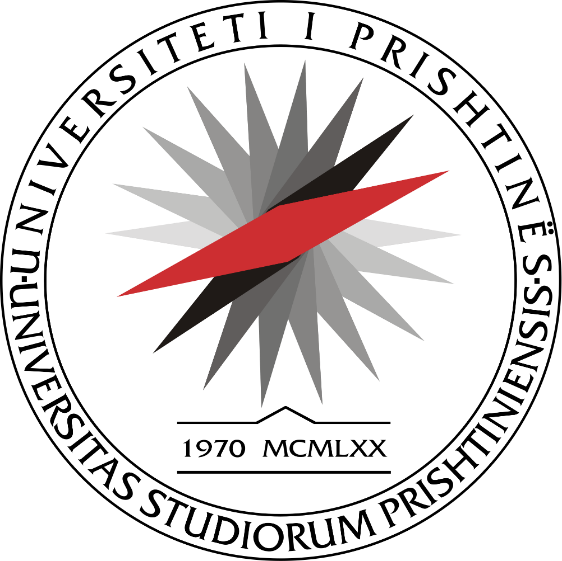
**UNIVERSITETI I PRISHTINËS “HASAN PRISHTINA”**

**FAKULTETI I INXHINIERISË ELEKTRIKE DHE KOMPJUTERIKE**



**Lënda:** Rrjeta kompjuterike

**Projekti 1:** Programimi me soketa

**Mentori:**

**Punoi:**

Enes Hasani

ID:190714100148

Msc. Haxhi Lajqi

Data 30/04/2021

**Projketi 1**

Vegla e perdorur per realizimin e porjektit të parë është një IDE e quajtur Spyder në të cilën kam programuar në gjuhën Python verzioni 3.9.4, ndersa testimi i programeve të zhvilluara është bërë në sistemin operativ Windows 10 64-bit Operating System.

Metodat e implementuara në tërë projektin

* IP
* NRPORTIT
* NUMERO
* ANASJELLTAS
* PALINDROM
* KOHA
* LOJA
* GCF
* KONVERTO
* PRIM
* FIBONACCI
* KONVERTO\_VALUTA
* THREADI\_KLIENT (thirret si thread)
* NUMEROZANORE (funksion ndihmes)
* COUNTNOTALETTER (funksion ndihmes)

Përmbajtja

[Hyrje 4](#_Toc69468779)

[1. TCP Protokolli 5](#_Toc69468780)

[**1.1. TCP Serveri** 5](#_Toc69468781)

[**1.1.1. Metoda IP** 9](#_Toc69468782)

[**1.1.2. Metoda NRPORTIT** 10](#_Toc69468783)

[**1.1.3. Metoda NUMERO** 10](#_Toc69468784)

[**1.1.4. Metoda ANASJELLTAS** 12](#_Toc69468785)

[**1.1.5. Metoda PALINDROM** 12](#_Toc69468786)

[**1.1.6. Metoda KOHA** 13](#_Toc69468787)

[**1.1.7. Metoda LOJA** 13](#_Toc69468788)

[**1.1.7. Metoda GCF** 14](#_Toc69468789)

[**1.1.8. Metoda KONVERTO** 15](#_Toc69468790)

[**1.1.9. Metoda PRIM** 16](#_Toc69468791)

[**1.1.10. Metoda FIBONACCI** 17](#_Toc69468792)

[**1.1.11. Metoda KONVERTO\_VALUTA** 18](#_Toc69468793)

[**1.2.TCP Klienti** 20](#_Toc69468794)

[2. UDP Protokolli 22](#_Toc69468780)

[**2.1.UDP Serveri** 22](#_Toc69468797)

[**2.2. UDP Klienti** 23](#_Toc69468798)

[3. Konkluzionet 24](#_Toc69468780)

**Hyrje**

Ky projekt ka fokus programimin duke perdorur soketa. Ky lloj programimi mundëson komunikimin mes dy pajisijeve fundore në rrjetë. Ne këtë projekt zhvillohen gjithsej katër programe, dy programe për shfrytëzues, të cilat ndërmjet veti dallojnë për nga lloji i soketave, përkatësisht lloji i protokollit i cili përdoret për bartjen e të dhënave ndërmjet shfryetëzuesit dhe serverit, dhe dy programe të cilat luajnë rolin e serverit e që kanë përgjegjësi t’i kthejnë përgjiegjje shfrytezuesit, përkatësisht të kryejne disa punë të thjeshta që i kërkon shfryetëzuesi si kontrollimi i një numri nëse është prim ose jo e kërkesa të ngjashme që do të paraqiten në vazhdim. Edhe këta dy programe dallojnë për nga lloji i soketave, përkatësisht për nga lloji i protokolleve që përdorin për komunikim. Njëri cift i programeve(TCPklienti dhe TCPserver) per komunikim(bartje të të dhënave) shfryetëzojnë protkollin TCP(Transmission Control Protocol), ndërsa cifti tjetër(UDPklienti dhe UDPserver) shfryetëzojnë protokollin UDP(User Datagram Protocol).

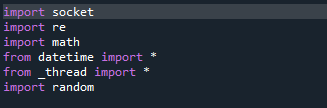
Transmission Control Protocol apo TCP është një bashkësi(set) rregullash që definon se si duhet të krijohet një lidhje dhe si duhet të zhvillohet ajo. Sipas këtij protokolli fillimisht kontrollohen pajisjet fundore a janë të gatshme për komunikim e në qoftëse ato janë të gatshme atëherë vendoset lidhja(ky proces njihet edhe si handshake), e vetëm pasi të jetë vendosur lidhja mund të shkëmbehen të dhënat mes tyre. Protokolli TCP është protokoll i besueshem(nuk ka humbje të të dhënave), ka kontrol të rrjedhjes të dhënave(floë control), andaj përdoret në aplikacionet që nuk tolerojnë humbje të të dhënave(p.sh. email). Ka congestion control(nuk e ngulfat marrësin, në rastin tonë serverin)

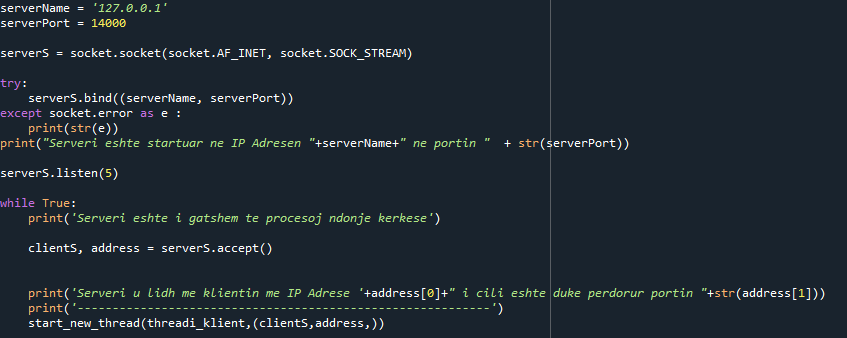
Përveq protokollit TCP kemi edhe protokollin UDP, i cili nuk konsiderohet i besueshem (mund të humbin të dhënat), po ashtu nuk garanton rrjedhje sekuenciale të të dhënave dhe nuk ndodh handshake(nuk vendos lidhjen, por drejtëpërdrejtë këmbehen të dhënat mes pajisjeve).Nuk ka congestion control(marrësi mund të ngulfatet) Përparësi e këtij protokolli është se është më i shpejtë, andaj përdoret për krijimn e lidhjeve me vonesë të ulët(kryesisht në lojra).

# **1. TCP Protokolli**

**1.1. TCP Serveri**

Le të shtjellojmë fillimisht kodin për zhvillimin e serverit që komunikon me klientin përmes protokollit TCP. Vlen të ceket se ky server është multithreaded, mund të pranojë kërkesa nga më shumë se një klient njëkohësisht. Po ashtu validimi i kërkesave bëhet vetëm nga ana e serverit.



****Fillimisht importojme librarite e nevojshme për zhvillimin e programit, **socket** për krijimn e soketave, **re** per shprehjet e rregullta(regular expressions), **math** per funksione matematike, **datetime** per datë , **\_thread** per të gjeneruar threada dhe **random** për operacione që lidhen me numra të rastit.

Pastaj caktojmë adresën dhe portin ku do të ngrisim(startojmë) serverin(variablat serverName dhe serverPort), pastaj permes funksionit **socket(socket.AF\_INET, socket.SOCK\_STREAM)** krijojmë një soket, ku parametri **socket**.**AF\_INET** tregon se për këtë soket po përdoren adresat e versonit IPV4 ndërsa parametri **socket.SOCK\_STREAM** tregon se soketi është i llojit TCP(këmbimi i informatave përmes soketave bëhet konform rregullave të protokollit TCP).

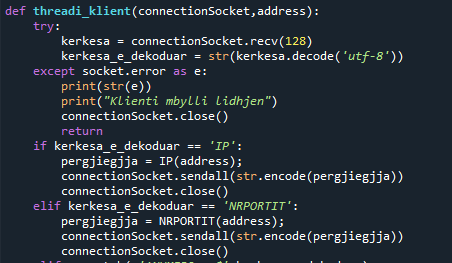
Hapi i ardhshëm është startimi i serverit, këtë e bëjmë përmes funksionit **bind()** të cilit si paramtera ia dërgojmë IP Adresen dhe portin ku dëshirojmë ta ngrisim serverin. Ky funksion mund të hedh exception për disa arsyje si psh porti që dëshirojmë ta ngrisim është i zënë, ose është bllokuar nga firëall, andaj është mirë të mbështjellet(ërapped) mbrenda bllokut try-except.

Funksioni i ardhshëm **listen()** e përcakton numrin e klientëve që mund të presin në rresht(queued clients) perderisa serveri është duke procesuar kërkesa(i zënë), nëse bëhen 5, klientit të gjashtë ia refuzon lidhjen(refuse connection), e mbron serverin nga crashi.

Serveri pret kërkesa nga shumë klient andaj funksionin **accept()** që vendos lidhjen mes klientit dhe serverit e fusim brenda një unaze while.Ky funksion kthen një soket(**clientS**) që reprezenton lidhjen dhe një dyshe(tuple) tjetër(**address**) që ka **IP adresën** e klientit dhe **portin** të cilin e përdor klienti për komunikim.

Që të pranojmë shumë kërkesa në kohë të njejtë duhet që lidhjet me secilin klient ti trajtojmë si thread-a(procese të lehta), andaj pasi të krijojmë lidhjen fillojmë një thread të ri përmes funksionit start\_neë\_thread(forma procedurale), përkatësisht funksionin **threadi\_klient** e trajtojmë si thread duke ia pasuar soketin që reprezenton lidhjen(clientS) dhe adresen(address). Shumë thread a mund të ekzekutohen “njëkohësisht”.

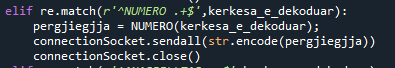
* 1. **Metoda threadi\_klient**



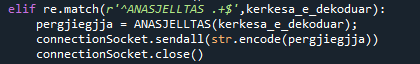
Metoda **threadi\_klient(connectionSocket, address)** siq mund të shihet edhe nga nënshkrimi pranon një soket që reprezenton lidhjen mes serverit(**connectionSocket)** dhe kilentit dhe adresen e ti**(address)**, përmbajtja e të cilës u cekë më lartë. **connectionSocket** është e nevojshme për të pranuar të dëna nga klienti, ndërsa **address** do të përdoret për t’ia dërguar si parametër dy funksioneve tjera që e shohi në vazhdim, ka funksione që nuk marrin si parametër asnjërën.

Kjo metodë fillimisht pranon kërkesën e klientit përmes funksionit **connectionSocket.recv(128 )** ku 128 e kufizon kërkesën e klientit në makimum 128 bajt. Pastaj e dekodon atë përmes funksionit **kerkesa.decode(‘utf-8’)** ku utf-8 tregon charachter set in e kerkeses sipas të cilit duhet të dekodohet. Këto funksione janë mbështjellur brenda bllokut try-except sepse pasi të jetë vendosur lidhja klienti mund ta ketë mbyllur programin dhe nuk kemi cfarë të pranojmë, kështu rreshti i kodit hedh një exception. E në rast se ndodh kjo ne e printojmë errorin dhe e mbylim soketin që reprezenton lidhjen me atë klient përmes (**connectionSocket.close())** dhe shuhet threadi i gjeneruar për atë klient. Në rastet kur pranohet kërkesa dhe është valide atëherë duhet t’a proesojmë kërkesën e klientit(të gjejmë përgjiegjjen) përmes njërit prej funksionve(duke i thirrur funksionet në bazë të kërkesës të klientit) që do të diskutohen në vazhdim, pastaj i’a dërgojmë përgjiegjjen klientit përmes funksionit **sendall(str.encode(pergjiegjja)),**  ku fillimisht e enkodojmepërmesfunksionit **str.encode(pergjiegjja).** Pas kësaj e mbyllim soketin që reprezenton lidhjen me atë klient(**connectionSocket.close()).** Në qoftëse është pranuar kërkesë por ajo nuk është valide atëherë useri lajmërohet përmes një mesazhi se kërkesa nuk ishte valide(figura më poshtë, blloku else).

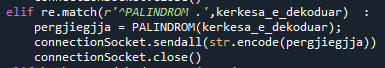
Validimi i shumicës së kërkeave të klientit bëhet përmes funksionit **match** të librarisë **re**, ku parametri i parë është shprehja e rregullt(RegEx) ndërsa parametri i dytë është stringu **kerkesa\_e\_dekoduar,** teksti i cili kontrollohet se a është i shkruar konform shprehjes së rregullt. Ndërsa për metodat IP,NRPORTIT,KOHA dhe LOJA, shikohet vetëm nëse përmbajtja e **kërkesa\_e\_dekoduar** është identike me emrin e tyre.



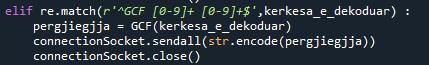
Para se të thirret funksioni **NUMERO** shikohet nëse **kerkesa\_e\_dekoduar** është e shkruar konform paternit **‘^NUMERO .+$’,** që do të thotë se fjalia duhet të filloi litralisht me “NUMERO “(vërejmë se ka hapësirë mas NUMERO) dhe të vazhdojë me cafrëdo karakteri, por që duhet të përsëritet 1 e më shumë herë.

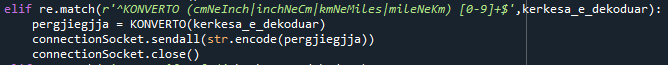
****

Shprehja e rregullt që validon kërkesën për funksionin **ANASJELLTAS** është komplet e njejtë me parapraken vetëm se fillimi i kërkesës duhet të jetë “**ANASJELLTAS**” e jo **NUMERO**.

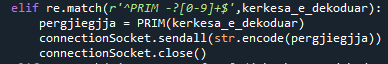


Shprehja e rregullt konform së cilës duhet të shkruhet **kerkesa\_e\_dekoduar** ashtu që të thirret funksioni **PALINDROM,** është plotësisht e njejtë me dy parapraket( sepse **.** është ekuivalente me **.+$** ), përveq se në këtë rast **kerkesa\_e\_dekoduar** duhet të filloi gjithsesi me fjalën **PALINDROM.**

****

Shprehja e rregullt konform së cilës duhet të shkruhet **kerkesa\_e\_dekoduar** ashtu që të thirret funksioni **GCF** e ka kuptimin se fjalia duhet patjetër të filloi me **GCF** të pasohet me një **hapsirë** pas hapësirës një numër një shifror apo disa numra njëshiforë të ngjitur të cilët i përkasin intervalit [0,9]( **[0-9]+** ) pastaj një **hapësirë** dhe përsëri shprehja **[0-9]+** të cilën e diskutuam, simboli **$** kushtëzon në këtë rast që fjalia të përfundoi me numër.

Shprehja e rregullt konform së cilës duhet të shkruhet **kerkesa\_e\_dekoduar** ashtu që të thirret funksioni **KONVERTO** e ka kuptim se fjalia duhet të filloi me “**KONVERTO** “(pas KONVERTO ka një hapësire), pjesa **(cmNeInch|inchNeCm|kmNeMiles|mileNeKm)** kushtëzon që duhët të pasohet me njëren prej alternativave në kllapa( **|** do të thotë **ose** ). Fjalia pasohet me një **hapësirë** pastaj kushtëzohet prejt pjesës **[0-9]+$** kuptimin e së cilës e diskutuam më herët.



Shprehja e rregullt konform së cilës duhet të shkruhet **kerkesa\_e\_dekoduar** ashtu që të thirret funksioni **PRIM** e ka kuptim se fjalia duhet të filloi me “**PRIM** “( pas PRIM ka një hapësirë ), pjesa tjetër kushtëzohet prejt **-?[0-9]+$** ku **-?** do të thotë që opsionalisht mundemi ta vendosim një **­–** apo jo(pra mund të vendosim edhe numra negativ)



Shprehja e rregullt konform së cilës duhet të shkruhet **kerkesa\_e\_dekoduar** ashtu që të thirret funksioni **FIBONACCI** e ka kuptim se fjalia duhet të filloi me “**FIBONACCI** “( pas FIBONACCI ka një hapësirë ), pjesa **[1-9][0-9]\*$** kushtëzon që kërkesa duhet të vazhdojë gjithsesi me një numër një shifrorë në intervalin **[1,9]**(kjo pasi nuk ka kuptim të themi 0 ****terma), pastaj opsionalisht mund të vendosim sado numra **[0-9]\*$**.

Shprehja e rregullt konform së cilës duhet të shkruhet **kerkesa\_e\_dekoduar** ashtu që të thirret funksioni **KONVERTO\_VALUTA** është komplet e njejtë me shprehjen për funksionin **KONVERTO** përveq se këtu kemi opsione tjera.

**1.1.1. Metoda IP**

Metoda IP është metoda e cila thirret kur shfryetëzuesi kërkon ta dijë IP Adresen e ti. Kështu pasi të jetë validuar kërkesa, thirret metoda IP nga funksioni **thread\_klienti.** Kjo metodë pranon si argument adresen e klientit(**address**) që si e dhënë paraqet një cift(tuple) ku anëtari i parë që gjendet në indeksin 0 paraqet IP Adresen(address[0]), andaj e kthen këtë anëtarë me një tekts informues se pergjiegjja është për IP Adresë të klientit.



 **Komunikimi klient-server**:

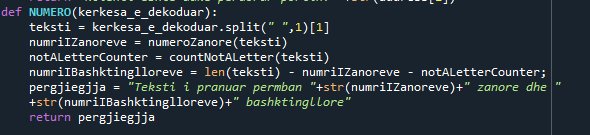
**1.1.2. Metoda NRPORTIT**

Metoda NRPORTIT është metoda e cila thirret kur klienti kërkon numrin e portit që po e shfrytëzon për komunikim. Pasi të jetë validuar kërkesa, thirret kjo metode nga funksioni **thread\_klienti.** Edhe kjo metode si argument pranon adresen e klientit(tuple), në indeksin 1 të së cilës gjendet porti i klientit, andaj ky funksion kthen këtë anëtarë(pra portin) të konvertuar në string me një tekst informues përpara siq mund të shihet nga figura.



**Komunikimi klient-server:**

**1.1.3. Metoda NUMERO**

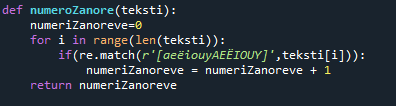
Metoda NUMERO, numeron numrin e zanoreve dhe bashkëtinglloreve në një tekst të dhënë nga klienti. Ajo si parametër pranon një string, përkatësisht kërkesën e klientit. Ne duhet ta largojmë fjalën “NUMERO”, për këtë fillimisht kërkesa ndahet në dy pjesë, në space-in e parë që hasët, përmes funksionit **split()**(**kerkesa\_e\_dekoduar.split(“ “,1)**) ku parametri i parë përcakton delimiterin(ndarësin) ndërsa parametri i dytë tregon sahere duhet të ndahet(1 here në këtë rast), pastaj e marrim vetëm anëtarin në indeksin 1 të listës që është kthyer (**kerkesa\_e\_dekoduar.split(“ “,1)[1])** që paraqet tekstin të cilit duhet t’ia numërojmë zanoret dhe bashkëtinglloret dhe e ruajmë në variablën **teksti**. Tekstit fillimisht i’a numërojmë zanoret përmes funksionit ndihmës **numeroZanore(teksti)** ku si parametër ia dërgojmë tekstin. Pastaj përmes funksionit tjetër ndhimës **countNotALetter(teksti)** i numërojmë karakteret që nuk janë shkronja, shenjat e pikësimit nëse ka, hapësirat etj. Së fundmi gjejmë numrin e bashkëtinglloreve(**numriIBashktinglloreve**) si ndryshimi mes gjatësisë së tërë tekstit dhe numrit të zanoreve dhe atyre karaktereve që nuk janë shkronja. Në fund me një string e kthejmë numrin e zanoreve dhe bashkëtinglloreve me një tekst informues.

**Komunikimi klient-server:**



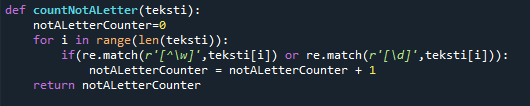
**Metoda numeroZanore**

Kjo metode si parametër pranon një string, përkatësisht pranon tekstin e klientit dhe ia numëron zanoret. Fillimisht marrim një variabël(**numeriZanoreve**) të cilën e perdorim për numërim të zanoreve dhe e inicializojmë me 0. Me një **for** loop iterojmë në një rang sa gjatësia e tekstit(**range(len(teksti))**)(dmth sa ka shkronja teksti) dhe përmes funksionit **re.match(r’[aeëiouyAEËIOUY]’,teksti[i]))** kontrollojmë secilën shkronjë se a është zanore, ku parametri i parë paraqet shprehje të rregullt(pattern) për të shikuar nëse karakteri **teksti[i]** është një zanore apo jo. Nëse është zanore atëhere e rrisim numriZanoreve për një, për ndryshe tentojmë të iterojmë edhe një herë. Kështu deri sa të përfundojë iterimi, pastaj e kthejmë numrin e zanoreve(return numeriZanoreve) e dhënë e cila është int.



**Metoda countNotALetter**

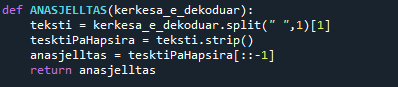
Edhe kjo metode si metoda e mëparshme si parametër pranon një string, përkatësisht tekstin e klientit.



Kjo metodë numëron karakteret që nuk janë shkronja. Fillimisht inicializojmë një variabël që e përdorim si numërues në 0(**notALetterCounter = 0**). Si në funksionin e mësipërm me një **for** loop iterojmë aq herë sa është gjatësia e tekstit, në secilin iterim përmes kushtit **if(re.match(r'[^\ë]',teksti[i]) or re.match(r'[\d]',teksti[i]))** kontrollojmë nëse karakteri **teksti[i]** nuk është ëord(**r'[^\ë]'** i bie as numër as shkonjë, nuk mjafton si kusht sepse numrat konsiderohn ëord) këtë e bënë pjesa e parë e kushtit, pastaj kontrollojmë nëse **teksti[i]** është numër, këtë e bënë pjesa e dyte kushtit. Në rast se plotësohen të dyja kushtet atëherë kemi një karakter që nuk është shkronjë, kështu rrisim **notALetterCounter** për një dhe iterojmë përderisa të kemi arritur tek shkronja e fundit. Në fund e kthejmë(return) **notALetterCounter** variabëltipi i të cilës është int.

**1.1.4. Metoda ANASJELLTAS**

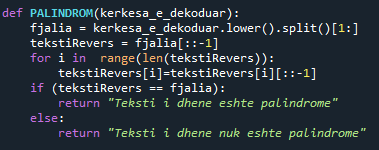
Kjo metodë është një metodë mjatë e thjeshtë, e cila si parametër pranon një string, konkretitsht pranon kërkesën e klientit, e cila paraprakisht është dekoduar në funksionin **threadi\_klient.** Sikurse te metoda **NUMERO** edhe këtu fillimisht kerkesa e dekoduar ndahet në space-in e parë, në dy vende dhe merret vetëm pjesa e tekstit që ka për tu kthyer revers(pjesa **ANASJELLTAS** nuk merret, formati i kërkesës mund të shihet më figurën më poshtë) dhe ruhet në variablën teksti, e tëra kjo përmes funksionit **split()**( **teksti = kerkesa\_e\_dekoduar.split(" ",1)[1**] ). Pastaj me funksionin strip() i largojmë hapësirat nga fillimi dhe fundi i tekstit, dhe në fund përmes slicing e kthejmë tekstin revers dhe e ruajmë me një variabël anasjelltas(**anasjelltas = tesktiPaHapsira[::-1]), tesktiPaHapsira[::-1]** me -1 indeksohet anëtari i fundit i një stringu(anëtari i parë prej mrapsht), dhe kjo shprehje e rrotullon stringun revers(anëtari i fundit i tij vendoset i pari , i parafundit vendoset i dyti e kështu me radhë ). Në fund e kthejmë variablën **anasjelltas** të tipit string.



**Komunikimi klient-server :**

**1.1.5. Metoda PALINDROM**

Metoda **PALINDROM** kontrollon se teksti që ka dërgur klienti është tekst palindrom(i cili kur rrotullohet revers mbetet i njejtë) apo jo. Edhe kjo metodë si paramtër merr një string, konkreitsht kërkesën e klientit e cila paraprakisht është dekoduar.



Kërkesën e dekoduar e kthejmë në shkonja të vogla përmes funksionit **lower(),** për shkak se zakonisht fjalitë fillojnë me shkronja të mëdha, e fjalia mund të jetë palindrome por shkronja e pare të jetë e madhe ndërsa e fundit e vogël e këto për procesorin janë të ndryshme(kanë vlera të ndryshme). Pastaj e ndajmë ku ka space-a me funksionin **split()** me crast formohet një listë e përbërë prej fjalëve të asaj fjalie, përmes slicing **[1:]** i marrim anëtarët që kanë indeksin 1 e tutje(fjalën ‘PALINDROM’ nuk e marrim), këtë listë e ruajmë në variablën **fjalia**, pastaj këtë listë e kthejmë revers(ua ndrrojmë fjalëve vendin) përmes slicing dhe e ruajmë te variabla **tekstiRevers**(**tekstiRevers=fjala[::-1]**)**.** Pastaj përmes një unaze **for** të gjithë anëtarët e listës i kthejmërevers(fjalët i kthejmë mbrapsht). Së fundi listën teksti revers e krahasojmë me listën që përmban fjalinë origjinale, i krahasojmë anëtarë për anëtarë(fjalë për fjalë) në kushtin

**if,** nëse të gjithë anëtarët korrespondues në indeksa janë identik atëherë fjalia është palindrome dhe kthejmë një string që e tregon këtë gjë si në figurë, nëse ndonjëri nuk është identik atëherë kthejmë një string që e tregon këtë gjë(si në figurë).

Komunikimi klient-server:

**1.1.6. Metoda KOHA**

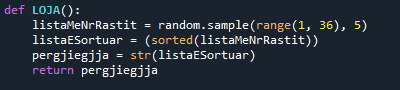
Metoda **KOHA** thirret kur useri kërkon kohën e serverit. Kjo metodë nuk merr asnjë parametër dhe përmes funksionit **noë** të librarisë **datetime** e kthen kohën e serverit. Funksionin **strftime("%d.%m.%Y %H:%M:%S %p")** e përdorim për ta formatuar rezultatin, ku në rastin tonë kemi vendosur formatin dita/muaji/viti/ora/minuti/sekondi, konvencionon(AM,PM). Para se të kthejmë rezultatin e konvertojmë në string përmes funksionit **str** ashtu që ta bëjmë të gatshme për enkodim**.**

****

 **Komunikimi klient-server:**

**1.1.7. Metoda LOJA**

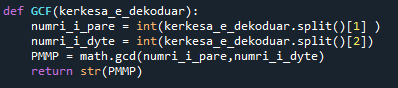
Ky funksion nuk merr asnjë parametër dhe si përgjiegjje kthen një string që ka pesë numra të rastit e që janë unik dhe janë në rangun [1,35].



Pesë numra të rastëisshëm i gjenerojmë përmes funksionit **sample** që i përket librarisë **random**(**random.sample()**) parametri i parë i të cilit është një funksion tjetër **range(1,36)** i cili përcakton intervalin(kufiri i sipërm nuk përfshihet dmth nuk përfshihet numri 36) në të cilin do të jenë numrat që do të gjenerohen, ndërsa **parametri i dytë(5)** përcakton se sa numra do t’i marrim nga ai interval. Ky funksion(**sample**)na kthen një listë të pa sortuar me 5 numra, listë të cilën e sortojmë përmes funksionit **sorted()** ku si parametër ia dërgojmë listën e pa sortuar(**sorted(listaMeNrRastit)**) dhe pastaj listën e sortuar e kthejmë në stringdhe e ruajmë në variablën **përgjiegjja**. Së fundmi përmes return e kthejmë përgjiegjjen.

 **Komunikimi klient-server**:

**1.1.7. Metoda GCF**

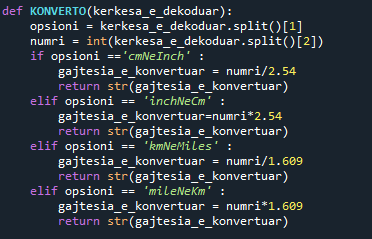
Metoda GCF(Greatest Common Factor) thirret kur klienti kërkon të gjejë pjestuesin më të madh të përbashkët të dy numrave(PMMP).

Si parametër merr një string, i cili përmabn kërkesën e klientit që është dekoduar paraprakisht, kërkesa është e formatit **“GCF NumriIParë NumriIDytë”,** andaj përmes metodës **split()** e ndajmë aty ku ka space me crast formohet listë me tre anëtarë, ku anëtari në indeksin 0 është emri i metodës, përkatësisht ‘GCF’, ndërsa anëtari në indeksin 1 është numri i parë prandaj variablës **numri\_i\_pare** ia shoqërojmë këtë anëtarë por të konvertuar në **int** ( **int(kerkesa\_e\_dekoduar.split()[1]** ) sepse fillimisht është **string**, anëtari në indeksin 2 është numri i dytë andaj këtë anëtarë ia shoqërojmë variablës **numri\_i\_dytë** të konvertuar në int ( **int(kerkesa\_e\_dekoduar.split()[2]** ). Përmes funksionit **gcd** të librarisë **math** gjejmë pjestuesin më të madh të përbashkët, duke ia dërguar këti funksioni numrat që i morëm nga kërkesa e klientit( **math.gcd(numri\_i\_pare,numri\_i\_dyte)** ), duhet të ceket se funksioni **gcd** pranon vetëm parametra të tipit **int**. Rezultatin e marrë e bëjmtë të gatshme për enkodim duke e konvertuar në string **str(PMMP)** dhe e kthejmë(return).

** Komunikimi klient-server**:

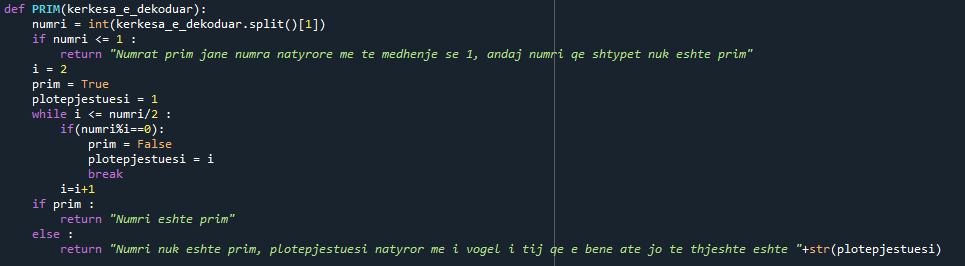
**1.1.8. Metoda KONVERTO**

Edhe kjo metodë pranon si parametër një string, përkatësishtë kërkesën e dekoduar të klientit. Kjo kërkesë ka formatin “KONVERTO Opsioni Numri”, andaj **kerkesa\_e\_dekoduar.split()[1]**(funksioni që u diskutua shumë herë deri më tani) na jep opsionin(nga 4 të mundshmet që shihen në fig) që ka zgjedhur klienti dhe e ruajmë në variablën **opsioni,** pastaj **kerkesa\_e\_dekoduar.split()[2]** na jep numrin që tregon gjatësinë që do të konvertohet, këtë e konvertojmë në **int** sepse fillimisht është **string**, pastaj përmes kushtëzimeve e kontrollojmë opsionin që ka zgjedhur klienti dhe vartësisht prej ti e bëjmë konvertimin(relacionet ndërmjet njësive të gjatësisë janë marrë nga google), dhe e kthejmë(return) gjatësinë e konvertuar, por fillimisht e bëjmë cast(konvertojmë) në string përmes funksionit **str**.(Cila njësi konvertohet në cilën shihet nga figura)



**Komunikimi server-klient:**

**1.1.9. Metoda PRIM**

Edhe kjo metodë si parametër pranon një string që paraqët kërkesën e klientit pasi të jetë dekoduar. Kjo metodë tregon se një numër i plotë a është numër i thjeshtë(prim) apo jo.

Kërkesa është e formatit “PRIM Numri”, ku Numri mund të jetë cfardo numri i plotë ashtu që kërkesa të konsiderohet valide. Fillimisht e marrim nga kërkesa numrin që kemi ta kontrollojmë e konvertojmë në **int**( **int(kerkesa\_e\_dekoduar.split()[1],** funksion që u diskutua shumë herë). Numrat e thjeshtë janë numrat natyrorë më të mëdhenjë se 1, andaj fillimisht shikojmë nëse numri është më i vogël se 1, nëse po atëherë shfaqet një mesazh që tregon këtë që cekëm(figura), nëse jo marrim një variabël **i** dhe e inicializojmë me vlerën **2**(numri i parë me të cilin do ta pjestojmë **numri**n e klientit nëse është më i madh se 3). Marrim një variabël **boolean prim=True**, e cila do të mbetet e tillë nëse **numri** është prim, ndërsa ndërron vlerën e vërtetësisë nëse **numri** nuk është prim.

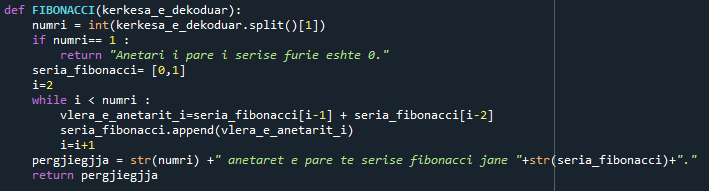
Pastaj marrim një variabël **plotëpjestuesi** që e inicializojmë me **1** e që do ta përdorim për të treguar plotëpjestuesin e parë që e bënë numrin e klientit jo-prim, nëse ai numër është jo-prim.

Që të caktohet se numri i klientit më i madh se 1 a është prim a jo mjafton të shikohet nëse numrat në intervalin **[2**,**numri/2]** paraqesin ndonjë plotëpjestues ose jo( numrat që janë më të mëdhenjë se **numri/2** jepin herësin në intervalin e pakufizuar (1,2) ), andaj në një unazë **while** vendosim kushtin **i <= numri/2**(ku **i** fillon nga 2). Pastaj me një **if** shikojmë nëse numri po plotëpjestohet me **i**( **if(numri%i == 0)** ), nëse po atëherë **numri** nuk është prim se ka plotëpjestues numrin **i** që nuk është 1 as vetvetja( **prim=False** ), **i**-ne e vendosim si plotëpjestues( **plotepjestuesi = i** ) dhe e përfundojmë unazën( **break** ), pasi që morrëm informatën se numri është prim.

Nëse **numri** nuk plotëpjestohet me **i** atëherë e rrisim numrin për **1** ( **i=i+1** ), shikojmë se mos e kemi kaluar gjysmën e **numri**t( **i <= numri/2** ), nëse jo e përsisim ciklin paraprak deri sa të përfundoi, unaza përfundon kur gjendet se numri nuk është prim( variabla **prim=True** ), ose kur **i** bëhet për **1** më e madhe se **numri/2** (që do të thotë se numri është prim, variabla **prim=True**). Në hapin e ardhshëm nëse variabla **prim** është **True( if prim** )kthehet(return) një string që tregon se numri i dhënë nga klienti është prim(figura). Nëse **prim** është **False** kthehet(return) një string që tregon se numri i dhënë nga klienti nuk është prim dhe jep plotëpjestuesin e parë që e bënë jo-prim.

 **Komunikimi klient-server**:

**1.1.10. Metoda FIBONACCI**

Edhe kjo metodë si parametër pranon një string, kërkesën e dekoduar të klientit. Ky funksion Kërkesa është e formatit “FIBONACCI Numri”, ku Numri mund të jetë një numër natyror(më i madh se 0). Funksioni i gjeneron aq anëtarë të serisë fibonacci sa është numri i dhënë nga klienti. Seria fibonacci përkufizohet si seri ku dy anëtarët e parë janë 0 dhe 1, ndërsa anëtarët e tjerë formohen si shumë e dy anëtarëve paraprak.

Fillimisht e marrim vetëm numrin nga kërkesa e dekoduar dhe e konvertojmë në **int** ( **int(kerkesa\_e\_dekoduar.split()[1]** funksion i diskutuar shumë herë deri më tani). Në vazhdim kontrollojmë nëse numri ka vlerë 1. Nëse po kthejmë(**return**) një string që tregon se anëtari i parë i serisë furie është 0(fig). Nëse ky kusht nuk plotësohet, formohet lista **seria\_fibonacci=[0,1]** që përmban dy anëtarët e parë të serisë. Formojmë një variabël **i=2**, e cila na hyn në punë kur **numri** që jep klienti është më i madh se 2(kur kemi për ti gjeneruar më shumë se 2 anëtarë). Me një unazë **while** iterojmë për derisa të kemi arrit tek anëtari i fundit që duhët gjeneruar nga ata anëtarë që na ka përcaktuar klienti (

**while i < numri “**<” përshkak se indeksi i listës **seria\_fibonacci** është zero-based ). Nëse plotësohet kushti **i < numri** e gjejmë vlerën e anëtarit **i**  si shumë të dy anëtarëve paraprak të serisë ( **vlera\_e\_anetarit\_i=seria\_fibonacci[i-1] + seria\_fibonacci[i-2]** ). Pastaj në listën **seria\_fibonacci** e shtojmë këtë anëtarë **seria\_fibonacci.append(vlera\_e\_anetarit\_i).** Pastaj e rrisim **i** për **1**(**i=i+1)** dhe e kontrollojmë nëse **i < numri** dhe vartësisht nëse plotësohet kushti apo jo, hymë në unazë e shtojmë edhe një anëtarë në listë ose dalim prej unaze sepse janë gjeneruar të gjithë anëtarët. Kështu gjenerohen të gjithë anëtarët. Pasi që të jenë gjeneruar të gjithë anëtarët e formojmë një variabël string **përgjiegjja** që tregon numrin e anëtarëve dhe anëtarët(listën e formuar), ky numër është i konvertuar në string po ashtu edhe lista ( figura ). Së fundi e kthejmë(return) variablën **përgjiegjja**( string ).

**Komunikimi klient-server:**

**1.1.11. Metoda KONVERTO\_VALUTA**

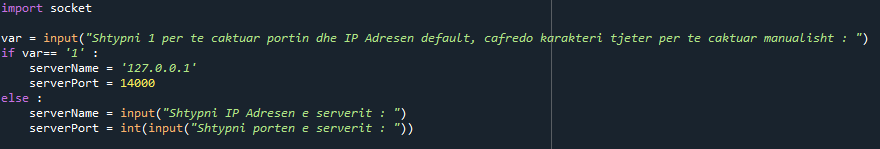
Këtë metodë nuk do ta shtjellojmë në detaje pasi që është analogji totale me metodën **KONVERTO**, vetëm se kjo metodë e bënë konvertimin prej një valute në një valutë tjetër. Metoda është implementuar për shkak se mund të jetë mjaftë praktike dhe e përdorshme. Edhe formati i kërkesës është komplet analoge me formatin e metodës **KONVERTO(KONVERTO\_ VALUTA Opsioni Numri).**

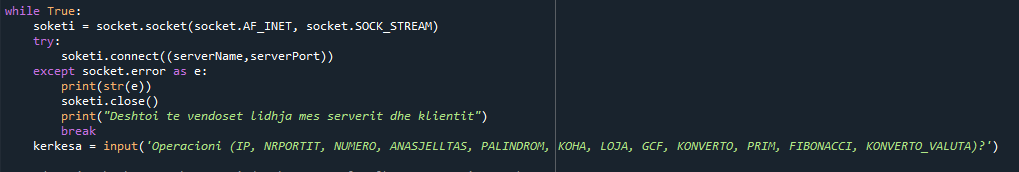
## 

## Njësoj sikur te metoda **KONVERTO** edhe këtu përmes kushtëzimeve kontrollojmë se cilin opsion ka zgjedhur klienti dhe vendosim se cilën valutë do ta konvertojmë në cilën. E kryejmë konvertimin(relacionet për valuta janë marrë nga google), më pastaj e kthejmë rezultatin duke e konvertuar para-prakisht në string me funksionin **str** njësoj sikur te metoda **KONVERTO.**(cilat valuta konvertohen në cilat shihet nga figura)

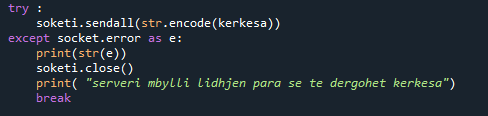
**Komunikimi klient-server:**

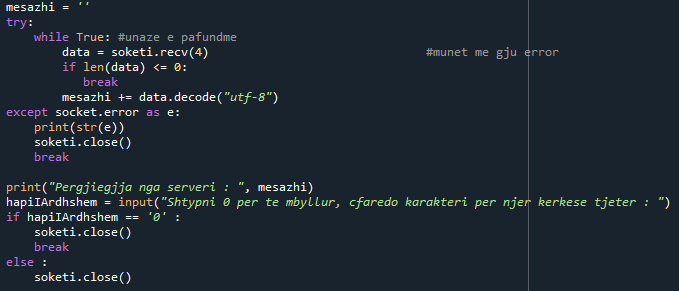
## **1.2.TCP Klienti**

Për kodimin e klientit na duhet të krijojmë soket, andaj fillimisht importojmë librarinë **socket**.

Pastaj pyesim se nëse shfryetëzuesi dëshiron ta caktojë portin dhe IP Adresën e default (14000 dhe 127.0.0.1) duke shtypur 1 apo ta caktojë manualisht duke shtypur cfaredo karakteri tjetër, pastaj kontollojmë se cfarë inputi ka dhënë dhe në vartësi të inputit e caktojmë **serverName( IP )** dhe **serverPort ( portin** )( që shihet edhe nga figura e mësipërme dhe është mjaftë triviale).

Unaza **while** është vetëm për t’ia mundësuar klientit të bëjë kërkesa vazhdimisht pa e mbyllur programin, unazë e cila përfundon kur klienti nuk dëshiron t’i bëjë më këresa serverit( do të shohim më vonë këtë). Sikur te serveri përmes funksionit **socket** krijojmë një soket të që përdor IP Adresa të verzionit IPV4 e që është i tipit TCP. Pastaj përmes funksionit **connect** që merr si parametra IP Adresen e serverit(**serverName)** dhe portin e serverit(**serverPort)** tentojmë të vendosim lidhjen mes klientit e serverit. Serveri mund të jetë i zënë(të refuzoj) lidhjen e në atë rast nuk mund të lidhemi me të( ka edhe raste tjera) andaj këtë funksion e mbështjellim(ërap) në bllokun try. Nëse na dështon lidhja e mbyllim soketin dhe e raportojmë errorin. Nëse është vendosur lidhja pyesim për kërkesën që do t’ia dërgojmë serverit.



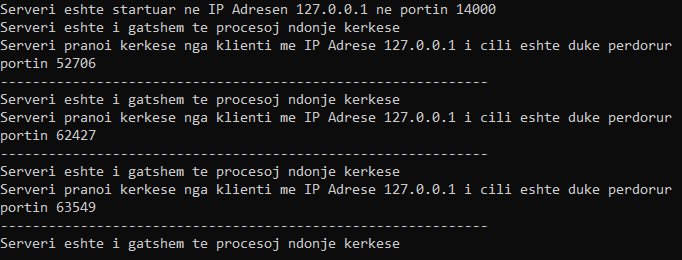
****Pasi të kemi marrë kërkesën nga shfryetëzuesi e enkodojmë atë **str.encode(kerkesa)** dhe tentojmë ta dërgojmë tek serveri përmes funksionit **sendall** të soketit që e krijaum( **soketi.sendall(str.encode(kerkesa))** ). Ka raste që pasi që është vënë lidhja serveri bjen ose diqka e tillë, andaj është përdorur blloku try. Nëse ndodh ndonjë error, raportohet ai, mbyllet soketi dhe dalim nga unaza. Nëse është dërguar kërkesa presim përgjiegjje.

Krijojmë një variabël **mesazhi** që do të përmbajë përgjiegjjen e kthyer nga serveri. Në unazënn **while** presim të dhëna nga serveri përmes funksionit **recv(4)** ku 4 tregon numrin maksimal të bajtave që pranohen njëkohësiht. Nëse nuk ka më informata për të pranuar **len(data) <=0** atëherë i kemi pranuar të gjitha dhe e përfundojmë unazën(**break)** nëse ka ende ato të dekoduara përmes funksionit **decode()** ia ngjesim mesazhit. Kjo metodë përdoret zakonisht kur ka fajlla të mëdhenjë dhe nuk mun t’i marrim përnjëherë( psh filma ). Unaza është vendosur në bllokun try se përgjatë kohës së marrjes së të dhënave serveri mund të ndalet ose diqka e tillë, e në këtë rast funksioni hedh error, nëse ka error e raportojmë atë në bllokun except dhe dalim prej unaze, nëse na janë kthyer komplet të dhënat e printojmë mesazhin e pyesim shfryetëzuesin se a dëshiron të bëjë një tjetër kërkesë apo jo, në vartësi të kësaj vendosim a ta përfundojmë unazën apo jo dhe në të dy rastet e mbyllim soketin.

# **2. UDP Protokolli**

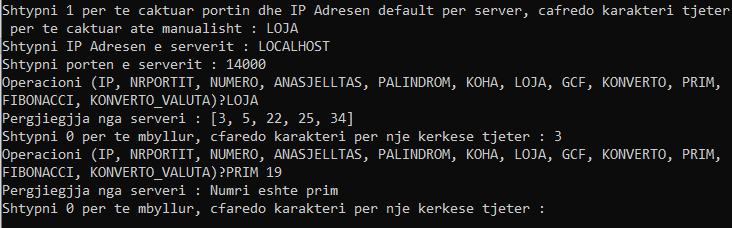
## **2.1.UDP Serveri**

Kodimi i UDP Serverit është po thujase i njejtë me atë të TCP Serverit. Andaj do të trajtojmë vetëm disa funksione që dallojnë nga TCP. Këtu kemi soketa që përdorin adresa të verzionit IPV4 ndërsa komunikojnë konform protokollit UDP, për krijmin e tyre përmes funksionit **socket()** në vend të **SOCK\_STREAM**, kemi **SOCK\_DGRAM** që është për soketa të tipit UDP. Tek ky server nuk përdoret funksioni **listen()** sepse protokolli UDP nuk ka congestion control(nuk kujdeset nëse marrësi ngulfatet). Nuk ka handshake, nuk vendoset lidhja mes dërguesit e marrësit andaj nuk përdoret funksioni **accept().** Për të pranuar kërkesa përdoret funksioni **recvfrom()** që kthen kërkesën e klientit, të enkoduar dhe adresen e klientit. Për të dërguar përgjiegjjen përdoret funksioni **sendto()** që si parametër pranon përgjigjjen e enkoduar dhe adresën e klientit.

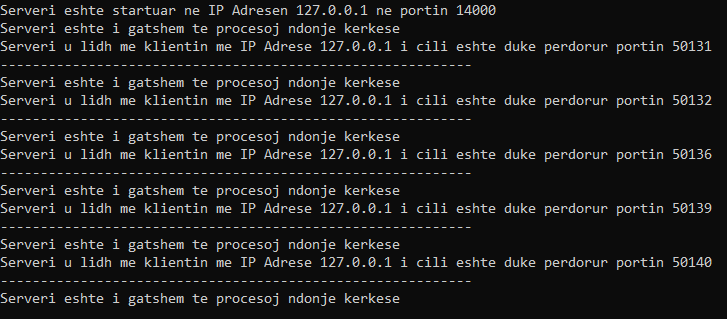
Disa pamje të serverit(multithreaded) kur në të janë të kycur disa klientë.

**2.2. UDP Klienti**

Të njejtat dallime që vlejnë mes TCP serverit dhe UDP serverit vlejnë edhe në mes TCP serverit dhe UDP serverit. Te funksioni socket në vend të **SOCK\_STREAM** kemi **SOCK\_DGRAM**, për të njejtat arsye të lartëcekura më herët. Nuk përdoret funksioni **connect()** sepse të protokolli UDP nuk ka handshake, sikurse te serveri për dërgimin e të dhënave përdoret funksioni **sendto(),** po ashtu njësoj si te serveri për të pranuar të dhëna përdoret funksioni **recvfrom().** Për shkak të ngjashmërisë me TCP klientin nuk do ta shtjellojmë rresht për rresht kodin.

****Një pamje e UDP klientit:

# **3. KONKLUZIONET**

Nga shpjegimet e mësipërme dhe figurat e paraqitura mund të vërehet se përsa i përket metodave. Po ashtu nga komunikimet klient-server shihet se funksiojnë edhe klientët edhe serverët. Na ka mbetur të shohim se TCP Serveri është multithreaded. Për këtë po shfaq vetëm një figurë të TCP serverit i cili është multithreaded server.