

Creating Chat Application with socket

الطالبة ديمه محمد داود

الطالب حُنين عدنان إبراهيم

الملخص

بعد التطور التقني الكبير الذي شهده هذا القرن، وزيادة استخدام الهواتف والأجهزة المحمولة، أصبحت لغات البرمجة حاجة ملحة لاستخدامها في التطبيقات التي تخدم الحياة اليومية للمستخدمين، كل هذه الأسباب أدت لتمكين دور المبرمج لخدمة معطيات العصر. تعتبر لغة برمجة python من أكثر اللغات التي ساهمت بشكل فعال في تطبيقات الذكاء الصناعي والتعلم العميق وبذلك أعطت البشرية الأمل في انتشار حقبة الروبوتات وتعلم الآلات.

مقدمة

الشبكة الحاسوبية *Computer Network*

شبكة الحاسوب هي نظام لربط جهازين أو أكثر باستخدام إحدى تقنيات نظم الاتصالات

- تستخدم من أجل تبادل المعلومات والموارد والبيانات المتاحة للشبكة فيما بينها (مثل الآلة الطابعة أو البرامج التطبيقية)
- تسمح بالتواصل المباشر بين المستخدمين

ظهرت الشبكات نظراً للحاجة إلى الاتصال بين الأفراد في الأماكن المتباعدة وتبادل الخدمات المختلفة، وساعد في ذلك التطور العلمي والتقني. لذلك دعت الحاجة إلى إنشاء نظام يمكن للمستخدم المشاركة في مصادر المعلومات مثل ربط فروع الشركة المنتشرة في عدة مناطق بنظام واحد

البروتوكول *Protocol*

البروتوكول هو عبارة عن مجموعة القواعد التي تحدد كيف يمكن لأجهزة الكمبيوتر أن تتفاهم مع بعضها البعض عبر الشبكة التي تتواجد عليها.

البروتوكول يختلف باختلاف نوع الخدمة التي تقدمها الشبكة، وعلى سبيل المثال فإن الإنترنت قد تأسس على مجموعة البروتوكولات التي تكون عائلة واحدة هي **TCP/IP**

أهداف البحث:

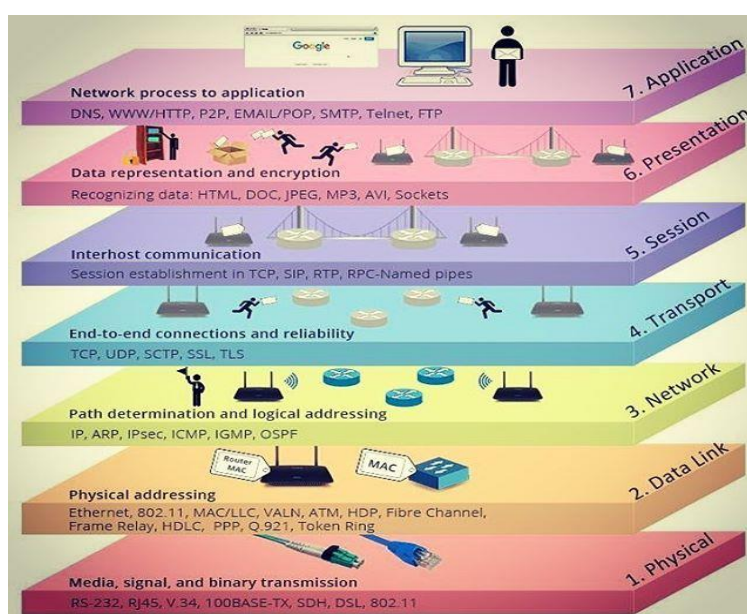
الهدف من هذا البحث هو إنشاء غرفة دردشة في الزمن الحقيقي تسمح بالتواصل بين الأشخاص باستخدام لغة برمجة **python** والمكتبة **Tcp Sockets**، مع تصميم واجهة **GUI** خاصة بالدردشة باستخدام أداة **tkinter**، وذلك من أجل ضمان التواصل الفعال عن بعد بين المستخدمين.

غرفة الدردشة هي مساحة للأشخاص في جميع أنحاء العالم للتواصل مع بعضهم البعض حول مواضيع مختلفة. يمكن أن تتراوح الموضوعات من أي شيء مثل تطوير التطبيق إلى أي وصفات يتم إعدادها في المطبخ. غرف الدردشة هي منصات رائعة لتعلم كيفية التواصل. في هذه المقالة، سأقدم لك كيفية إنشاء غرفة محادثة صغيرة باستخدام المفاهيم الأساسية مع المقابس.

نماذج الطبقات

OSI (Open Systems Interconnection)

يعرض مرجع OSI على شكل سبع طبقات، تتكون بشكل عمودي، أعلاه "طبقة التطبيقات" وأسفله "الطبقة الفيزيائية"



الشكل 1-1: نموذج OSI

طبقات نموذج OSI

Application layer

هذه الطبقة المسؤولة عن التطبيقات مثل البرامج التي يتعامل معها المستخدم مثل المتصفح، يحتاج الى البرامج مثل برامج التصفح **Google Chrome** أو **Mozilla**، أو عندما يريد رفع ملفات إلى السيرفر أو سحب ملفات يحتاج أيضاً إلى برامج النقل مثل **FTP Client**

البروتوكولات التي تعمل في طبقة التطبيقات

SNMP , DNS , FTP , LDAP , LMP , NTP , HTTP , DHCP , Open VPN , SMTP , POP3 , IMAP , WAE , WAP , SSH, Telnet , SIP , PKI , SOAP , rlogin , TLS / SSL .

Presentation layer

هذه طبقة العرض المسؤولة عن تهيئة البيانات والتفريق ما بين كل نوع من البيانات، وتقوم هذه الطبقة بالتشفير وفك التشفير للبيانات.

البروتوكولات التي تعمل في طبقة العرض

JPEG , MPEG , ASCII , EBCDIC , HTML , AFP , PAD , NDR , RDP , PAD , AVI .

Session layer

هي الطبقة المسؤولة عن جلسة العمل وعن إدارة وفتح وإغلاق أي اتصال ما بين المستخدمين

البروتوكولات التي تعمل في الطبقة المسؤولة عن جلسة العمل

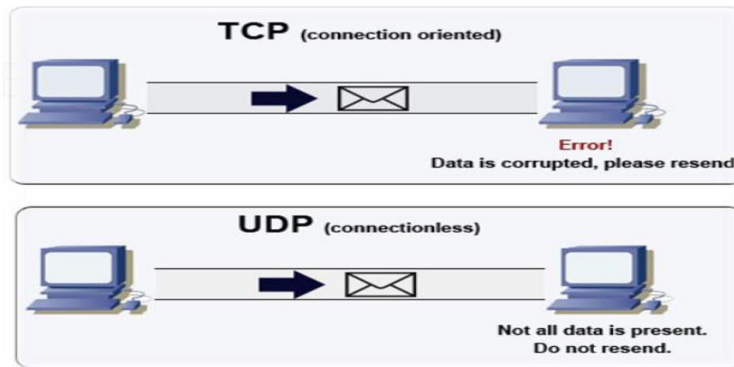
SAP, RTP, NFS, SQL, RPC, NETBIOS NAM, NCP, SOCKETS, SMB, NETBEUI.

Transport layer

هذه الطبقة المسؤولة عن نقل وإدارة البيانات وتحديد نوع البيانات المرسل والمستقبل وبعده تقوم بتحديد نوع البروتوكول المناسب في عملية إرسال البيانات.

البروتوكولات التي تعمل في الطبقة المسؤول عن نقل وإدارة البيانات

TCP, UDP

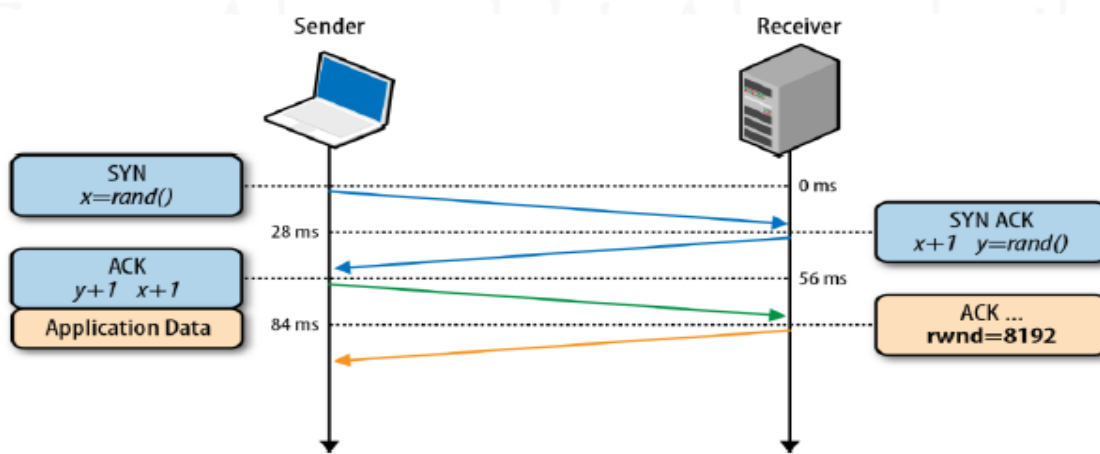


الشكل 2-1: TCP, UDP

TCP: Transmission Communication Protocol

هو بروتوكول يتحقق من وصول البيانات المرسله وهو يحتاج إلى جلسة عمل ما قبل إرسال البيانات إلى الحاسوب الآخر وتسمى هذه العملية **Three Way handshake** ومن خلال هذه العملية يقوم ببناء جلسة عمل ما بين الجهاز المرسل والمستقبل.

يعبر الشكل التالي عن كيفية إرسال واستقبال البيانات ما بين الحواسيب وكيفية بناء الاتصال ما بينهم في بروتوكولات TCP



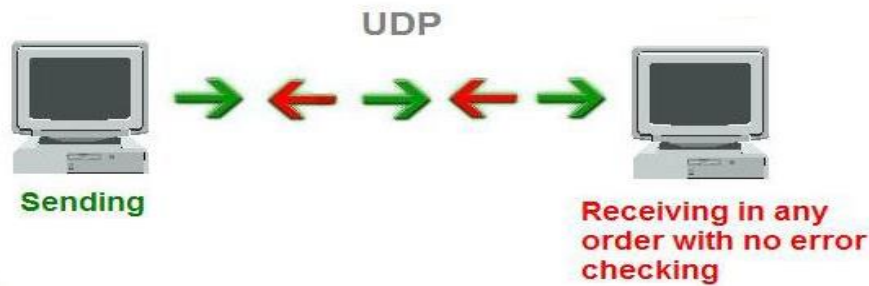
الشكل 3-1: آلية عمل بروتوكول النقل TCP

UDP: User Datagram Protocol

بروتوكول بيانات المستخدم يقوم بتقسيم الرسالة إلى عدة أجزاء ويقوم بإرسال هذه الأجزاء إلى المستقبل مع وضع عنوان المستقبل في كل جزء من أجزاء الرسالة طبع، ويرسل هذه الأجزاء في فضاء الانترنت مما قد يجعل جزء يصل قبل جزء آخر فهذه الأجزاء لا تسلك نفس الطريق في الشبكة.

إن هذا البروتوكول لا يقدم أي ضمان لوصول الحزمة بشكل صحيح أو كامل لأن هدف هذا البروتوكول هو إيصال الحزمة بشكل سريع وفي أقرب وقت ممكن، وليس هدفه إيصال الحزمة بشكل صحيح والتأكد من وصولها بسلامه كما يفعل بروتوكول ال TCP

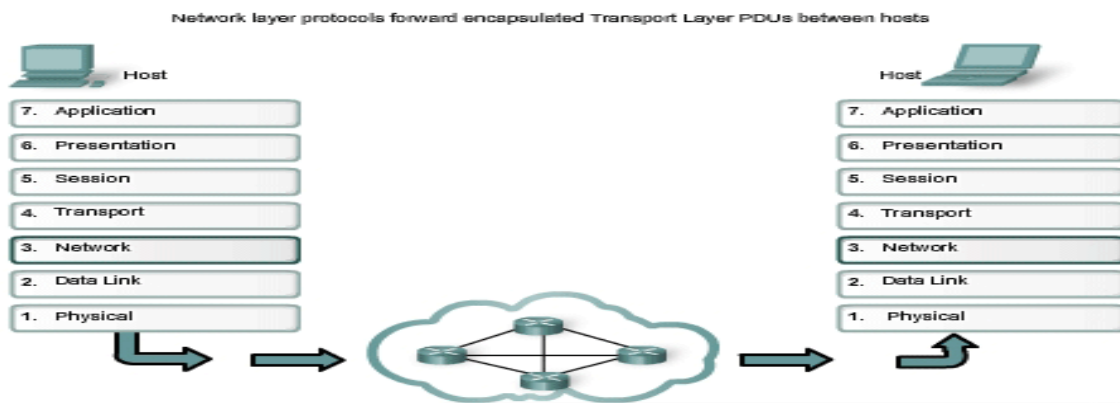
يوضح الشكل التالي كيفية إرسال البيانات في بروتوكول UDP بشكل مباشر من دون جلسة عمل مسبقة



الشكل 4-1: آلية عمل بروتوكول UDP

Network layer

هذه الطبقة المختصة في الشبكة وهي المسؤولة عند إدارة الرزم التي تأتي من طبقة النقل.



الشكل 5-1: Network Layer

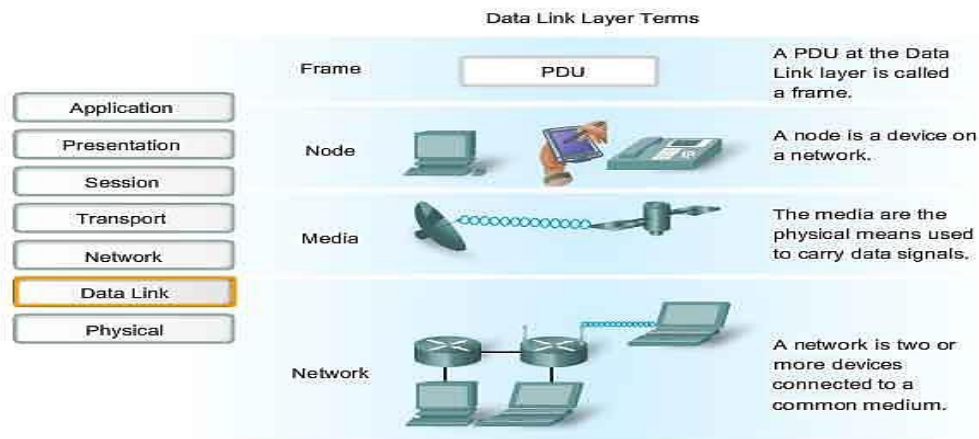
البروتوكولات التي تعمل في طبقة الشبكة

IPv4, IPv6, IPx, ICMP, IPsec, IGMP, CLNP, EGP, EIGRP, IGRP, IPx

SCCP, GRE, OSPF, ARP, RIP, Routed-SMLT

Data link layer

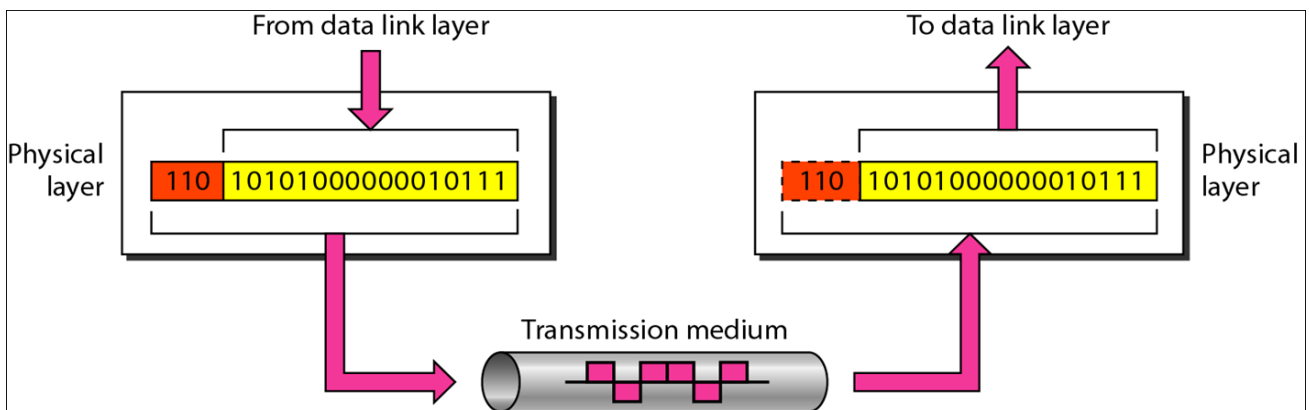
طبقة ربط البيانات أو طبقة ربط المعطيات طبقة ربط البيانات هي الطبقة التي يتم فيها تجهيز البيانات من أجل تسليمها للشبكة أي تحويل البت الخام إلى جدول من الإطارات.



الشكل 6-1 : Data Link Layer

Physical layer

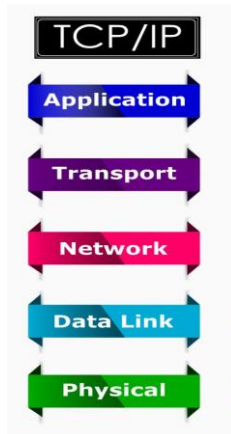
آخر مرحلة تمر فيها البيانات بشكل نهائي ليتم إيصالها للجهاز المطلوب



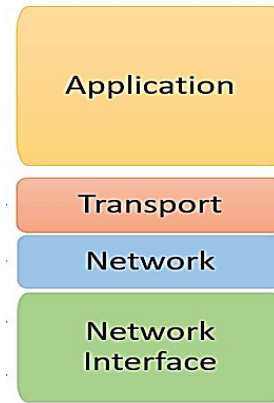
الشكل 71 :- Physical Layer

TCP/IP

يوضح الشكل الآتي نموذج TCP/IP ونموذج OSI، حيث تم اختصار نموذج TCP/IP إلى أربع أو خمس طبقات



الشكل 91:- TCP/IP Updated



الشكل 81:- TCP/IP

طبقات نموذج TCP/IP

يحتوي نموذج TCP/ IP على أربع طبقات

طبقة التطبيقات وطبقة النقل وطبقة الإنترنت وطبقة الوصول إلى الشبكة.



الشكل 101:- طبقات TCP/IP

طبقة التطبيقات



الشكل (10): طبقة التطبيقات

تعالج طبقة التطبيقات في نموذج TCP/ IP البروتوكولات عالية المستوى ومسائل العرض والترميز والتحكم في الحوار.

يحتوي TCP/ IP على بروتوكولات لدعم نقل الملفات والبريد الإلكتروني وتسجيل الدخول عن بعد، بالإضافة إلى تطبيقات أخرى.

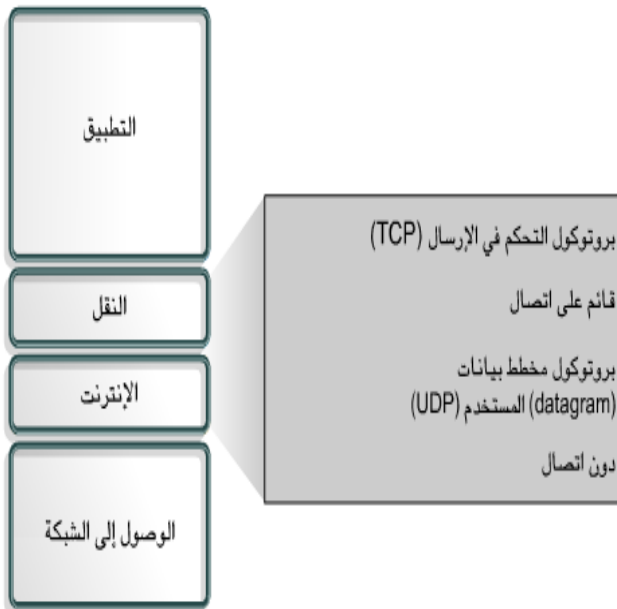
البروتوكولات التي تعمل على طبقة التطبيقات

طبقة النقل

توفر طبقة النقل خدمات النقل من المضيف المصدر إلى المضيف الوجهة.

تُنشئ طبقة النقل اتصالاً منطقياً بين المضيف المرسل والمضيف المستقبل.

تتضمن خدمات النقل جميع الخدمات التالية:

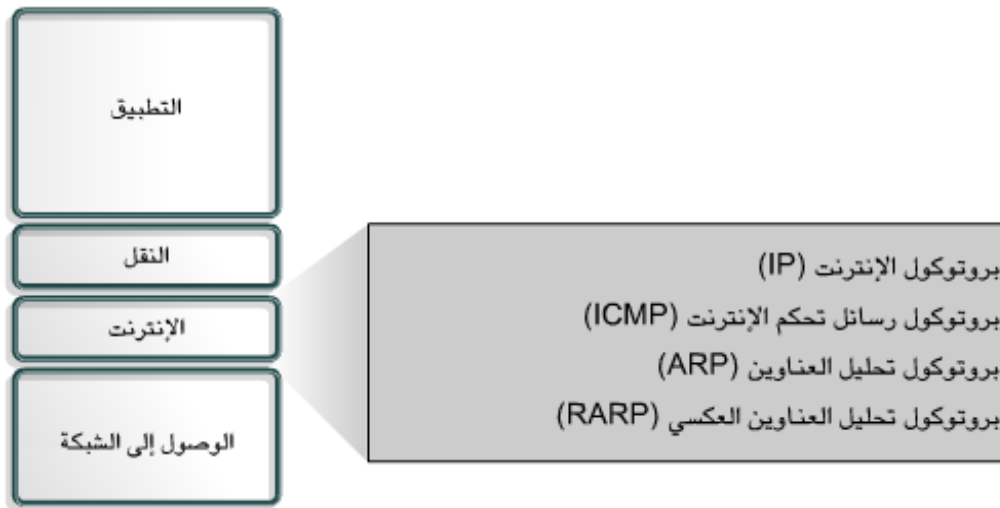


الشكل (11): طبقة النقل

طبقة الانترنت

الغرض من طبقة الإنترنت هو تحديد أفضل مسار خلال الشبكة يمكن أن تنتقل فيه الحزم packet

والبروتوكول الرئيسي الذي يعمل في هذه الطبقة هو بروتوكول الإنترنت IP



الشكل (12): طبقة الانترنت

طبقة الوصول إلى الشبكة



يعمل كل من ARP (بروتوكول تحليل العناوين) و RARP (بروتوكول تحليل العناوين العكسي) في كل من طبقة الإنترنت وطبقة الوصول إلى الشبكة

الشكل (13): طبقة الوصول للشبكة

عناوين IP (بروتوكول الانترنت)

يجب إعطاء كل كمبيوتر في شبكة (TCP/ IP) معرفاً فريداً، أو عنوان (IP).

يسمح هذا العنوان لجهاز كمبيوتر بتحديد موقع كمبيوتر آخر على الشبكة، والذي يعمل في الطبقة الثالثة .Network

ويكون لكافة أجهزة الكمبيوتر أيضاً عنوان مادي فريد، يُعرف بعنوان (MAC) التحكم في الوصول إلى الوسائط).

ويتم تعيين هذه العناوين بواسطة الشركة المصنعة لبطاقة واجهة الشبكة.

أنواع الشبكات

شبكات الند للند وشبكات مزود/زبون

Peer-to-Peer Networks & Server-Client Networks

Peer-to-Peer Networks

المقصود بشبكات الند للند أن الحواسيب في الشبكة يستطيع كل منها تأدية وظائف الزبون والمزود في نفس الوقت، وبالتالي فإن كل جهاز على الشبكة يستطيع تزويد غيره بالمعلومات وفي نفس الوقت يطلب المعلومات من غيره من الأجهزة المتصلة بالشبكة مكونة من مجموعة من الأجهزة لها LAN

إذا تعريف شبكات الند للند: هي شبكة حاسب محلية مخصص بل كل جهاز في الشبكة ممكن أن يكون Server حقوق متساوية ولا تحتوي على مزود أو زبوناً أي إن شبكات الند للند تنتمي لشبكات الإدارة الموزعة . Workgroup وهذا النوع من الشبكات يطلق عليه أيضاً اسم مجموعة عمل

Server-Client Networks

شبكات الزبون / المزود والتي تسمى أيضاً شبكة قائمة على Server-Based Network

ال Server:

- ✓ قد يكون جهاز حاسب شخصي يحتوي على مساحة تخزين كبيرة ومعالج قوي وذاكرة وفيرة
- ✓ من الممكن أن يكون جهاز مصنع خصيصاً ويكون عمله (dedicated) ليكون مخدم شبكات وتكون له مواصفات خاصة مخصص الوظيفة فقط كمخدم ولا يعمل كزبون، وعندما يصبح عدد الأجهزة في

شبكات Client / Server كبيراً يكون من الممكن إضافة مخدّم آخر، أي أن شبكات Client / Server قد تحتوي على أكثر من مخدّم واحد عند الضرورة ولكن هذه المخدمات لا تعمل أبداً كزبائن، وفي هذه الحالة تتوزع المهام على المخدمات المتوفرة مما يزيد من كفاءة الشبكة

تزود طبقة الإرسال بروتوكولين أيضاً:

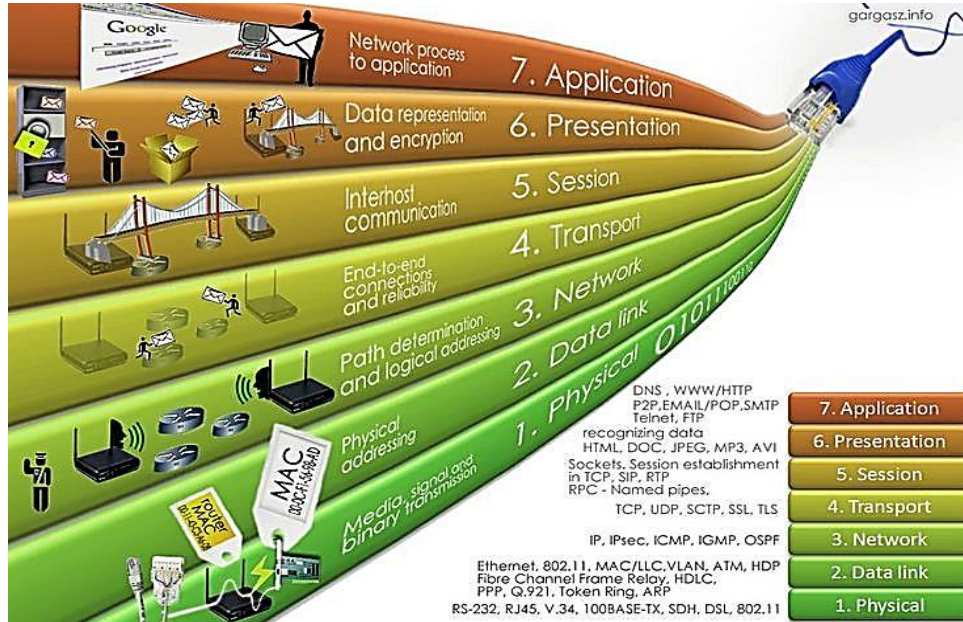
- TCP: بروتوكول موثوق بسبب تزويده أرقام تسلسل وإشعارات. يعيد TCP إرسال أي شيء لم يتم تلقيه.
- UDP: بروتوكول غير موثوق به؛ رغم أنه مسؤول عن إرسال الرسائل، لا يتم في هذه الطبقة تزويد برنامج للتحقق من تسليم الرزم. الميزات التي يقدمها UDP هي السرعة.

Sockets with TCP and UDP

أساسيات socket

مقبس الشبكة Socket ببساطة، هو نقطة نهاية افتراضية، حيث يمكن للكيانات إجراء اتصالات فيما بينها.

على سبيل المثال، تقوم إحدى العمليات في الحاسوب بتبادل البيانات مع عملية أخرى موجودة على نفس الحاسوب أو على جهاز آخر.



يملك IP لتحديد الجهاز المرسل والجهاز الهدف الذي يتصل معه، بالإضافة إلى port number يحدد التطبيق

عادة يتم تسمية العملية الأولى التي تبدأ بالاتصال بالعميل Client، أما الأخير الذي يتلقى البيانات يسمى مخدم Server. يوجد نوعين من المآخذ:

▪ TCP sockets

▪ UDP sockets

مميزات اتصال TCP:

• الموثوقية

1. يعالج الحزم المفقودة.

2. يعالج تسلسل الحزم

3. يتعامل مع الحزم المكررة.

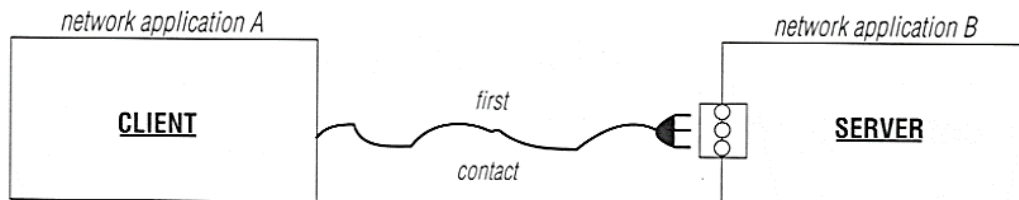
• التحكم في الازدحام.

• التحكم في التدفق.

نموذج Client/server

من الممكن أن يبدأ تطبيقان للشبكة في وقت واحد، ولكن من غير العملي طلب ذلك، لذلك يتم تصميم تطبيقات لأداء عمليات الشبكة بالتسلسل، وليس في وقت واحد.

ينفذ الخادم أولاً ثم ينتظر إلى أن يستلم العميل الأمر، ينفذ العميل العملية الثانية ويرسل حزمة إلى المخدم، تسمى هذه العملية الاتصال الأولي، بعدها يكون كل من العميل والمخدم قادرين على إرسال واستقبال البيانات.



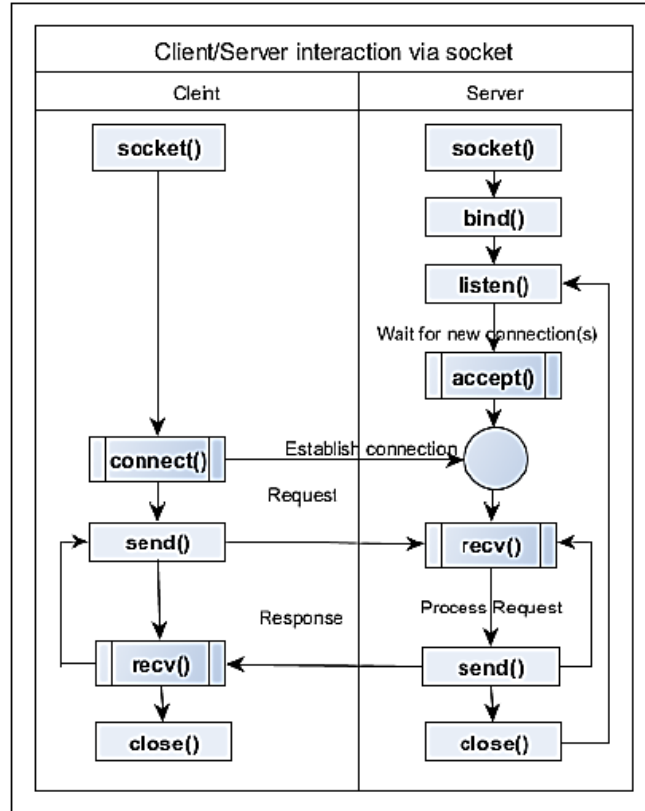
A client initiates communications to a server.

الشكل (14): Client/server model

Programming with Sockets

لدى Python طريقة سهلة للبدء بواجهة المقبس.

في الشكل التالي، يظهر التفاعل بين Client/server.



الشكل (15): Client/server interaction through socket

بعد إنشاء كائن Socket، فإن الخادم يربط هذا المقبس بعنوان IP و Port معين. هذا يشبه إلى حد كبير الهاتف اتصال برقم داخلي. في مكتب الشركة، بعد موظف جديد مع هاتف مكتبه، وعادة ما يتم تخصيص هاتف جديد له رقم التوصيلة. لذا، إذا أجرى أي شخص مكالمة هاتفية مع هذا الموظف، الاتصال يمكن إنشاؤها باستخدام رقم هاتفه وامتداده. بعد الربط الناجح، ستبدأ عملية الخادم في الاستماع إلى اتصال عميل جديد. لعميل صالح جلسة، يمكن لعملية الخادم قبول طلب عملية العميل. في هذه المرحلة، نحن يمكن القول أن الاتصال بين الخادم والعميل قد تم تأسيسه.

ثم يدخل العميل / الخادم في حلقة الطلب / الاستجابة. عملية العميل يرسل البيانات إلى عملية الخادم، ويقوم الخادم بمعالجة البيانات ويرجع ردًا إلى العميل. عند انتهاء عملية العميل، يتم الخروج من خلال الإغلاق أسفل

الاتصال. في تلك اللحظة، من المحتمل أن تعود عملية الخادم إلى حالة الاستماع. التفاعل أعلاه بين العميل والخادم هو تمثيل مبسط للغاية من الواقع الفعلي. في الممارسة العملية، تحتوي أي عملية خادم إنتاج على مؤشرات ترابط متعددة أو العمليات الفرعية للتعامل مع الاتصالات المتزامنة من آلاف العملاء عبر القنوات الافتراضية المعنية

Client and server

الإعداد الأساسي في نموذج العميل / الخادم هو جهاز واحد، الخادم الذي يقوم بتشغيل ملف الخدمة وينتظر العملاء للاتصال وتقديم طلبات للخدمة. الحوسبة النموذجي هو خادم الويب. يستمع الخادم إلى منفذ TCP للعملاء الذين يحتاجون إلى صفحات الويب الخاصة بهم. عندما يتطلب العميل، على سبيل المثال مستعرض الويب صفحة ويب يستضيفها الخادم، فإنه يتصل بالخادم ثم يقدم طلبًا لتلك الصفحة. يرد الخادم بمحتوى الصفحة ثم العميل يفصل. يعلن الخادم عن نفسه من خلال وجود اسم مضيف، والذي يقوم العملاء به يمكن استخدامه لاكتشاف عنوان IP حتى يتمكنوا من الاتصال به. في كلتا الحالتين، يكون العميل هو الذي يبدأ أي تفاعل - الخادم هو محض الاستجابة لهذا التفاعل. لذا، فإن احتياجات البرامج التي تعمل على العميل والخادم مختلفان تمامًا. عادةً ما يتم توجيه برامج العميل نحو الواجهة بين المستخدم والخدمة. يسترجعون الخدمة ويعرضونها ويسمحون للمستخدم بالتفاعل معها. تتم كتابة برامج الخادم لتظل قيد التشغيل لفترات زمنية غير محددة، أن تكون مستقرة، لتقديم الخدمة بكفاءة للعملاء الذين يطلبونها، وإمكانية التعامل مع عدد كبير من الاتصالات المتزامنة بأقل قدر ممكن التأثير على تجربة أي عميل. في هذا الفصل، سوف نلقي نظرة على هذا النموذج من خلال إعداد خادم دردشة، يمكنه التعامل مع جلسة متعددة العملاء. تناسب وحدة المقبس في Python هذه المهمة تمامًا.

An echo protocol

يجب أن يستمع خادم echo حتى يتصل العميل ويرسل سلسلة بايت، بعد ذلك يعيد echo تلك السلسلة إلى العميل. نحتاج فقط إلى بعض القواعد الأساسية لفعل هذا. هذه القواعد هي كما يلي:

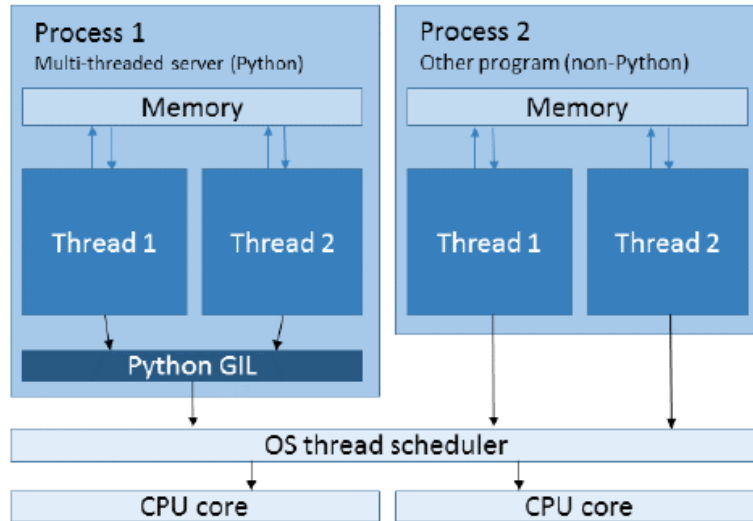
1. سيتم الاتصال عبر TCP
2. سيبدأ العميل جلسة echo عن طريق إنشاء اتصال مقبس ب الخادم.
3. سيقبل الخادم الاتصال ويستمع للعميل لإرسال ملف سلسلة بايت.
4. سيرسل العميل سلسلة بايت إلى الخادم.
5. بمجرد إرسال سلسلة البايت، سيستمع العميل للرد من الخادم
6. عندما يتلقى سلسلة البايت من العميل، سيرسل الخادم ملف سلسلة بايت إلى العميل.

7. عندما يتلقى العميل سلسلة البايت من الخادم، فإنه يغلق مقبس لإنهاء الجلسة.

هذه الخطوات واضحة بما فيه الكفاية. العنصر المفقود هنا هو كيف سيعرف الخادم والعميل متى تم إرسال رسالة كاملة. تعد غرفة الدردشة أحد التطبيقات اتصال TCP كتدفق لا نهائي من البايتات، لذلك نحن بحاجة إلى تحديد تدفق البايت الذي سيشير إلى نهاية الرسالة

Multithreading and multiprocessing

لدى Python واجهات برمجة تطبيقات تسمح لنا بكتابة كل من *Multithreading* و *multiprocessing*. المبدأ الكامن وراء تعدد العمليات والمعالجات المتعددة هو ببساطة لأخذ نسخ من الكود الخاص بنا وتشغيلها في سلاسل أو عمليات إضافية. ال يقوم نظام التشغيل تلقائيًا بجدولة الخيوط والعمليات عبر المتاحة أنوية وحدة المعالجة المركزية لتوفير تخصيص وقت معالجة عادل لجميع سلاسل العمليات والعمليات. يسمح هذا بشكل فعال للبرنامج بتشغيل عمليات متعددة في وقت واحد. في بالإضافة إلى ذلك، عندما يتم حظر خيط أو عملية، على سبيل المثال، عند انتظار الإدخال / الإخراج، فإن ملف يمكن إلغاء ترتيب الخيط أو العملية بواسطة نظام



الشكل (16): ارتباط العمليات ببعضها البعض باستخدام threading

التشغيل، ويمكن تخصيص نوى وحدة المعالجة المركزية إلى سلاسل العمليات أو العمليات الأخرى التي لديها حساب فعلي للقيام به. فيما يلي نظرة عامة على كيفية ارتباط الخيوط والعمليات ببعضها البعض:

الخيوط موجودة داخل العمليات. يمكن أن تحتوي العملية على خيوط متعددة ولكنها دائماً تحتوي على خيط واحد على الأقل، يسمى أحياناً الخيط الرئيسي. خيوط داخل نفس العملية تشترك في الذاكرة، لذا فإن نقل البيانات بين الخيوط هو مجرد حالة الرجوع إلى الأشياء المشتركة. العمليات لا تشترك في الذاكرة، لذلك واجهات أخرى، مثل الملفات أو المقابس أو المناطق المخصصة خصيصاً للذاكرة المشتركة، يجب استخدامها نقل البيانات بين العمليات.

عندما يكون للخيوط عمليات لتنفيذها، فإنها تسأل مؤشر ترابط نظام التشغيل الجدول لتخصيص بعض الوقت لهم على وحدة المعالجة المركزية، ويخصص الجدول انتظار مؤشرات الترابط إلى أنوية وحدة المعالجة المركزية بناءً على معلمات مختلفة، والتي تختلف من نظام تشغيل إلى نظام التشغيل. قد تعمل الخيوط في نفس العملية على أنوية منفصلة لوحدة المعالجة المركزية في نفس الوقت. على الرغم من عرض عمليتين في الرسم التخطيطي السابق، لا تحدث المعالجة المتعددة هنا، لأن العمليات تنتمي إلى عمليات مختلفة التطبيقات. يتم عرض العملية الثانية لتوضيح الفرق الرئيسي بين خيوط Python والخيوط في معظم البرامج الأخرى. هذا

الاختلاف هو وجود GIL

A chat protocol

سيكون الغرض الرئيسي من تحديث البروتوكول الخاص بنا هو تحديد أن العملاء يجب أن يكونوا قادرين لقبول جميع الرسائل التي يتم إرسالها إليهم متى تم إرسالها. من الناحية النظرية، سيكون أحد الحلول لهذا هو أن يقوم العميل نفسه بإعداد استماع مأخذ توصيل، بحيث يمكن للخادم الاتصال به متى كانت لديه رسالة جديدة لتسليمها. في العالم الحقيقي، نادراً ما يكون هذا الحل قابلاً للتطبيق. العملاء دائماً محميون بنوع من جدار الحماية، والذي يمنع أي اتصالات واردة جديدة من الاتصال بالعميل. لكي يقوم الخادم بإجراء اتصال بمنفذ بالنسبة للعملاء، سنحتاج إلى التأكد من تكوين أي جدران حماية متداخلة للسماح للخادم بالاتصال. هذا الشرط من شأنه أن يجعل برنامجنا كثيراً أقل جاذبية لمعظم المستخدمين نظراً لوجود حلول دردشة بالفعل لا تفعل ذلك يتطلب هذا. هناك طريقتان يمكننا من خلالهما القيام بذلك. أولاً، يمكننا تشغيل عملائنا في نطاق حالة قطع الاتصال افتراضياً، ثم

اجعلهم يتصلون بشكل دوري بال خادم. بدلا من ذلك، نستطيع اطلب من عملائنا الاتصال بال خادم ثم ترك الاتصال مفتوحًا. يستطيعون ثم استمع باستمرار إلى الاتصال والتعامل مع الرسائل الجديدة التي يرسلها الخادم في موضوع واحد، مع قبول إدخال المستخدم وإرسال الرسائل عبر نفس الاتصال في موضوع آخر. قد تتعرف على هذه السيناريوهات كخيارات السحب والدفع المتوفرة في بعض عملاء البريد الإلكتروني. يطلق عليهم سحب ودفع بسبب كيفية العمليات تظهر للعميل. يقوم العميل إما بسحب البيانات من الخادم، أو يدفع الخادم البيانات للعميل. هناك إيجابيات وسلبيات لاستخدام أي من الطريقتين، والقرار يعتمد على احتياجات التطبيق. سحب النتائج في تحميل أقل على الخادم، ولكن زمن انتقال أعلى للعميل في تلقي الرسائل. في حين أن هذا جيد بالنسبة للكثيرين التطبيقات، مثل البريد الإلكتروني، في خادم الدردشة نتوقع عادةً تحديثات فورية. بينما يمكننا إجراء استطلاعات الرأي بشكل متكرر، فإن هذا يفرض عبئًا غير ضروري على العميل، الخادم والشبكة حيث يتم إعداد الاتصالات بشكل متكرر وتمزيقها. إن Push هو الأنسب لخادم الدردشة. حيث أن الاتصال يظل مفتوحًا بشكل مستمر يقتصر مقدار حركة مرور الشبكة على إعداد الاتصال الأولي، والرسائل نفسها. أيضًا، يحصل العميل على رسائل جديدة من الخادم تقريبًا مباشرة.

لذلك، سنكتب الآن بروتوكول الدردشة الخاص بنا على النحو التالي:

1. سيتم تأسيس جلسات الاتصال.
2. سيبدأ العميل جلسة محادثة من خلال إنشاء اتصال مقبس ب الخادم.
3. سيقبل الخادم الاتصال، ويستمع إلى أي رسائل من العميل، وقبولهم.
4. سيستمع العميل على الاتصال لأي رسائل من الخادم، وقبولهم.
5. سيرسل الخادم أي رسائل من العميل إلى الآخر العملاء المتصلين.
6. سيتم تشفير الرسائل في مجموعة أحرف UTF-8 للإرسال، وسيتم إنهاؤها بالبايت الفارغ.

النتائج والمناقشة

برمجة غرفة الدردشة من جانب الخادم *server*

أولاً، نقوم بإنشاء ملف باسم `server.py`

استيراد المكتبات المطلوبة

```
# server.py
import time, socket, sys
from datetime import *
```

إنشاء مأخذ التوصيل واسترداد اسم المضيف

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

host = socket.gethostname()

ip = socket.gethostbyname(host)

port = 8080
```

- نقوم في البداية بإنشاء object من نوع `socket` باستخدام

```
Socket.socket (socket.AF_INET, socket.SOCK_STREAM)
```

حيث `AF_INET` تعني عائلة عناوين Address Family تستخدم لتعيين نوع العناوين التي يمكن للمقبس الاتصال بها، وهنا تعني أنه يمكن الاتصال بعناوين بروتوكول الانترنت الإصدار 4 (v4)

`SOCK_STREAM` تعني أن الاتصال TCP

بمجرد إنشاء المقبس، نقوم باسترداد اسم المضيف / اسم الجهاز المحلي باستخدام `gethostname()` ، والذي يعد مرة أخرى وظيفة لمكتبة المقابس.

ربط المضيف مع المنفذ

```
s.bind((host, port))
```

طباعة اسم المضيف مع عنوان IP له

```
print(host, "(" , ip, ")\n")
```

فيظهر بالشكل:

```
DESKTOP-AST1106 { 169.254.227.48 }
```

نطلب من المضيف إدخال اسمه ليتم حفظه في server:

```
name = input(str("Enter your name: "))
```

الاستماع إلى الاتصالات

```
s.listen(5)
```

```
print("\nWaiting for incoming connections...\n")
```

تقوم التعليمة `s.listen(5)` بالاستماع، مع السماح لـ 5 طلبات اتصال معلقة قبل أن يتم قبولها

قبول الاتصالات الواردة

```
conn, addr = s.accept()
```

```
print("Received connection from ", addr[0], "(" ,  
addr[1], ")\n")
```

تخزين بيانات الاتصال الواردة

```
s_name = conn.recv(1024)
```

```
s_name = s_name.decode()
```

```
print(s_name, "has connected to the chat app\nEnter  
[e] to exit the chat\n")
```

```
conn.send(name.encode())
```

يتم تخزين تفاصيل الاتصال الوارد في متغير s_name ، يمكن أن يكون الحد الأقصى لعدد الـ bytes لاسم العميل 1024 bytes. يتم فك تشفيره على الخادم، ثم يرسل رسالة تفيد بأنه تم توصيله. ثم يرسل الخادم اسم المضيف.

قبل بدء المحادثة، نقوم بإنشاء غرض لعرض تاريخ المحادثة

```
date_nowh = datetime.now().strftime(' (%Y-%m-%d) ')
```

```
print(date_nowh)
```

تسليم الرسائل

```
while True:
```

```
    date_nowt = datetime.now().strftime(' (%H:%M:%S) ')
```

```
    print(date_nowt, end='    ')
```

```
    message = input(str("Me : "))
```

```
    if message == "[e]":
```

```
        message = "Left chat app!"
```

```
        conn.send(message.encode())
```

```
        print("\n")
```

```
        break
```

```
    conn.send(message.encode())
```

```
    message = conn.recv(1024)
```

```
    message = message.decode()
```

```
    print(date_nowt, '    ', s_name, ":", message)
```

في البداية تم إنشاء غرض لعرض وقت إرسال كل رسالة، فتظهر في بداية كل رسالة رسالة.

عندما يقوم المستخدم بإدخال الرسالة، يتم ترميزها باستخدام `encode()` ثم إرسالها عبر المقبس. يتم إرسال الرسالة باستخدام `send()` التي يتم استدعاؤها عن طريق كائن الاتصال `conn` الذي تم إنشاؤه أثناء استدعاء وظيفة `Accept()` سابقاً.

يتم استلام الرسالة الواردة باستخدام `recv()`، حيث يقوم الكائن `conn` أن تتلقى ما يصل إلى 1024 بايت من المعلومات. يتم فك تشفير الرسالة على جانب الخادم باستخدام `decode()`.

ثم طباعة الرسالة بالإضافة إلى وقت إرسال كل رسالة واسم المرسل. تم إضافة حالة الخروج من الدردشة، ففي حال تم إرسال **[e]** سيتم الخروج من الدردشة.

كود غرفة الدردشة من جانب server

```
# server.py

import time, socket, sys

from datetime import *

print("\nWelcome to Chat app\n")

print("Initialising....\n")

#time.sleep(1)

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

host = socket.gethostname()

ip = socket.gethostbyname(host)

port = 8080

s.bind((host, port))

print(host, "(", ip, ") \n")
```

```

name = input(str("Enter your name: "))

s.listen(5)

print("\nWaiting for incoming connections...\n")

conn, addr = s.accept()

print("Received connection from ", addr[0], "(", addr[1],
")\n")

s_name = conn.recv(1024)

s_name = s_name.decode()

print(s_name, "has connected to the chat app\nEnter [e] to
exit the chat\n")

conn.send(name.encode())

date_nowh = datetime.now().strftime('%Y-%m-%d')

print(date_nowh)

while True:

    date_nowt = datetime.now().strftime('%H:%M:%S')

    print(date_nowt, end=' ')

    message = input(str("Me : "))

    if message == "[e]":

        message = "Left chat app!"

        conn.send(message.encode())

        print("\n")

        break

```

```
conn.send(message.encode())
```

```
message = conn.recv(1024)
```

```
message = message.decode()
```

```
print(date_nowt, ' ', s_name, ":", message)
```

برمجة غرفة الدردشة من جانب العميل Client

استيراد المكتبات

نقوم باستيراد نفس المكتبات المستخدمة من جانب الخادم

```
# client.py
```

```
import time, socket, sys
```

```
from datetime import *
```

إنشاء المقبس وقبول اسم مضيف إدخال المستخدم

```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
shost = socket.gethostname()
```

```
ip = socket.gethostbyname(shost)
```

```
print(shost, "(", ip, ")\n")
```

```
host = input(str("Enter server address: "))
```

```
name = input(str("\nEnter your name: "))
```

```
port = 8080
```

- تم إنشاء المقبس في الخادم باستخدام الطريقة (`socket()`), حيث تم تحديد الاتصال على أنه TCP كما في حالة ال `server`

- تم استرداد اسم مضيف الخادم من جانب العميل وتخزينه كـ `shost`

- تم تخزين عنوان IP في `ip`

- تم تخزين منفذ الخادم كـ 8080 في المتغير `port`

- يتم إدخال IP الخادم، يجب ملاحظة أنه من المهم إدخال عنوان IP الخادم نفسه وإلا سيفشل الاتصال.

- ثم يدخل الـ client اسمه

الاتصال بالخادم

```
print("\nTrying to connect to ", host, "(", port,
")\n")
```

```
s.connect((host, port))
```

```
print("Connected...\n")
```

استلام الرسائل من الخادم

```
s.send(name.encode())
```

```
s_name = s.recv(1024)
```

```
s_name = s_name.decode()
```

```
print(s_name, "has joined the chat app\nEnter [e] to
exit the chat\n")
```

```
date_nowh = datetime.now().strftime('%Y-%m-%d')
```

```
print(date_nowh)
```

```
while True:
```

```
    date_nowt = datetime.now().strftime('%H:%M:%S')
```

```
    message = s.recv(1024)
```

```
    message = message.decode()
```

```
    print(date_nowt, ' ', s_name, ":", message)
```

```
    print(date_nowt, end=' ')
```

```
    message = input(str("Me : "))
```

```
    if message == "[e]":
```

```
        message = "Left chat app!"
```

```
s.send(message.encode())
```

```
print("\n")
```

```
break
```

```
s.send(message.encode())
```

- في البداية أرسلنا اسم المضيف إلى الخادم لتخزينه فيه، ثم تم تخزين الاسم في متغير s_name، وطباعته من أجل إظهاره في واجهة الدردشة.

- نقوم بإنشاء عرض لعرض تاريخ المحادثة

```
date_nowh = datetime.now().strftime(' (%Y-%m-%d) ')
```

```
print(date_nowh)
```

- تم إنشاء عرض لعرض وقت إرسال كل رسالة، فتظهر في بداية كل رسالة رسالة.

- يتم استقبال الرسائل من الخادم باستخدام (1024) recv تخزين في متغير message

- يتم طباعة الرسالة باسم المضيف والرسالة المستلمة بالإضافة إلى وقت استلام الرسالة.

- يتم إرسال الرسائل من المضيف باستخدام تعليمة الإدخال والتي تخزن ضمن نفس المتغير

```
s.send(message.encode())
```

 باستخدام server ثم ترسل إلى الـ message

- تم إضافة حالة الخروج من الدردشة، ففي حال تم إرسال [e] سيتم الخروج من الدردشة.

يمكن للعميل إدخال أي رسالة ليتم تشفيرها وإرسالها إلى الخادم باستخدام socket

كود غرفة الدردشة من جانب العميل

```
# client.py

import time, socket, sys

from datetime import *

print("\nWelcome to Chat app\n")

print("Initialising....\n")

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

shost = socket.gethostname()

ip = socket.gethostbyname(shost)

print(shost, "(", ip, ")\n")

host = input(str("Enter server address: "))

name = input(str("\nEnter your name: "))

port = 8080

print("\nTrying to connect to ", host, "(", port, ")\n")

s.connect((host, port))

print("Connected...\n")


s.send(name.encode())

s_name = s.recv(1024)

s_name = s_name.decode()

print(s_name, "has joined the chat app\nEnter [e] to exit the chat\n")
```

```

date_nowh = datetime.now().strftime(' (%Y-%m-%d) ')

print(date_nowh)

while True:

    date_nowt = datetime.now().strftime(' (%H:%M:%S) ')

    message = s.recv(1024)

    message = message.decode()

    print(date_nowt, ' ', s_name, ":", message)

    print(date_nowt, end=' ')

    message = input(str("Me : "))

    if message == "[e]":

        message = "Left chat app!"

        s.send(message.encode())

        print("\n")

        break

    s.send(message.encode())

```

النتائج

بداية الاتصال:

```
Welcome to Chat app

Initialising....

DESKTOP-AST1106 ( 169.254.227.48 )

Enter your name: Deemah

Waiting for incoming connections...
```

الشكل 1: بداية الاتصال، تفعيل server

```
Welcome to Chat app

Initialising....

DESKTOP-AST1106 ( 169.254.227.48 )

Enter server address: 169.254.227.48

Enter your name: Honen
```

الشكل 1-2: تفعيل client

غرفة الدردشة بعد بناء اتصال ناجح بين Client و server

من جهة الخادم

```
(2022-05-29)
(22:07:58) Me : Hi
(22:07:58) Honen : Hi Deemah
(22:09:19) Me : How are you?
(22:09:19) Honen : I'm Fine... and you?
(22:09:51) Me : very good..
(22:09:51) Honen : sorry..I will talk to you later
(22:10:59) Me : don;
(22:10:59) Honen : bye
(22:11:34) Me : bye
(22:11:34) Honen : Left chat app!
```

من جهة ال Client

```
(2022-05-29)
(22:07:58) Deemah : Hi
(22:07:58) Me : Hi Deemah
(22:09:19) Deemah : How are you?
(22:09:19) Me : I'm Fine... and you?
(22:09:51) Deemah : very good..
(22:09:51) Me : sorry..I will talk to you later
(22:10:59) Deemah : don
(22:10:59) Me : bye
(22:11:34) Deemah : bye
(22:11:34) Me : [e]
```

تصميم واجهة GUI لتطبيق Chat App

واجهة GUI هي اختصار لـ Graphical User Interface، وهي واجهة للمستخدم تؤمن للمستخدم تفاعل مع الحاسب باستخدام أغراض وصور رسومية، تتكون من نوافذ منبثقة بالإضافة لنصوص توجه المستخدم لاستخدام أحداث مخصصة (مثل النقر على mouse لإضافة نصوص ليقوم الحاسب بعدها بالقيام بما يطلبه المستخدم

إن تصميم واجهة GUI للمستخدم تتم عن طريق كتابة تعليمات برمجية في كود المضيف Client.

هناك العديد من مجموعات أدوات واجهة GUI التي يمكن استخدامها مع لغة python، الأداة الشائعة التي تستخدم لتصميم واجهة GUI هي (tkinter)

Tkinter هي عبارة عن مكتبة في python تؤمن تنفيذ واجهات رسومية للبرنامج، من أجل التفاعل مع المستخدم.

فكرة التطبيق:

تطبيق دردشة جماعية متعددة باستخدام tkinter و python socket

الفكرة العامة له هي إنشاء واجهة رسومية يتفاعل معها المستخدم لخدمة الدردشة في الزمن الحقيقي

وظائفه:

- إنشاء غرفة / الانضمام إلى الغرفة باستخدام معرف الغرفة.

- إرسال الرسائل داخل الغرفة.

برمجة غرفة الدردشة من جهة الخادم *Server*

نقوم في البداية باستيراد المكتبات المطلوبة

```
from socket import *
```

```
from datetime import *
```

```
import threading
```

```
import time
```

threading module تسمح لنا server بتخديم أكثر من زبون في آن واحد

ثم نقوم بإنشاء مأخذ توصيل مع إعطاء host و IP ثم نقوم بربط المضيف مع المنفذ ، ثم يبدأ بانتظار المضيف

```
Host='127.0.0.1'
```

```
port=9090
```

```
server=socket(AF_INET,SOCK_STREAM)
```

```
server.bind((Host,port))
```

```
server.listen()
```

نقوم بتعريف قائمة لإدخال المضيفين

```
clients=[]
```

```
nicknames=[]
```

```
#broadcast
```

```
def broadcast(message):
```

```
for client in clients:
```

```
client.send(message)
```

نقوم بتعريف تابع broadcast، هذا التابع يعمل كالآتي: عند استقبال الرسالة من أحد ال Client يقوم بإعادة

توجيهها إلى جميع ال Clients الأخرى.

عمليات اتصال المضيفين والدردشات بين المضيفين، حتى المضيفين المغادرين يتم إظهارها في ال server

ولكن لا تظهر في الواجهات


```

def handle (client):
    while True:
        try:
            message=client.recv(1024).decode('utf-8')
            print(f"{nicknames[clients.index(client)]}
                  says {message}")
            broadcast(message.encode())
        except:

            index=clients.index(client)
            clients.remove(client)
            client.close()
            nickname=nicknames[index]
            print(f"{nickname} left the
                  conversation!!!!")
            msg=f"{nickname} left the conversation
                room!!!!\n"
            broadcast(msg.encode())
            nicknames.remove(nickname)
            break

```

التابع Remove يحذف المستخدم من لائحة الغرفة ويحذف المعرف الخاص به لمنح هذا المعرف لمستخدم آخر.

يتم تشغيل المخدم على العنوان المحلي ويعطيه البورت ويبدأ باستقبال طلبات الدخول الى الغرفة.

وأخيراً نقوم ببناء server كما في الحالة السابقة، حيث تم تعريف تابع للاستقبال من أجل إرسال واستقبال الرسائل من Clients

```

def receive():
    while True:
        client,address=server.accept()
        print(f"connected with{str(address)}")
        clients.append(client)

        client.send('nick'.encode('utf-8'))
        nickname=client.recv(1024).decode('utf-8')
        nicknames.append(nickname)

```

```

        print(f"name of client is {nickname}")
date_nowt2 = datetime.now().strftime('%H:%M:%S')
        broadcast(f" {date_nowt2} {nickname} join to
                    conversation room @@@ the
subscribes{nicknames}\n".encode('utf-8'))
        client.send(f'new subscribe is
{nickname}'.encode('utf-8'))

thread=threading.Thread(target=handle,args=(client,))
        thread.start()
        print('server is running---')
        receive()

```

server غرفة الدردشة من الخطة

```

        from socket import *
        from datetime import *
        import threading
        import time
        Host='127.0.0.1'
        port=8080
server=socket(AF_INET,SOCK_STREAM)
        server.bind((Host,port))
        server.listen()
        clients=[]
        nicknames=[]
        #broadcast
        def broadcast(message):
        for client in clients:
            client.send(message)
        def handle (client):
            while True:
                try:
                    message=client.recv(1024).decode('utf-8')
                    print(f"{nicknames[clients.index(client)]}
                        says {message}")
                    broadcast(message.encode())
                except:

```

```

        index=clients.index(client)
        clients.remove(client)
        client.close()
        nickname=nicknames[index]
        print(f"{nickname} left the
              conversation!!!!")
    msg=f"{nickname} left the conversation
         room!!!!\n"
        broadcast(msg.encode())
        nicknames.remove(nickname)
        break

    def receive():
        while True:
            client,address=server.accept()
            print(f"connected with{str(address)}")
            clients.append(client)

            client.send('nick'.encode('utf-8'))
            nickname=client.recv(1024).decode('utf-8')
            nicknames.append(nickname)

            print(f"name of client is {nickname}")
            date_nowt2 = datetime.now().strftime('%H:%M:%S')
            broadcast(f" {date_nowt2} {nickname} join to
                     conversation room @@@ the
                     subscribes{nicknames}\n".encode('utf-8'))
            client.send(f'new subscribe is
                       {nickname}'.encode('utf-8'))

thread=threading.Thread(target=handle,args=(client,))
                        thread.start()
    print('server is running---')
        receive()

```

برمجة غرفة الدردشة من جانب العميل *Client*

نقوم في البداية باستيراد المكتبات التي نحتاجها

```
from socket import *
from datetime import *
import threading
import tkinter
import tkinter.scrolledtext
from tkinter import simpledialog
```

نلاحظ استيراد مكتبة `threading` التي تسمع للمستخدمين بالاتصال على التوازي

ومكتبة الواجهات الرسومية `tkinter` التي سنستخدمها لتصميم الواجهة التفاعلية

من أجل تصميم الواجهة، نعرّف تابع ليقوم بالاتصال بال `server` عبر `ip host` و `port`

```
class Client:

    def __init__(self, Host, port):
        self.sock=socket(AF_INET, SOCK_STREAM)
        self.sock.connect((Host, port))
        msg=tkinter.Tk()
        msg.withdraw()

        self.nickname=simpledialog.askstring('Nickname', 'please
            choose name', parent=msg)
        self.gui_done=False
        self.running=True

        gui_thread =threading.Thread(target=self.gui_loop)

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

البداية نقوم ببناء المضيف:

✓ نشق غرض من tkinter كالتالي msg=tkinter.Tk()

✓ لتظهر نافذة للمستخدم ليقوم بإدخال معلومات عنه

```
self.nickname=simpledialog.askstring('Nickname', 'please  
choose name', parent=msg)
```

ثم يتم دخول client إلى تصميم واجهة الدردشة

```
def gui_loop(self):  
    self.win=tkinter.Tk()  
    self.win.title('CHATT APPLICATION')  
    self.win.configure(bg='#8AFAC5')  
  
    self.chat_label=tkinter.Label(self.win, text="CHat  
App", bg="lightgray")  
    self.chat_label.config(font='Algerian 20 bold')  
    self.chat_label.pack(padx=20, pady=5)  
  
    self.text_area=tkinter.scrolledtext.ScrolledText(self.win)  
    self.text_area.pack(padx=20, pady=5)  
    self.text_area.config(state='disabled')  
  
    self.msg_label=tkinter.Label(self.win, text="Message", bg='#  
8AFAC5')  
    self.msg_label.config(font='Vivaldi 20 bold')  
    self.msg_label.pack(padx=20, pady=5)  
  
    self.input_area=tkinter.Text(self.win, height=3)  
    self.input_area.pack(padx=20, pady=5)  
  
    self.send_button=tkinter.Button(self.win, text='send', comma  
nd=self.write)  
    self.send_button.config(font='Vivaldi 15 bold')  
    self.send_button.pack(padx=20, pady=5)  
  
    self.gui_done=True
```

```
self.win.protocol('WM_DELETE_WINDOW', self.stop)
self.win.mainloop()
```

نحدد عنوان النافذة، مع إعطاءها لون ونوع خط محدد، ثم نعرف النوافذ ضمن الواجهة وهي نافذة لعرض الدردشة ونافذة إرسال الرسالة، ونضيف زر الإرسال، ونحدد لكل منها أبعاد وألوان وخط محدد.

تم تصميم واجهة لكن لا يمكننا استخدامها للكتابة، لذلك نعرف تابع يرسل ويستقبل الرسائل ثم تعرض في الواجهة المخصصة

```
def write(self):
    date_nowt = datetime.now().strftime('%H:%M:%S')
    message=f" {date_nowt} {self.nickname}
    :{self.input_area.get('1.0','end')}\n"
    self.sock.send(message.encode('utf-8'))
    self.input_area.delete('1.0','end')
```

نعرف تابع لإيقاف استقبال الرسائل وإغلاق ال Socket إذا أراد المستخدم المغادرة

```
def stop(self):
    self.runnig=False
    self.win.destroy()
    self.sock.close()
    exit(0)
```

وأخيراً، كما في الكود السابق، نقوم بكتابة التابع المسؤول عن الاتصال مع ال server وتبادل الرسائل معه

```
def receive(self):
```

```

        while self.running:
            try:
                message=self.sock.recv(1024).decode('utf-8')
                if message == 'nick':

self.sock.send(self.nickname.encode('utf-8'))

            else:
                if self.gui_done:

self.text_area.config(state='normal')

self.text_area.insert('end',message)
self.text_area.yview('end')

self.text_area.config(state='disabled')

        except ConnectionAbortedError as err:
            print(err)
            break
        except:
            print('error')
            self.sock.close()
            break
    client=Client(Host,port)

```

وبذلك تم تصميم واجهة ل client

ملحوظة: غرفة الدردشة من جهة العميل client

```

from socket import *
from datetime import *
import threading
import tkinter
import tkinter.scrolledtext
from tkinter import simpledialog
Host='127.0.0.1'
port=8080

```

```

class Client:

    def __init__(self, Host, port):
        self.sock=socket(AF_INET, SOCK_STREAM)
        self.sock.connect((Host, port))
        msg=tkinter.Tk()
        msg.withdraw()

self.nickname=simpledialog.askstring('Nickname', 'please
choose name', parent=msg)
self.gui_done=False
self.running=True

gui_thread =threading.Thread(target=self.gui_loop)

receive_thread=threading.Thread(target=self.receive)
    gui_thread.start()
    receive_thread.start()

    def gui_loop(self):
        self.win=tkinter.Tk()
        self.win.title('CHATT APPLICATION')
        self.win.configure(bg='#8AFAC5')

self.chat_label=tkinter.Label(self.win, text="CHat
App", bg="lightgray")
self.chat_label.config(font='Algerian 20
bold')
self.chat_label.pack(padx=20, pady=5)

self.text_area=tkinter.scrolledtext.ScrolledText(self.win)
    self.text_area.pack(padx=20, pady=5)
    self.text_area.config(state='disabled')

self.msg_label=tkinter.Label(self.win, text="Message", bg='#
8AFAC5')
    self.msg_label.config(font='Vivaldi 20 bold')
    self.msg_label.pack(padx=20, pady=5)

self.input_area=tkinter.Text(self.win, height=3)

```



```

        self.input_area.pack(padx=20, pady=5)

self.send_button=tkinter.Button(self.win, text='send', command=
                                nd=self.write)
self.send_button.config(font='Vivaldi 15
                        bold')
self.send_button.pack(padx=20, pady=5)

        self.gui_done=True

self.win.protocol('WM_DELETE_WINDOW', self.stop)
        self.win.mainloop()

        def write(self):
            date_nowt =
datetime.now().strftime('%H:%M:%S')
message=f" {date_nowt}      {self.nickname}
        :{self.input_area.get('1.0', 'end')}\n"
self.sock.send(message.encode('utf-8'))
        self.input_area.delete('1.0', 'end')

        def stop(self):
            self.runnig=False
            self.win.destroy()
            self.sock.close()
            exit(0)

        def receive(self):

            while self.running:
                try:

message=self.sock.recv(1024).decode('utf-8')
                    if message == 'nick':

self.sock.send(self.nickname.encode('utf-8'))

                        else:
                            if self.gui_done:

                                self.text_area.config(state='normal')

```

```
        self.text_area.insert('end',message)
        self.text_area.yview('end')

self.text_area.config(state='disabled')

except ConnectionAbortedError as err:
    print(err)
    break
except:
    print('error')
    self.sock.close()
    break
client=Client(Host,port)
```

النتائج

في حالة محادثة بين مستخدمين

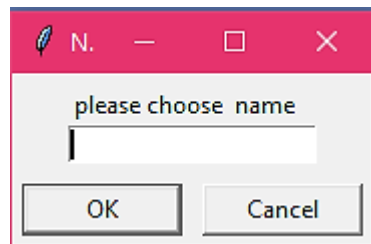
عند بداية تشغيل ال server وقبل اتصال أي عميل

```
C:\Users\ASUS\PycharmProjects\untitled5\venv\Scr
server is running---
```

عند اتصال أول عميل في ال Server

```
C:\Users\ASUS\PycharmProjects\untitled5
server is running---
connected with('127.0.0.1', 53840)
|
```

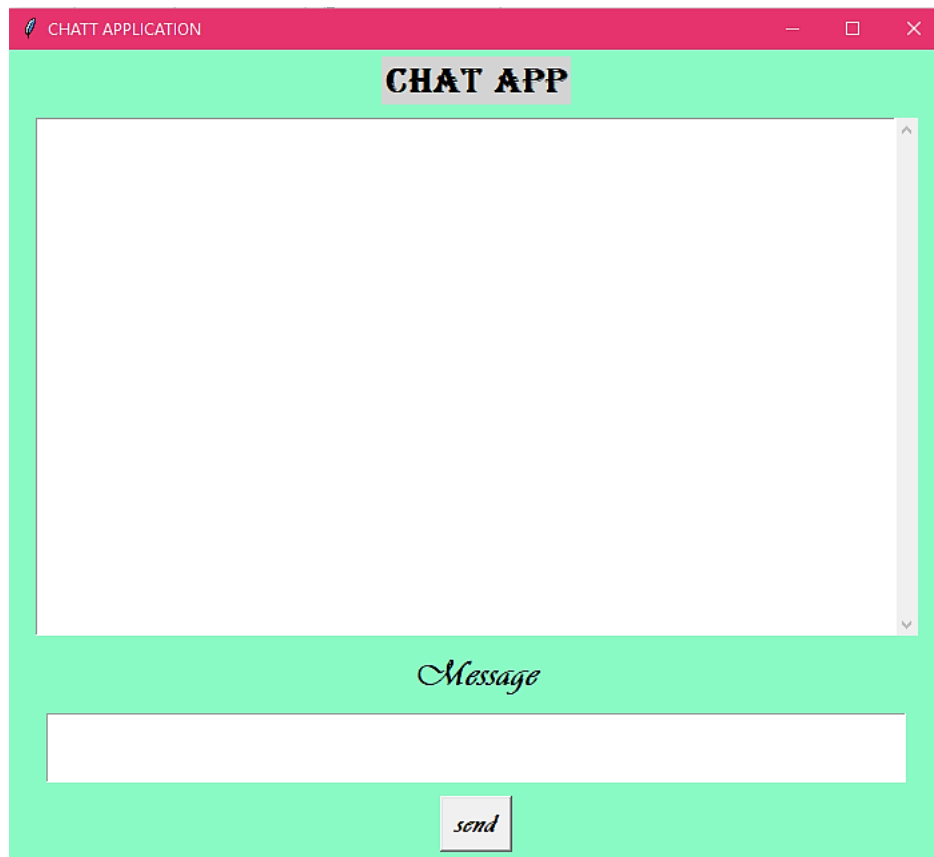
في جهة client تظهر واجهة تسجيل الدخول



فيتم تسجيل اسم العميل في ال server ويتم طباعة ذلك في جهة ال server

```
C:\Users\ASUS\PycharmProjects\untitled5\venv
server is running---
connected with('127.0.0.1', 53840)
name of client is Deemah
```

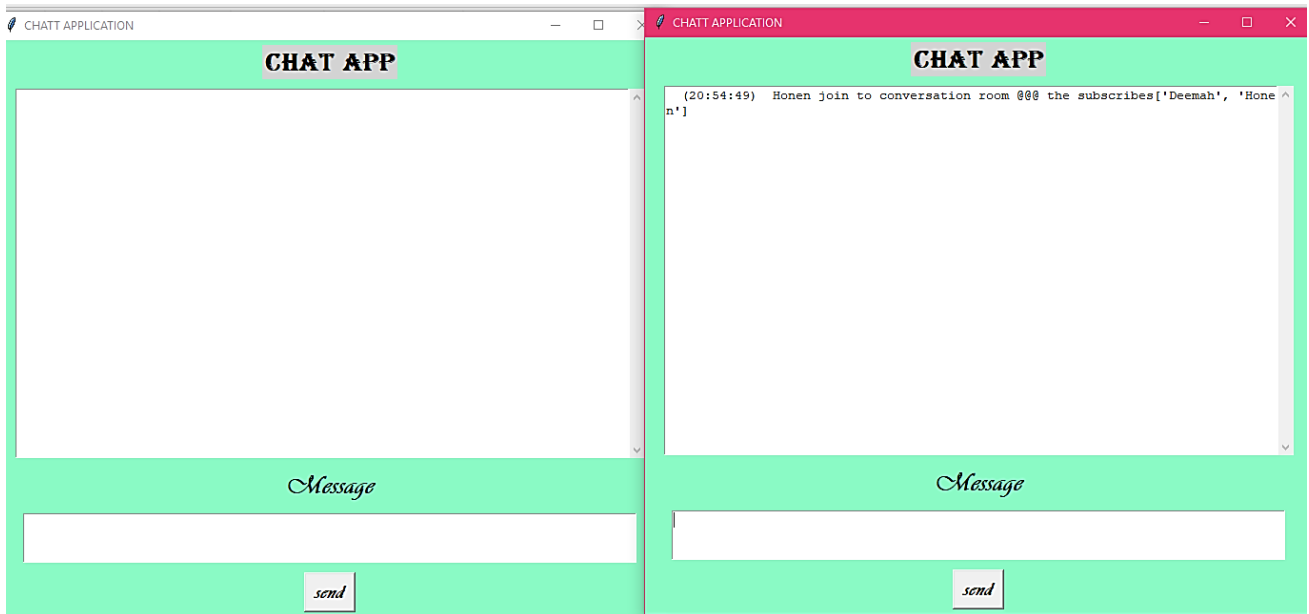
ثم تظهر الواجهة الخاصة بال دردشة للعميل



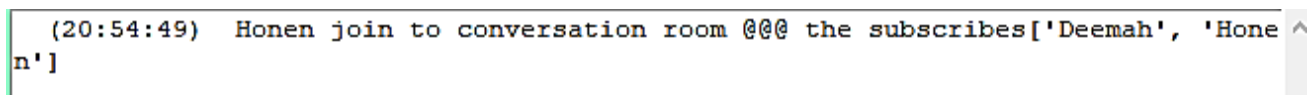
عند اتصال عميل آخر في ال server وبعد تسجيل دخوله

```
C:\Users\ASUS\PycharmProjects\untitled5\venv\  
server is running---  
connected with('127.0.0.1', 53840)  
name of client is Deemah  
connected with('127.0.0.1', 53841)  
name of client is Honen  
,
```

يتم التواصل بين clients عبر الواجهات



نلاحظ أنه تظهر رسالة عند العميل الأول أن العميل الثاني قد دخل في غرفة الدردشة، ويعرض أسماء الموجودين في غرفة الدردشة



المحادثة بين العملاء

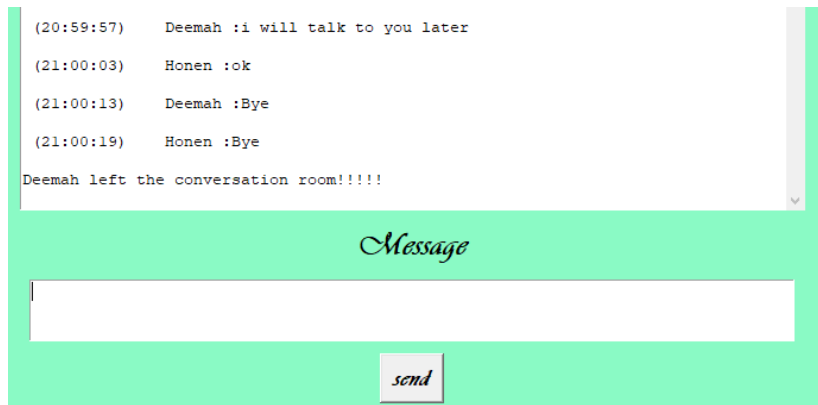


يتم طباعة محادثة العملاء في ال server بالشكل

```
name of client is Deemah
connected with('127.0.0.1', 53841)
name of client is Honen
Deemah says (20:56:35) Deemah :Hello
```

```
Honen says (20:56:59) Honen :Hi.. How are U
```

خروج أحد العملاء من المحادثة



The screenshot shows a chat application window with a light green background. At the top, there is a scrollable list of messages in a monospaced font. The messages are: (20:59:57) Deemah :i will talk to you later, (21:00:03) Honen :ok, (21:00:13) Deemah :Bye, (21:00:19) Honen :Bye, and Deemah left the conversation room!!!!. Below the list is a text input field with the word "Message" centered above it. At the bottom right of the input field is a button labeled "send".

```
(20:59:57) Deemah :i will talk to you later
(21:00:03) Honen :ok
(21:00:13) Deemah :Bye
(21:00:19) Honen :Bye
Deemah left the conversation room!!!!
```

Message

send

في جهة ال server

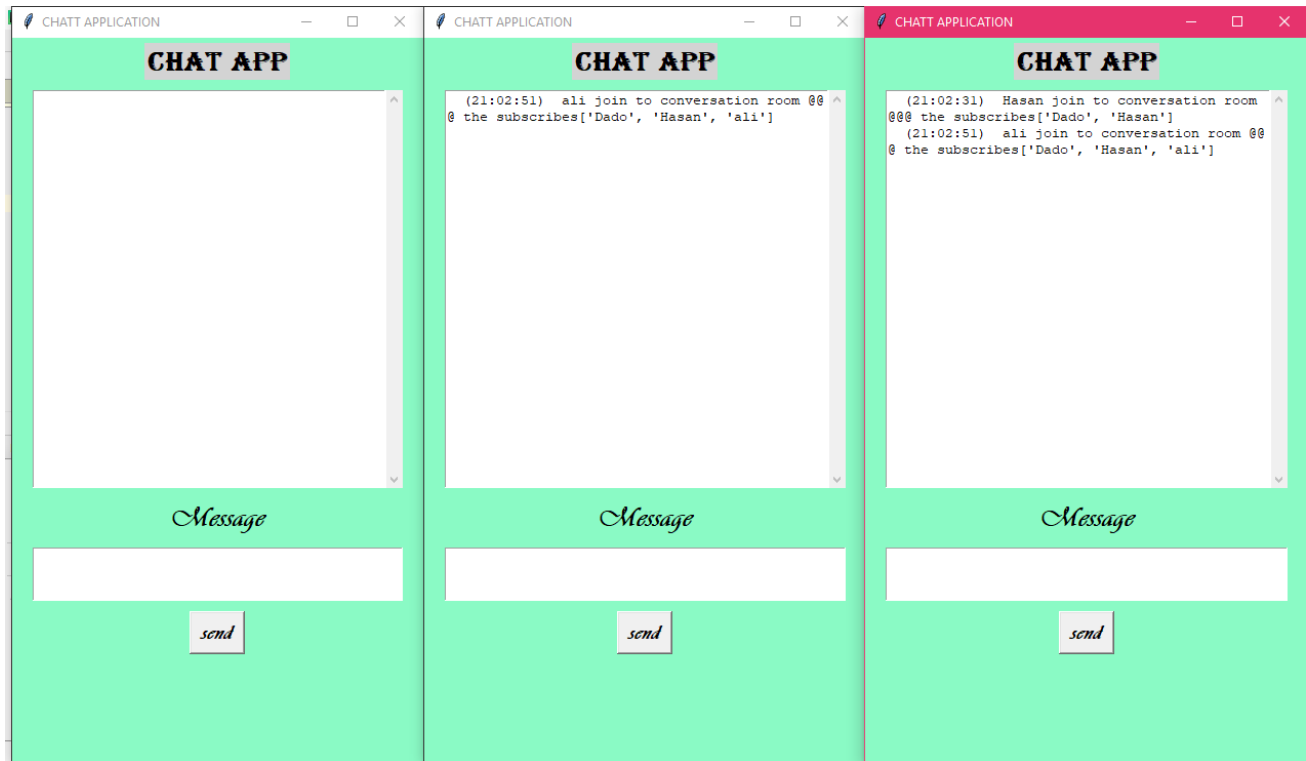


The screenshot shows a terminal window with a white background and a vertical scrollbar on the left. It displays the server-side log of the chat conversation in a monospaced font. The log entries are: Deemah says (20:59:57) Deemah :i will talk to you later, Honen says (21:00:03) Honen :ok, Deemah says (21:00:13) Deemah :Bye, Honen says (21:00:19) Honen :Bye, and Deemah left the conversation!!!!. A cursor is visible at the end of the last line.

```
Deemah says (20:59:57) Deemah :i will talk to you later
Honen says (21:00:03) Honen :ok
Deemah says (21:00:13) Deemah :Bye
Honen says (21:00:19) Honen :Bye
Deemah left the conversation!!!!
|
```

في حالة محادثة بين ثلاث مستخدمين

الواجهات التي تظهر عند العملاء

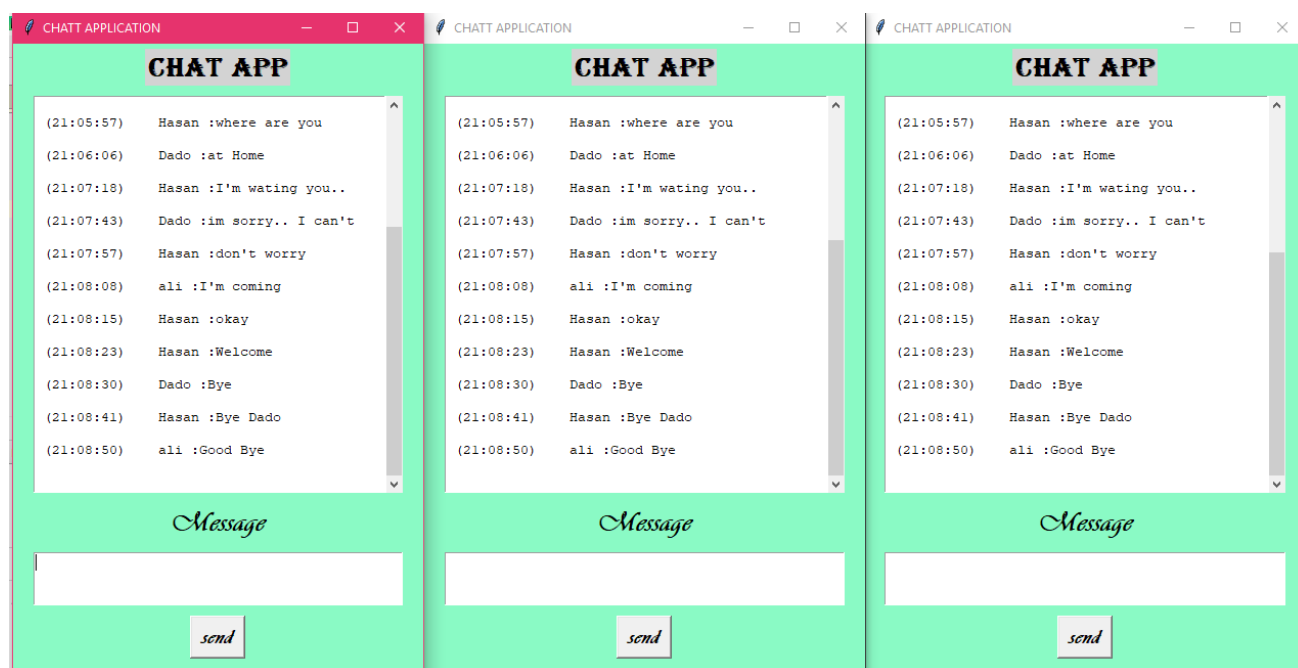


عند اتصال العميل الثاني، يظهر ذلك عند العميل الأول، وعند اتصال العميل الثالث، يظهر ذلك عند كل من العميلين الأول والثاني

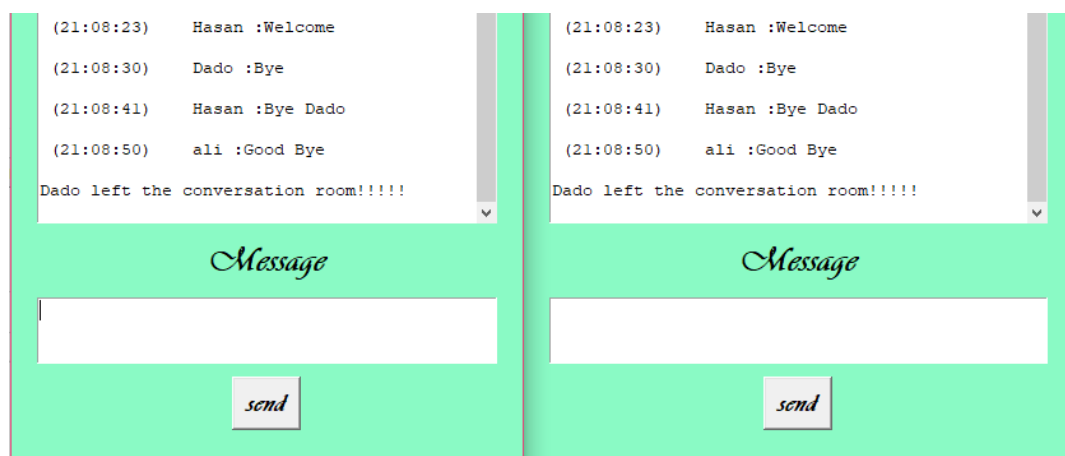
في جهة ال server

```
connected with('127.0.0.1', 53842)
name of client is Dado
connected with('127.0.0.1', 53843)
name of client is Hasan
connected with('127.0.0.1', 53844)
name of client is ali
```

واجهة المحادثة بين العملاء



خروج أول عميل، تظهر عند العميلين الآخرين رسالة تخبرهم بذلك



وكذلك عند خروج العميل الآخر تظهر رسالة عند العميل الثالث

(21:08:19) hasan :Okay

(21:08:23) Hasan :Welcome

(21:08:30) Dado :Bye

(21:08:41) Hasan :Bye Dado

(21:08:50) ali :Good Bye

Dado left the conversation room!!!!

Hasan left the conversation room!!!!

Message

send

في جهة server عند خروج كل العملاء

Hasan says (21:08:23) Hasan :Wel

Dado says (21:08:30) Dado :Bye

Hasan says (21:08:41) Hasan :Bye

ali says (21:08:50) ali :Good By

Dado left the conversation!!!!

Hasan left the conversation!!!!

ali left the conversation!!!!

|

