برنامج محادثة باستخدام البروتوكول TCP

ابي نزار ابراهيم 2097

ملخص: في عالم الشبكات، أي اتصال هو عبارة عن اتصال خادم\عميل. غرفة الدردشة هي مساحة للأشخاص في جميع أنحاء العالم للتواصل مع بعضهم البعض حول مواضيع مختلفة. لغة بايثون تعتبر من أسهل اللغات وأبسطها وجاءت لتلبية مختلف المتطلبات في مختلف المجالات البرمجية حيث تحتوي مكتبات عديدة لبرمجة تطبيقات الشبكة والويب، ويمكن باستخدام لغة البرمجة بايثون تجريد طبقات الشبكة في النموذج المرجعي للشبكة TCP/IP والتعامل مع كل طبقة على حدى والربط بينها، في هذا المشروع سوف نعمل على طبقة النقل في النموذج المرجعي للشبكة لإنشاء غرفة محادثة بين عدة مستخدمين وسنتعامل مع بروتوكول TCP باستخدام لغة البرمجة بايثون ومكتباتها وتوابعها لأداء المهمة المطلوبة. سيكون لدينا خادم واحد يستضيف الدردشة والعديد من العملاء الذين يتصلون به ويتواصلون مع بعضهم البعض.

الكلمات المفتاحية: TCP، مقابس

Chat Application Using TCP

Abstract: In the world of networking, any connection is a client/server connection. Chat room is a space for people all over the world to communicate with each other on different topics. The Python language is considered one of the easiest and simplest languages and came to meet the various requirements in various programming fields, as it contains many libraries for programming network and web applications. Using the Python programming language, it is possible to abstract the network layers in the TCP/IP network reference model and deal with each layer separately and connect them, in this The project We will work on the transport layer in the network reference model to create a chat room between multiple users and we will deal with the TCP protocol using the programming language Python and its libraries and dependencies to perform the required task. We will have one server hosting the chat and many clients connecting to it and communicating with each other.

Keywords: TCP, sockets

مقدمة:

بمجرد توجيه البيانات عبر الشبكة وتسليمها إلى مضيف معين، يجب تسليمها إلى المستخدم أو العملية الصحيحة. أثناء تحرك البيانات لأعلى أو لأسفل طبقات TCP/IP، هناك حاجة إلى آلية لتسليمها إلى البروتوكولات الصحيحة في كل طبقة. يجب أن يكون النظام قادرا على الجمع بين البيانات من العديد من التطبيقات في عدد قليل من بروتوكولات النقل، ومن بروتوكولات النقل إلى بروتوكول الإنترنت. يسمى الجمع بين العديد من مصادر البيانات في دفق بيانات واحد تعدد الإرسال. يجب إلغاء تعدد البيانات التي تصل من الشبكة: مقسمة للتسليم إلى عمليات متعددة. لإنجاز هذه المهمة، يستخدم IP أرقام البروتوكولات لتحديد بروتوكولات النقل، وتستخدم بروتوكولات النقل أرقام المنافذ لتحديد التطبيقات. يتم حجز بعض أرقام البروتوكولات والمنافذ لتحديد الخدمات المعروفة. الخدمات المعروفة هي بروتوكولات الشبكة القياسية، مثل FTP وTelnet، والتي يشيع استخدامها في جميع أنحاء الشبكة.

طبقة النقل هي جزء من نموذج شبكاتTCP/IP ، وتسمى أحيانا بنية الشبكات. تحتوي على مجموعة شاملة من الوظائف التي تصف كل ما هو مطلوب لشبكة الكمبيوتر للعمل. طبقة النقل مسؤولة عن الاتصال المنطقي بين التطبيقات التي تعمل على مضيفين مختلفين، وبالتالي توفير الخدمات لبروتوكولات طبقة التطبيق على طبقة أعلى من نموذج شبكةTCP / IP . على الرغم من وجود العديد من بروتوكولات طبقة النقل، فإن البروتوكولين الأكثر استخداما هما بروتوكول التحكم في الإرسال (TCP) وبروتوكول مخطط بيانات المستخدم(UDP) . توفر هذه البروتوكولات وظائف مختلفة لمتطلبات التطبيق المختلفة. سنعمل في هذا المشروع على بروتوكول طبقة النقل TCP.

أهمية البحث:

إنشاء برنامج محادثة بين عدة أشخاص على مبدأ غرف المحادثة وبالتالي تطبيق الاستفادة من مقرر برمجة الشبكات فيما يخدم أغراض في الحياة الواقعية.

منهجيات البحث:

1- بروتوكول TCP:

بروتوكول التَّحكُّم بالنقل (TCP) هو بروتوكول هامٌّ من بروتوكولات الشبكة، ويستخدم في إرسال البيانات عبر الشبكات. والبروتوكول في عالم الشبكات هو مجموعة من القواعد والإجراءات التي تتحكَّم في كيفية تنفيذ نقل البيانات، حيث يقوم أي شخص حول العالم بالشيء نفسه، بغض النظر عن الموقع أو البرامج أو الأجهزة المستخدمة.

يسمى بروتوكول التحكم بالنقل أو بروتوكول التحكم بالإرسال وهي ترجمة الجملة الإنجليزية (Transmission Control Protocol)أو اختصاراً يسمّى"TCP" ، وهو البروتوكول المستخدم لإدارة الاتصالات، ونقل و تدفق البيانات بشكل فعَّال، ويعتبر بروتوكول "TCP" بروتوكول معتمد به من قبل"Internet Society" ، وهي الهيئة المسؤولة عن تحديد أسس الاتصال عبر الإنترنت. بروتوكول TCP هو بروتوكول يدعم عمليات الاتصال التي ينبغي أن يتم تحديد (تأسيس) المسار فيها بين المصدر (source) والوجهة(Destination) ، قبل البدء في إرسال البيانات، وهو ما يجعلها تتميَّز بالموثوقية فيما يتعلَّق بالاتصال من طرف إلى آخر. طبقا لـ وثيقة RFC-793 فإن بروتوكول TCPيعمل ما بين طبقات المستوى العلوي وطبقة بروتوكول الإنترنت، طبقا لـ نموذج OSI المعياري، وتحديداً في طبقة النقل Transportation Layer – (الطبقة الرابعة). بروتوكول TCP هو المسؤول عن تدفُّق البيانات بين جهازين، ودائما يوجد الجهاز الذي يقوم بإرسال البيانات ويسمى بـ المصدر أو المرسل، والجهاز الآخر الذي يستقبل البيانات ويسمّى بـ الوجهة أو المستقبل، وقد يستخدم برتوكول TCP لنقل بيانات بين برنامجين على جهاز واحد، ولكن للتَّبسيط سوف نفترض نقل البيانات بين جهازين متصلين بشبكة واحدة.

عندما يقوم الجهاز المصدر بتحديد البيانات المراد إرسالها إلى الجهاز المُستقبل، يقوم بروتوكول TCP بتجزئة هذه البيانات إلى عدة أجزاء (أو مقاطع)، كل مقطع من هذه المقاطع يتكوَّن من سلسلة أو مجموعة Sequence – من الوحدات الأصغر حجما، ومقدار كل وحدة منها واحد بايت.

وكما تعلمنا في تمثيل البيانات باستخدام النظام الثنائي، فإن كل بايت (Byte) مكوَّن من 8 بت، والـ بت (Bit) هي الوحدة الأصغر، والتي تحتوي على 0 أو 1. وطبقا لـ نموذج الاتصال المعياريOSI ، فإنَّ كل طبقة أو مستوى من المستويات السبعة تقوم بإضافة مجموعة من البيانات تسمى بـ مقدمة أو ترويسة Header إلى كل مقطع من البيانات Headerالتي يتمّ التعامل معها، فإنَّ بروتوكول TCP يقوم أيضا بإضافة مجموعة من البيانات، لكلِّ مقطع أو بايت يريد إرساله.

كما ذكرنا أن بروتوكول "TCP" يتعامل مع طبقة بروتوكول الإنترنت IP – في الجهاز المستقبل، لذلك يقوم المصدر بإرسال مقاطع "TCP" بمثابة وحدات بيانات ذات تنسيقIP ، حتى يتمكن المستقبل من التعامل مع هذه البيانات، بالإضافة إلى مقدمة مقطع "TCP" والتي تسمى بـ(Header) ، مقدمة مقطع بيانات "TCP" تتكون من التالي:

TCP Header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size														
16	128	Checksum																	Urgent pointer (if URG set)														
20 ...	160 ...	Options (if *data offset* > 5. Padded at the end with "0" bytes if necessary.) ...																															

الشكل 1 ترويسة البروتوكول TCP

الجدول السابق يتكوَّن من مجموعة من الصفوف (Rows) بحيث يتكوَّن كل صف من 32 بت، وكل صف ينقسم إلى أربعة أقسام، كل قسم مكوّن من 8 بت أو بايتByte ، وتبدأ كل بايت من اليسار بترتيب البت المكونة لها من 7 إلى صفر، وهو ما يعرف بـ ليتل إنديان، وهو خارج نطاق هذا المقال، وتفصيل البيانات كالتالي:

حقل "Source port" :وهو عبارة عن حقل مكون من 16 Bit-بصيغة Unsigned Binary Number ممَّا يعني أن الرقم الموجود بداخله يمكن أن يكون بين 0, $2^{16}-1$، أي أنَّه يتكون من رقم من صفر إلى 65,535، وهو يمثل الرقم الخاص بمُنفذ اتصال المصدر، أو باختصار هو رقم الـ Port الخاص بالجهاز مرسل البيانات.

حقل "Destination port": هو أيضا عبارة عن حقل مكوَّن من 16 بت، ويمثل الرقم الخاص بمنفذ اتصال الوجهة (الجهاز مستقبل البيانات).

حقل "Flags": بالنظر إلى الجدول السابق سوف تجد ستة حقول ملونة باللون الأخضر، كل حقل منهم يحتوي على 1 بت، بحيث تكون قيمته إما 0 أو 1، وكل حقل منهم له اسم مكتوب بداخله، وله دلالة إذا كانت قيمته تساوي واحد، وسوف نتناولهم بالتفصيل ولكن إذا ما قمت بالإشارة لـ Flag XXX أو بت التحكمXXX ، فإنه يعني قيمة البت الموجودة في الحقلXXX

حقل "Sequence Number": وهو عبارة عن حقل مكون من 32 بت، وتعتمد القيمة بداخله على قيمة الـ Flag (SYN).

- إذا كانت قيمة الـ Flag (SYN) تساوي 1: فإن قيمة الحقل هي الرقم المتسلسل الأبتدائي للمقطع الذي يتم إرساله ويسمّى بـ(initial sequence number (ISN ، وفي هذه الحالة الرقم المتسلسل لأول بايت سوف يصبحISN+1
- أما إذا كانت قيمة الـ Flag (SYN) تساوي 0: فيعني ذلك أن قيمة الحقل هي الرقم المتسلسل للبايت الذي يتم إرساله.

حقل"Acknowledgment Number" : هو عبارة عن حقل مكون من 32 بت، وقيمته تعتمد على الـ Flag (ACK)، فإذا كانت قيمة بت التحكم ACK تساوي 1، فذلك يعني أن قيمة هذا الحقل تحتوي على الرقم المسلسل التالي المتوقع استقباله، ودائما ما يتم إعداد بت التحكم بعد إنشاء الإتصال.

حقل "Data Offset": وهو حقل مكون من 4 بت، أي أن قيمته يمكن أن تكون من 0 إلى 15، ولكنَّه له حد أدنى وحد أقصى من 5 إلى 15، أي أن أقل قيمة يمكن أن يحتويها هذا الحقل هي 0101 وهي قيمة الرقم 5 بالنظام الثنائي، وهذا الحقل يعبر عن عدد الكلمات (Words) في مقدمة مقطع الـTCP ، والمقصود بالكلمة هنا أن كل 32 بت تسمى كلمة، أو بطريقة مبسطة إذا نظرت إلى الجدول السابق سوف تجد أنه يحتوي على خمسة صفوف، قبل الصف الأخير، وبعد نهاية الجدول يتم دمج البيانات التي نريد إرسالها، وسوف تلاحظ أن الصف الأخير في الجدول والمُسمى بـ Options، ليس له قيمة محدَّدة كباقي الصفوف، فقد يحتوي على أي عدد من الـ بت، وبالتالي يتم تحديد عدد الصفوف حتى يتمكَّن الجهاز المستقبل من تحديد البيانات وفصلها عن المقدمة الخاصة بـ بروتوكول الـTCP

حقل "Reserved" هو حقل مكون من 6 بت، دائما قيمتهم تساوي الصفر، إلَّا أنَّ في بعض التحديثات لمعيار الـ TCPتم إضافة بعض بتات التحكُّم(Flags) ، ولكنَّها إلى الآن قيد التجريب، لذلك لم أقم بإضافتهم على هذا النموذج.

حقل "Flags / Control Bits" وهو حقل مكون من 6 بت، في الجدول تم تمثيلهم باللون الأخضر، وكل بت منهم يعبر عن حالة خاصة في مقدمة مقطع البيانات، وتفصيلهم كالتالي

Urgent – URG يتم استخدامه لقراءة بيانات الحقل Urgent Pointer

Acknowledgment – ACK كما تعرفنا على استخدامه مع الحقلAcknowledgment

Push – PSH يستخدم لتنفيذ الأمر PUSH

Reset – RST يستخدم لإعادة ضبط الاتصال عن طريق الأمر RESET

Synchronize – SYNيستخدم لمعرفة قيمة الحقلSequence Number ، كما تعرفنا عليه مسبقا .

FINISHED – FINويعني أنه لا يوجد المزيد من البيانات الصادرة من المرسل، وبالتالي يتم إنهاء الاتصال.

حقل "Window Size" وهو حقل مكون من 16 وحدة بت، بحيث تحتوي قيمته على عدد البايت (Bytes) في المقطع (Sequence) الذي يتم إرساله حاليا.

حقل "Checksum" وهو حقل مكون من 16 بت، ويستخدم بهدف كشف الأخطاء من عدمها أثناء نقل البيانات، حيث يستخدم بعض العمليات الحسابية باستخدام "Source Address" و "Destination Address" بالإضافة إلى طول مقدمة بروتوكول الـ TCP من أجل الكشف عن الأخطاء التي قد تحدث أثناء عملية نقل البيانات.

حقل "Urgent Pointer" هو حقل مكون من 16 بت، وتحتوي قيمته على عنوان وحدة البايت العاجلة في التسلسل (المقطع من البيانات)، ولا يتم النظر إلى محتوى هذا الحقل إلا إذا كانت قيمة الـ Flag (URG) تساوي 1، بصورة مختصرة، أحيانا يتم تحديد جزء من البيانات التي يتم إرسالها على أنها بيانات عاجلة، أي يجب أن يتم إرسالها قبل أي بيانات أخرى، وفي هذه الحالة يتم استخدام بت التحكم"URG" ، مع "Urgent Pointer" ليتم تحديد البيانات العاجلة.

حقل :"Options" هذا الحقل قد يحتوي على بيانات ما بين 0 و 320 بت، أي رقم يقبل القسمة على 32، فكما أشرنا إليه في الحقلData Offset ، فهو يعتبر الحقل الأخير الذي يتم إضافته من قبل بروتوكول الـTCP ، وهو حقل اختياري، أي أنه يمكن عدم تحديد أي بيانات إضافية، ويُكتفي بوضع بايت واحد مكوَّن من أصفار ليكون فاصل بين البيانات التي يتم إرسالها والمقدمة (Header) الذي تناولناه حتى الآن.

يتعامل بروتوكول TCP على الجانب الأول (المصدر) مع العمليات الخاصة بالمستخدمين أو طبقة التطبيقات، وعلى الجانب الآخر (الوجهة) يتعامل مع بروتوكول من المستوى السفلي مثل بروتوكول الإنترنت .(IP) ويمكن تقسيم خطوات عمل بروتوكول التحكُّم بالنقل إلى ثلاثة مراحل أساسية تبدأ بأن يقوم الجهاز المصدر بإرسال رسالة إلى الجهاز الوجهة مفادها أنه يريد أن يرسل له بعض البيانات، ثم ينتظر المصدر حتى يتم الرد عليه بالموافقة من الوجهة، ولا يتم البدء في إرسال البيانات حتى يتم الرد بالموافقة، وهنا تبدأ عملية إرسال البيانات، وتعدُّ هذه هي المرحلة الثانية، وبانتهاء إرسال كافة البيانات تبدأ المرحلة الثالثة والتي تهتم بانهاء الاتصال، والتَّخلّي عن الموارد التي سبق حجزها لإتمام عملية نقل البيانات. وعادة ما يتم إدارة بروتوكول التحكُّم بالنقل من قبل نظام التشغيل، حيث يوفِّر كل نظام تشغيل واجهة

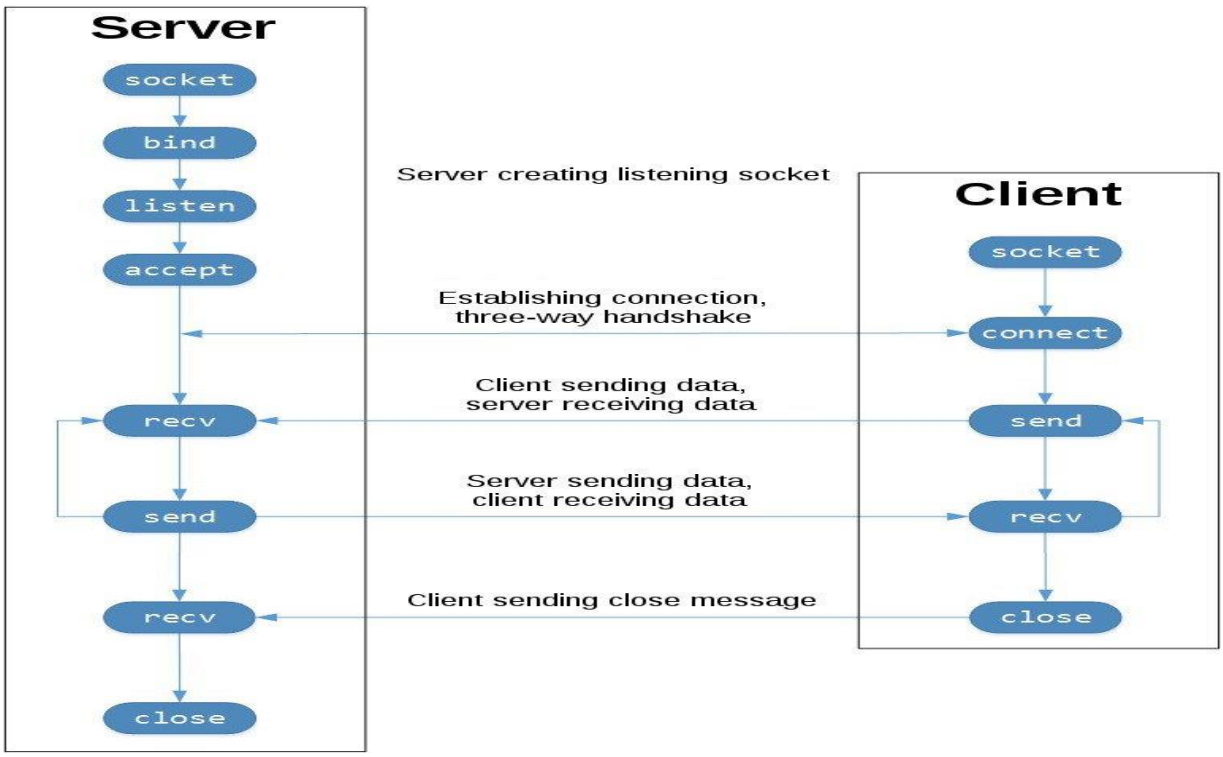
للتعامل مع هذا البروتوكول، وعادة ما تسمَّى هذه الواجهة بـ "Internet Socket API" بحيث تتيح للمبرمجين إنشاء برامج بهدف التعامل مع بروتوكول التحكم بالنقلTCP

2- المقابس:

بشكل عام، المقابس هي نقاط نهاية داخلية مصممة لإرسال البيانات واستقبالها. ستحتوي الشبكة الواحدة على مقبسين، واحد لكل جهاز أو برنامج متصل. هذه المقابس هي مزيج من عنوان IP ومنفذ. يمكن أن يحتوي جهاز واحد على عدد "n" من المقابس استنادا إلى رقم المنفذ المستخدم. تتوفر منافذ مختلفة لأنواع مختلفة من البروتوكولات

لقد جاء هذا المصطلح " socket" بعدة طرق.

1. يمكن تعريف المآخذ على أنها نقاط نهاية اتصال بين جهازي كمبيوتر يتم تحديدهما بواسطة عنوان IP ورقم منفذ.
2. يمكن تعريف المقابس أيضًا على أنها تجريد للبرامج يستخدم لتمثيل "محطات" اتصال بين جهازين.
3. يمكن تعريفه أيضًا على أنه فكرة مجردة يتم توفيرها لمبرمج تطبيق لإرسال البيانات أو استقبالها إلى عملية أخرى.
4. المقبس هو الباب بين عملية التطبيق و.TCP
5. المقبس هو واجهة بين التطبيق والشبكة.



الشكل 2 آالية عمل TCP

الجزء العملي:

لنبدأ الآن بتطبيق الخادم أولاً. لهذا سنحتاج إلى استيراد مكتبتين ، وهما socket و threading. سيتم استخدام الأول للاتصال بالشبكة والثاني ضروري لأداء مهام مختلفة في نفس الوقت.

```
import socket
import threading
```

المهمة التالية هي تحديد بيانات الاتصال الخاصة بنا وتهيئة المقبس الخاص بنا. سنحتاج إلى عنوان IP للمضيف ورقم منفذ مجاني لخادمنا. في هذا المثال ، سنستخدم عنوان المضيف المحلي (127.0.0.1) والمنفذ 55555. المنفذ

في الواقع غير ذي صلة ولكن عليك التأكد من أن المنفذ الذي تستخدمه مجاني وغير محجوز. إذا كنت تقوم بتشغيل هذا البرنامج النصي على خادم فعلي، فحدد عنوان IP للخادم باعتباره المضيف.

```
host = '127.0.0.1'
port = 55555

# Starting Server
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((host, port))
server.listen()

# Lists For Clients and Their Nicknames
clients = []
nicknames = []
```

عندما نحدد المقبس الخاص بنا ، نحتاج إلى تمرير بارمترين يحددان نوع المقبس الذي نريد استخدامه. يشير الأول (AF_INET) إلى أننا نستخدم مقبس إنترنت بدلاً من مقبس يونكس. المعامل الثاني يرمز إلى البروتوكول الذي نريد استخدامه يشير SOCK_STREAM إلى أننا نستخدم TCP وليس UDP.

بعد تحديد المقبس ، نربطه بمضيفنا والمنفذ المحدد عن طريق تمرير مجموعة تحتوي على كلتا القيمتين. ثم نضع خادمنا في وضع الاستماع ، بحيث ينتظر اتصال العملاء. في النهاية ، نقوم بإنشاء قائمتين فارغتين ، سنستخدمهما لتخزين العملاء المتصلين وألقابهم لاحقًا.

```
def broadcast(message):
    for client in clients:
        client.send(message)
```

نحدد هنا وظيفة صغيرة ستساعدنا في بث الرسائل وتجعل الكود أكثر قابلية للقراءة. ما تفعله هو مجرد إرسال رسالة إلى كل عميل متصل وبالتالي في قائمة العملاء. سوف نستخدم هذه الطريقة في التوابع الأخرى.

الآن سنبدأ بتنفيذ الوظيفة الرئيسية الأولى. ستكون هذه الوظيفة مسؤولة عن التعامل مع الرسائل من العملاء.

```
def handle(client,i):
    while True:
        try:
            # Broadcasting Messages
            message = client.recv(1024).decode()
            if message=="CLOSE123123":
                index = clients.index(client)
                clients.remove(client)
                client.close()
                nickname = nicknames[index]
```

```
                broadcast('{} left!'.format(nickname).encode())
                nicknames.remove(nickname)
                print('{} left!'.format(nickname))
                break

            broadcast(message)
        except:
            index = clients.index(client)
            clients.remove(client)
            client.close()
            nickname = nicknames[index]
            broadcast('{} left!'.format(nickname).encode())
            nicknames.remove(nickname)
            print('{} left!'.format(nickname))
            break
```

كما ترى ، تعمل هذه الوظيفة في حلقة while. لن يتوقف الأمر ما لم يكن هناك استثناء بسبب خطأ ما أو إذا تم ارسال الرسالة التالية: CLOSE123123. الوظيفة تقبل العميل كمعامل. في كل مرة يتصل فيها العميل بخادمنا نقوم بتشغيل هذه الوظيفة له ويبدأ حلقة لا نهاية لها.

ما يفعله بعد ذلك هو تلقي الرسالة من العميل (إذا أرسل أيًا منها) وبثها إلى جميع العملاء المتصلين. لذلك عندما يرسل أحد العملاء رسالة ، يمكن لأي شخص آخر رؤية هذه الرسالة. الآن إذا كان هناك خطأ ما في الاتصال بهذا العميل لسبب ما ، نقوم بإزالته واللقب الخاص به ، ونغلق الاتصال ونبث أن هذا العميل قد ترك الدردشة. بعد ذلك نكسر الحلقة وتنتهي هذه الثريد. لقد انتهينا تقريبًا من الخادم ولكننا بحاجة إلى وظيفة أخيرة.

```
while True:
    # Accept Connection
    client, address = server.accept()
    print("Connected with {}".format(str(address)))

    # Request And Store Nickname
    nickname = client.recv(1024).decode()
    nicknames.append(nickname)
    clients.append(client)

    # Print And Broadcast Nickname
    print("Nickname is {}".format(nickname))
    broadcast("{} joined!".format(nickname).encode())
    client.send('Connected to server!'.encode())

    # Start Handling Thread For Client
    thread = threading.Thread(target=handle, args=(client,address))
    thread.start()
```

عندما نكون مستعدين لتشغيل خادمنا ، سنقوم بتنفيذ هذه التعليمات . تبدأ حلقة لا نهاية لها والتي تقبل باستمرار اتصالات جديدة من العملاء. بمجرد اتصال العميل ، ينتظر الرد (الذي نأمل أن يحتوي على اللقب) ويلحق العميل

باللقب المعني بالقوائم. بعد ذلك نقوم بطباعة هذه المعلومات وبثها. أخيرًا ، نبدأ ثريد جديد يقوم بتشغيل وظيفة المعالجة التي تم ذكرها مسبقاً لهذا العميل المعين.

لاحظ أننا نقوم دائمًا بترميز وفك تشفير الرسائل هنا. والسبب في ذلك هو أنه يمكننا فقط إرسال البايت وليس السلاسل. لذلك نحتاج دائماً إلى تشفير الرسائل (على سبيل المثال باستخدام utf-8) ، عندما نرسلها ونفك تشفيرها ، عندما نستقبلها.

تنفيذ العميل

الخادم عديم الفائدة إلى حد كبير بدون عملاء يتصلون به. لذلك نحن الآن بصدد تنفيذ عميلنا. لهذا ، سنحتاج مرة أخرى إلى استيراد نفس المكتبات.

```
import socket
import threading
```

أولى خطوات العميل هي اختيار اسم مستعار والاتصال بخادمنا. سنحتاج إلى معرفة العنوان الدقيق والمنفذ الذي يعمل فيه خادمنا. بعدها نرسل الاسم الى الخادم ونستقبل رسالة منه.

```
nickname = input("Choose your nickname: ")
print("WHEN YOU WANT TO CLOSE ENTER CLOSE123123")
# Connecting To Server
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(('127.0.0.1', 55555))
client.send(nickname.encode())
msg=client.recv(1024).decode()
print(msg)
```

نحن نستخدم وظيفة مختلفة هنا. بدلاً من ربط البيانات والاستماع، نقوم بالاتصال بخادم موجود.

الآن، يحتاج العميل إلى ثريدين يتم تشغيلهما في نفس الوقت. الأول سيتلقى البيانات باستمرار من الخادم والثاني سيرسل رسائلنا الخاصة إلى الخادم. إذن ، سنحتاج إلى وظيفتين هنا. دعونا نبدأ مع الجزء المتلقي.

```
def receive():
    while True:
        try:
            msg=client.recv(1024).decode()
            print(msg)
        except:
            client.close()
            break
```

مرة أخرى لدينا حلقة لا نهاية هنا. يحاول باستمرار تلقي الرسائل وطباعتها على الشاشة. في حالة وجود خطأ ما ، نغلق الاتصال ونقطع الحلقة. الآن نحن فقط بحاجة إلى وظيفة لإرسال الرسائل ونحن على وشك الانتهاء.

```
def write():
    while True:
        s=input('')
        if s=="CLOSE123123":
            client.close()
            break
        message = '{}: {}'.format(nickname, s)
        try:
            client.send(message.encode())
        except:
            client.close()
            break
```

يتم تشغيله أيضاً في حلقة لا نهاية والتي تنتظر دائماً إدخالًا من المستخدم. بمجرد حصوله على بعض، إذا كان الإدخال CLOSE123123 نقوم باغلاق السوكيت أما غيرها فإنه يجمعه مع اللقب ويرسله إلى الخادم. هذا هو. آخر شيء يتعين علينا القيام به هو بدء سلسلتين تقومان بتشغيل هاتين الوظيفتين.

```
receive_thread = threading.Thread(target=receive)
receive_thread.start()

write_thread = threading.Thread(target=write)
write_thread.start()
```

التشغيل والنتائج:

نقوم بتشغيل الخادم وبعدها نقوم بتشغيل العميل A والعميل B ونرى الخرج:

```
Connected with ('127.0.0.1', 54146)
Nickname is ehab
Connected with ('127.0.0.1', 54147)
Nickname is amal
```

الشكل 3خرج الخادم عند بداية الاتصال

```
Choose your nickname: ehab
WHEN YOU WANT TO CLOSE ENTER CLOSE123123
ehab joined!
Connected to server!
amal joined!
```

الشكل 4خرج العميل A عند بداية الاتصال

```
Choose your nickname: amal
WHEN YOU WANT TO CLOSE ENTER CLOSE123123
amal joined!
Connected to server!
```

الشكل 5 خرج العميل B عند بداية الاتصال

```
Choose your nickname: ehab
WHEN YOU WANT TO CLOSE ENTER CLOSE123123
ehab joined!
Connected to server!
amal joined!
hi
ehab: hi
amal: welcome
CLOSE123123

Process finished with exit code 0
```

الشكل 6المحادثة والخروج من قبل العميل A

```
Choose your nickname: amal
WHEN YOU WANT TO CLOSE ENTER CLOSE123123
amal joined!
Connected to server!
ehab: hi
welcome
amal: welcome
ehab left!
```

الشكل 7 المحادثة عند العميل B

```
Connected with ('127.0.0.1', 54184)
Nickname is ehab
Connected with ('127.0.0.1', 54185)
Nickname is amal
ehab left!
```

الشكل 8خرج الخادم

المراجع:

1- Stevens, W. (1997). TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms.
2- Forouzan, B. A. (2002). *TCP/IP protocol suite*. McGraw-Hill Higher Education.
3- https://www.geeksforgeeks.org/simple-chat-room-using-python/