TCP Socket Clients/Server

Chat app with GUI using tkienter library

نضير سليم حسن محمد حسين القريمه

التاريخ

28/6/2021

□الملخص□

إن الفكرة الأساسية للمشروع هي الربط بين مكتبة tkinter في ال python (المخصصة لبناء الواجهات الرسومية GUI) وهيكلية socket client/sever وذلك لخلق بنية أكثر تفاعلية بين الserver والclinets،حيث سنقوم ببناء server يقوم بتخديم مجموعة من الclients معاً وفي آنٍ واحد وذلك اعتماداً على تعدد المسارات multithreading،ويتعامل هذا الserver مع كل client على حدى من خلال واجهة رسومية GUI تظهر عند تلقي الserver لرسالة من الclient (أي من أجل كل client سوف تظهر واجهة رسومية GUI يتعامل معها الserver من خلالها)،في الجهة المقابلة فإن كل client بمجرد اتصاله بالserver سوف تظهر له واجهة رسومية للإرسال يرسل من خلالها رسالته، ويدخل بعدها في وضع الانتظار إلى حين وصول رسالة من الserver فتظهر واجهه أخرى من خلالها يستطيع قراءة رد ال server.

الكلمات المفتاحية:

الواجهات الرسومية GUI،Tkinter،Client،Server،Threading،Socket،GIL

مقدمة:

إن ال socket هي أحدى طرق الاتصال القائم بين عميلين ضمن شبكة معينة،وهي الطريقة التي سوف نستفيد منها مشروعنا للربط بين clients وserver ،ويمرر لكل socket أو مقبس ثلاث بارامترات(عنوان IP،وport numer للخدمة وبروتوكول طبقة النقل TCP أو UDP).

يوجد socket خاصه بال server وكذلك socket خاصة بكل client حيث تتكون socket السيرفر من عنوان IP(في مشروعنا سنفرضه local host أي 127.0.0.1)، و Port number(سوف نختاره بحيث لا تشغله أي خدمة)،وبروتوكول طبقة النقل(سوف نستخدم TCP)،إما الsocket الخاصة بال client فهي تتكون من عنوان ال IP الخاص بال server،ورقم ال port ويتم اختياره بشكل عشوائي،وبرتوكول طبقة النقل TCP(نفسه المستخدم في مقيس ال server).

الفرق بين بروتوكول UDPوTCP هو أن الTCP هو برتوكول اتصال موثوق أي يتحقق فيما إذا كان الهدف قد استقبل الرسالة وفي حال لم يستقبلها يقوم بإعادة أرسالها من جديد،بينما الUDP يقوم بالإرسال بشكل متواصل ودون أي تحقق.

في الحالة الطبيعية فإن كل سيرفر بإمكانه تخديم client واحد فقط خلال فترة زمنية معينة، ويكون عدد من الclient في حالة استماع إما بقية ال client فيتم رفضهم مباشرةً، وعند انتهاء الclient الحالي يبدأ الserver بتخديم client جديد من ال clients المنتظرة وتستمر العملية بهذه الطريقة، وهنا يظهر لنا مدى التفاعلية بين الserver وال clients (الزمن الطويل الذي سوف يستغرقه السيرفر لتخديم جميع الclients) وهنا تكمن أهمية مفهوم multi threading.

المقصود بال multi threading هو أن السيرفر سوف ينشئ thread خاص بكل client يتصل به ويتعامل معه من خلاله وذلك بدلاً من أن يتم التعامل مع جميع ال clients من خلال thread واحد مما يجعل التطبيق أكثر تفاعلية ويسرع الاستجابة، وكمثال فإن عند استخدام المتصفح chrom فإن تنزيل الملفات يكون ضمن thread وتصفح الأنترنت يكون ضمن thread أخرى وبالتالي لو حدث خطأ في عملية التنزيل لن يتوقف عمل الchrom بل استطيع متابعة عملية تصفح الأنترنت.

يوجد لدى ال python ما يسمى الGIL والذي يسمح بالقيام بالmulti threading،حيث أنه يضمن عدم وصول أكثر من thread إلى نفس الموقع في الذاكرة،أي أنه وبالرغم من العمل التفرعي إلا أنه في لحظه واحدة thread واحد هو الذي يخدم أما البقية فيتم عمل lock لهم من خلال GIL.

اي باختصار يمكن تمثيل المشروع كشخص يملك موبايل (SERVER) ويتلقى رسائل ن جهات اتصال لديه (clients) ويرد على كل محادثة من نافذة خاصة (GUI).

اهمية البحث واهدافه:

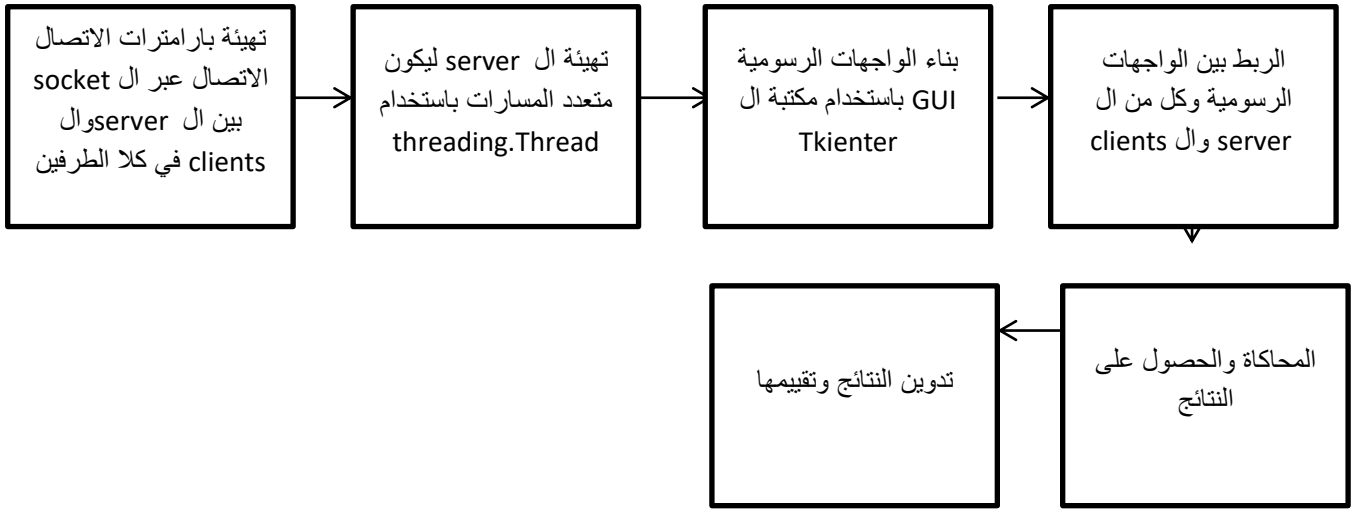
نهدف في هذا المشروع الى شرح الية الاتصال الذي يحصل من خلال ال socket الموجودة في اعلى طبقة النقل وذلك من خلال تطبيق دردشة بين server ومجموعة من ال clients في آن معا وسوف نستخدم ال multithreading لتحقيق ذلك وسنقوم باظهار كيفية تبادل الرسائل من خلال الواجهات الرسومية باستخدام مكتبة ال tkienter في البايثون مما يجعل العملية اكثر تفاعلية واسهل في الفهم.

منهجية العمل:

سوف نستخدم لغة البرمجة python لمحاكاة هذا المشروع وسوف نعتمد على اكثر من مكتبة ابرزها:

- مكتبة ال socket : توفر لنا جميع ما نحتاجه من اجل الاتصال عبر طبقة النقل مثل اختيار البروتكول (tcp,udp) وتحديد العناوين والكثير من البارامترات المتعلقة بالاتصال.
- مكتبة ال Tkienter : توفر لنا البنية المناسبة لبناء واجهات رسومية بسيطة وفعالة بنفس الوقت (شكل الازرار ,ابعادها و مكان اظهار الرسائل الواردة).
- مكتبة ال time: للحصول على الزمن بدقة (الزمن –التاريخ).
- موديول threading : من خلاله سوف نقوم ببناء المسارات من اجل الحصول على بنية متعددة المسارات Multithreading وذلك لكي يتمكن ال server من الاتصال مع اكثر من client في نفس الوقت.

يوضح المخطط الصندوقي التالي تسلسل الخطوات التي سنتبعها في محاكاة هذا النظام:



الشكل(1): مخطط خطوات العمل في المشروع

تم اجراء البحث في كلية الهندسة الميكانيكية والكهربائية – قسم هندسة الاتصالات والالكترونيات في العام الدراسي 2020-2021م.

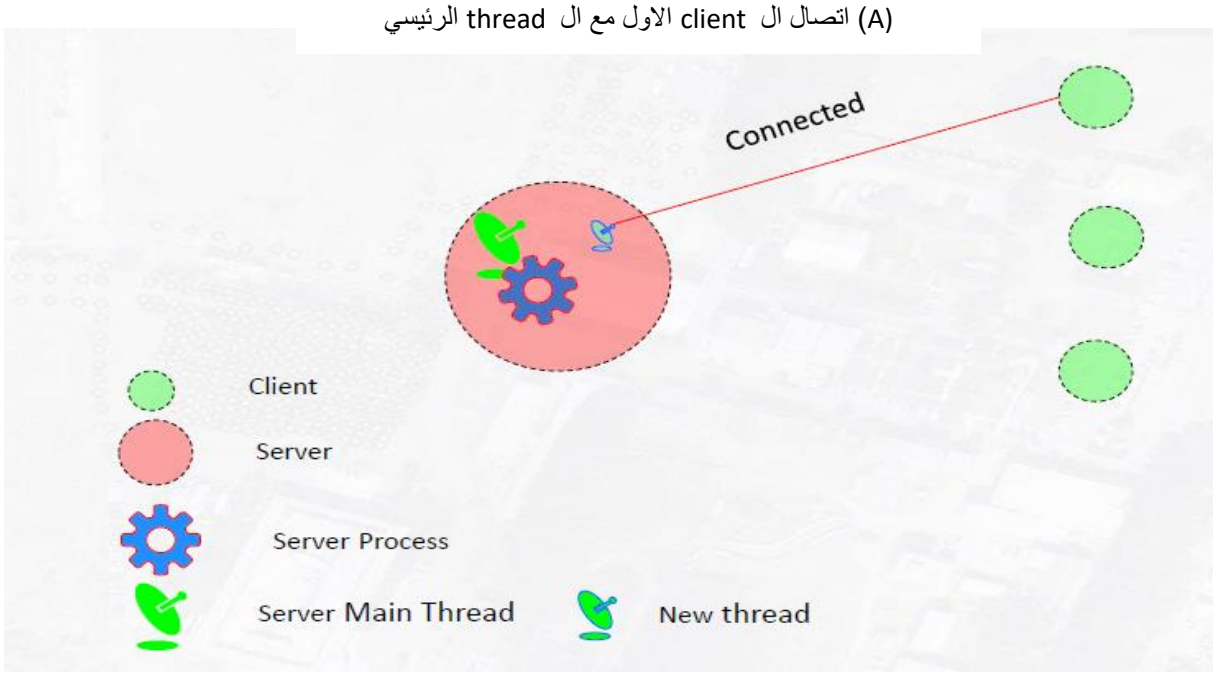
-تعدد المسارات:

في الحالة الاعتيادية سوف يكون هناك مقبس واحد فقط لكل server ومقبس مقابل في كل client وعندما يريد client الاتصال بال server فان الاتصال يتم عن طريق مسار بين المقبسين ولا يمكن بعدها للسيرفر استقبال اي اتصال من client جديد حتى ينتهي اتصال ال client الذي يخدمه وذلك بسبب عدم وجود الا مقبس واحد فقط في ال server كما ذكرنا.

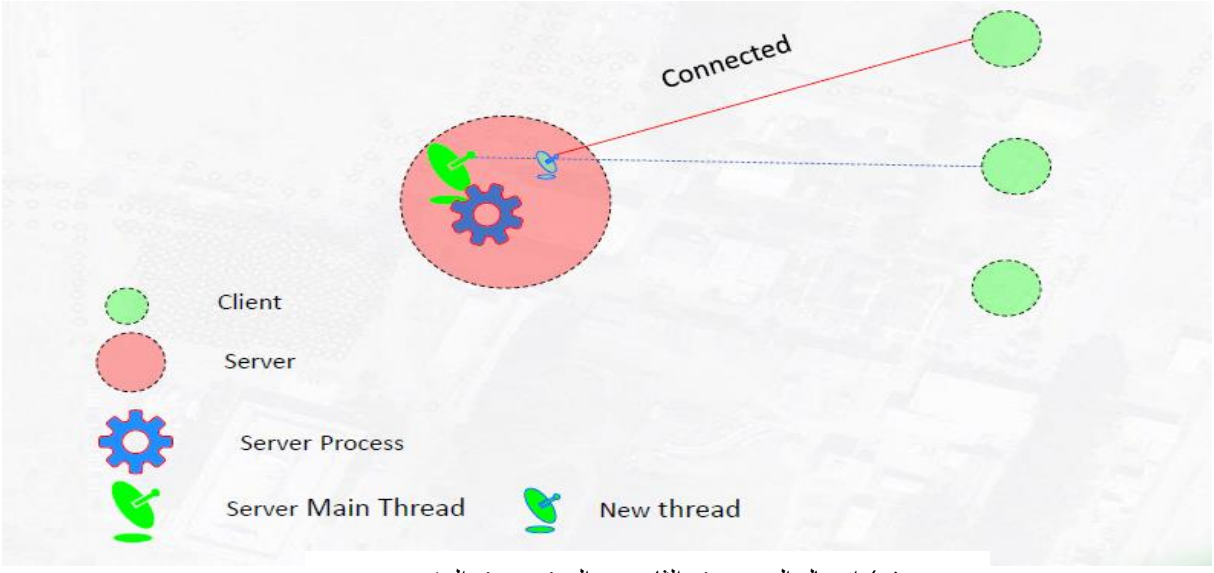
في حالة تعدد المسارات يختلف الامر حيث ان كل server سوف يكون له اكثر من مقبس يتم انشاؤها مع كل اتصال جديد من قبل client حيث انه في البداية يكون في ال server مقبس واحد هو المقبس الرئيسي وعندما يريد ال client الاتصال به يتكون مسار thread بين مقبس ال client ومقبس ال server الرئيسي ليقوم بعدها ال server بانشاء مقبس خاص بال client هذا ويحول اتصاله عليه وبهذا يكون قد تكون لدينا مسار جديد وتتكرر العملية السابقة من اجل كل client , وعندما ينتهي ال client من اتصاله يتم حذف مساره وبالتالي عند اتصاله مجددا سوف يمنح مسار جديد.

توضح الاشكال التالية تسلسل اتصال ال clients مع ال server في حالة تعدد المسارات

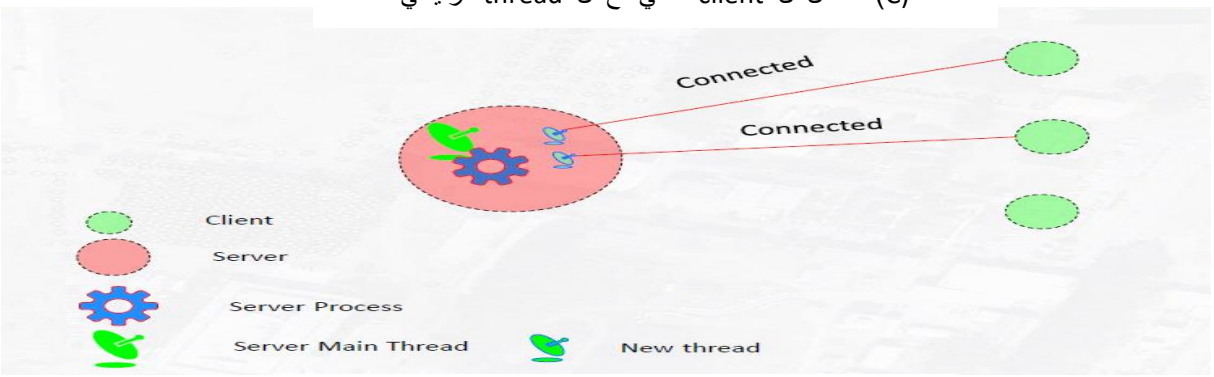
(A) اتصال ال client الاول مع ال thread الرئيسي



(B) تحويل اتصال ال client الاول الى thread جديد خاص به



(C) اتصال ال client الثاني مع ال thread الرئيسي

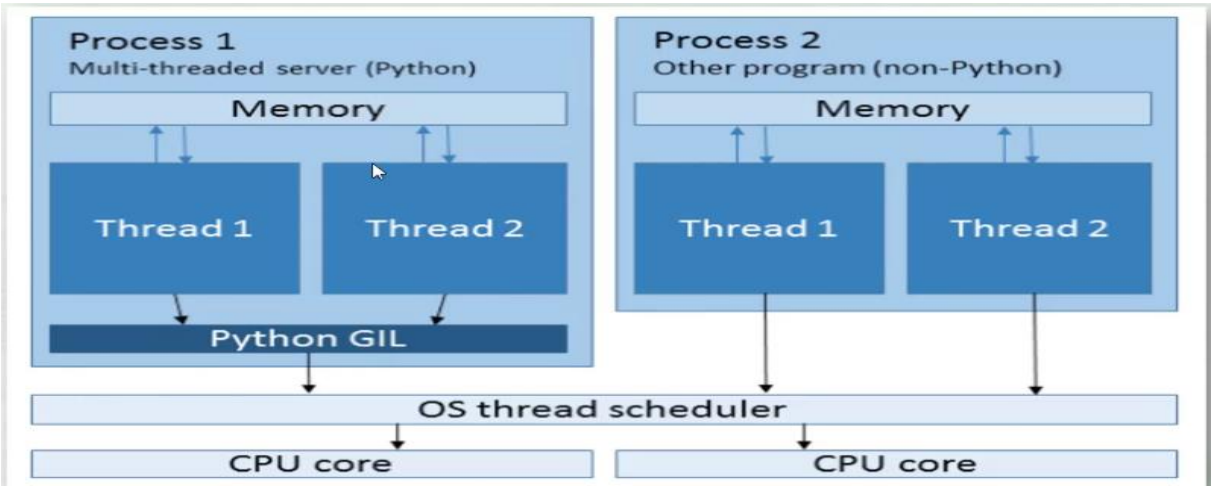


(D) كل من ال client الاول والثاني متصلان بالسيرفر ويعملان معا

الشكل(2): خطوات الاتصال بين ال clients وال server متعدد المسارات

تؤمن عملية تعدد المسارات فصل بين ال clients وبالتالي بامكان ال serverالتعامل مع اكثر من client في نفس الوقت ولكنها لا تؤمن عمل تفرعي بشكل كامل وذلك بسبب وجود ما يسمى GIL(global interpreter lock) في البايثون والتي تكون مهمتها هي منع وصول اكثر من thread الى نفس موقع الذاكرة بنفس اللحظة ,ولكن بالنسبة لمشروعنا فان وجودها لن يكون ذو تأثير .

يوضح الشكل التالي مقارنة بين ال multithreading في حال وجود ال GIL كما في البايثون وحالة عدم وجودها:



الشكل(3): مقارنة بين تعدد المسارات في حال وجود GIL وفي حال عدم وجودها

-تهيئة ال server لاتصال ال socket :

بعد استدعاء المكتبات التي ذكرناها, نقوم بداية بتحديد ال IP وال port number الذي سوف يعمل عليهما ال server وفي مشروعنا سوف نشغله على عنوان ال IP لل local host وهو 127 .0.0.1 اما عنوان ال port فهو اختياري شريطة الا تكون تعمل عليه اي خدمة اخرى وسوف نحدد العدد الاعظمي للمسارات التي يبنيها ال server ب 6 مسارات اي يمكنه تخديم 6 clients في آن معا.

نحدد برتوكول طبقة النقل TCP حيث سوف نعمل على بروتوكول النقل الموثوق ومن ثم نقوم بتعريف توابع لارسال واستقبال الرسائل من كل client .

-تهيئة ال clients لاتصال ال socket :

- نستدعي المكتبات اولا ومن ثم نحدد بروتوكول النقل TCP بشكل متوافق مع بروتكول النقل في الserver
- نحدد عنوان ال IP و ال port number للسيرفر الوجهة وفي مشروعنا يكون IP الوجهة هو 127.0.0.1 وال port number يجب اختياره نفسه الذي وضعناه للسيرفر.
- تعريف توابع لارسال واستقبال الرسائل من ال server .
- يجب الانتباه هنا ان رقم ال port الذي وضعناه هو رقم ال port للسيرفر وليس لل client حيث ان رقم ال port لل client لا يتم تحديده ولا اختياره وانما يؤخذ بشكل عشوائي .
- تكون الخطوات السابقة هي نفسها من اجل كل client .

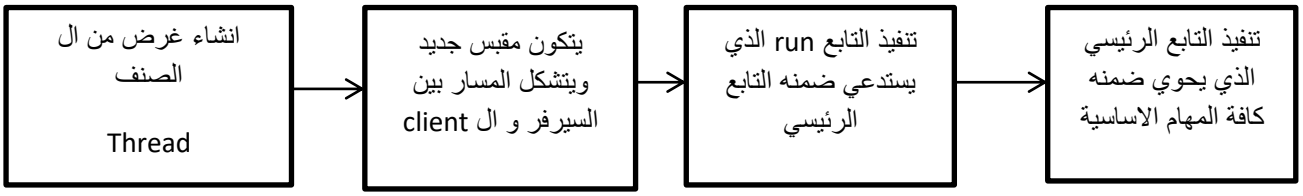
-تهيئة ال server من اجل ال multithreading :

سوف نقوم بتعريف صنف جديد يرث الصنف (threading.thread) وسوف نقوم بانشاء غرض منه عند كل اتصال جديد ل client مع السيرفر وذلك من خلال حلقة لا نهائية.

عند انشاء كل غرض من هذا الصنف يتم تنفيذ تابع run والذي بدوره يستدعي التابع الرئيسي الذي يحتوي على جميع المهام التي يقوم بها السيرفر (ارسال-استقبال –قطع اتصال ...).

مع كل انشاء ل thread فان هذا يعني انشاء مقبس جديد للسيرفر يكون خاص بال clientالذي انشئ له ال thread (ال clientالذي اتصل لتوه).

يوضح الشكل التالي تسلسل عمليات تهيئة السيرفر :



الشكل (4): تسلسل عمليات تهيئة السيرفر من اجل تعدد المسارات

-آلية الاتصال :

عند تشغيل ال client فانه يطلب مباشرة عنوان السيرفر فاذا كان السيرفر الذي يطلبه غير مشغل فان الاتصال سوف يفشل اما في حال كان السيرفر مشغلا سوف ينجح الاتصال ويصبح هناك مسار خاص لتبادل البيانات بين هذا ال client والسيرفر كما ذكرنا.

-امكانيات ال server في مشروعنا:

- ان السيرفر غير قادر على البدء في المحادثة مع ال client بل مهمته تكون هي الاستجابة على رسائل ال client .
- بمجرد وصول رسالة الى السيرفر من client معين سوف تظهر لديه واجهة خاصة بهذا ال client يستطيع من خلالها قراءة رسائل ال client وكتابة رسائل للرد عليه.
- لن تظهر رسائل ال client التالية ما لم يضع السيرفر نفسه في وضع الاستعداد ولكن سوف تجمع الرسائل التي لم تصل لتصل دفعة واحدة عند الدخول في وضع الاستعداد .
- عندما يدخل السيرفر في وضع الاستعداد ستختفي نافذته الى حين وصول رسالة من ال client .

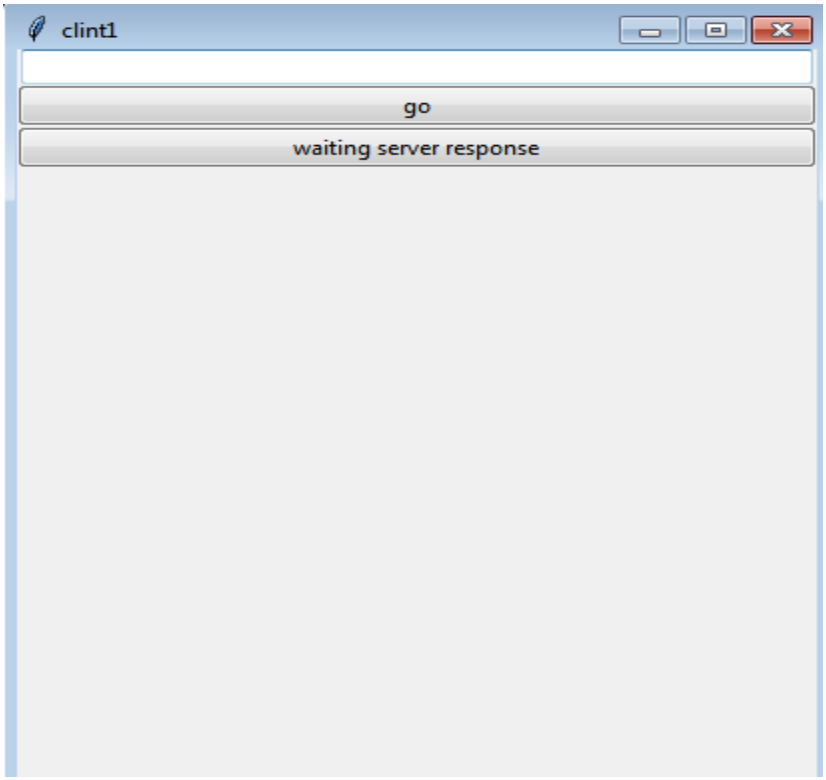
- يمكن للسيرفر ان يرد بالوقت على ال client وذلك اعتمادا على مكتبة time التي تكلمنا عنها
- بإمكان السيرفر قطع اتصال اي client متى ما اراد او حين يطلب ال client ذلك بنفسه.

امكانيات ال client في مشروعنا

- بمجرد تشغيل ال client فإنه سوف يتصل مباشرةً مع الserver وسوف تظهر واجهه رسومية له، تتيح له إرسال رسالة.
- بعد أن ال client الرسال ويضغط على Go لكي تُرسل فإنه يجب أن يضغط على زر الانتظار كي يتلقى الرد من ال server ففي حال لم يفعل ذلك لن تصله رسالة ال server
- وعند رد ال server على ال client سوف تنبثق نافذة جديدة تتضمن label لتظهر عليها الرسالة.
- بعد قراءة الرسالة إذا أراد ال server تكرار العملية يضغط على زر I want to send فتعود تظهر له النافذة الأولى (نافذة الإرسال).

-الواجهات الرسومية لل client :

1- واجهة الارسال: هي الواجهة التي تظهر بمجرد تشغيل ال client و اتصاله بال server وهي مخصصة لكتابة الرسائل وارسالها لل server وهي موضحة بالشكل التالي:

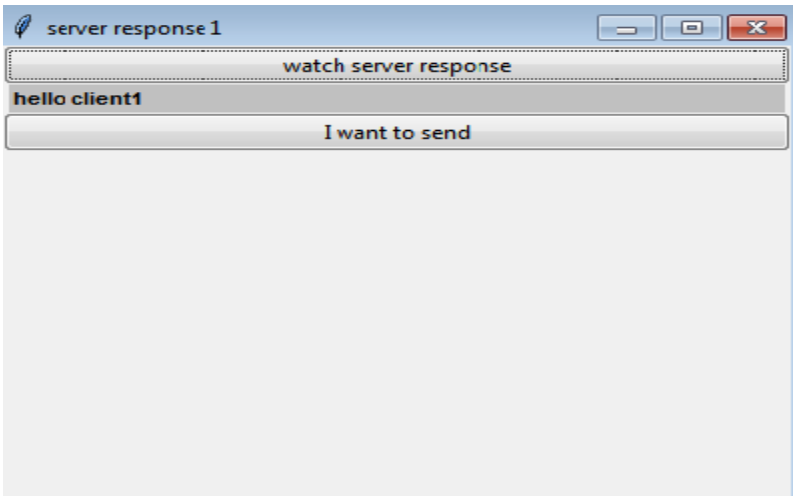


الشكل(5): واجهة الارسال لل client

محتويات هذه الواجهة:

- حقل ادخال : تكتب فيه الرسالة التي يريد ال client ارسالها الى ال server
- زر go : بعد كتابة الرسالة يتم الضغط عليه كي يتم ارسالها الى ال server
- زر waiting server response :عند الضغط عليه اي ان ال client جاهز لاستقبال رسالة ال server وينتظرها

2- واجهة server response: تظهر عند وصول رد من ال sever على رسالة كان قد ارسلها ال client مسبقا ويوضح الشكل التالي هذه الواجهة:



الشكل(6): واجهة server response لل client

محتويات هذه الواجهة:

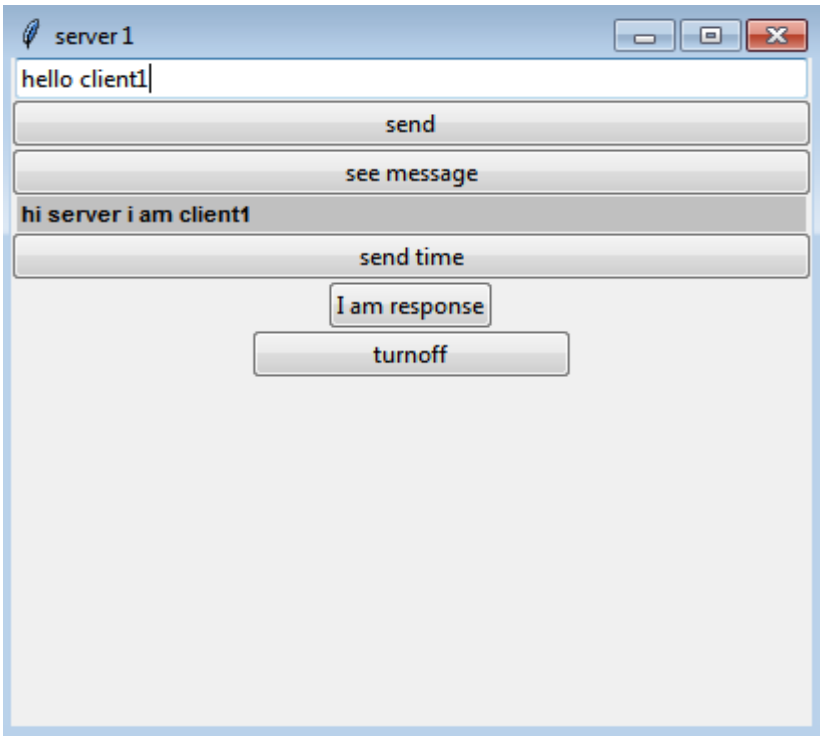
- زر watch server response : عند الضغط عليه تظهر استجابةارسالة السيرفر في الحقل اسفله مباشرة .

- حقل اظهار : تظهر فيه الرسالة القادمة من السيرفر عند الضغط على زر watch server response .

- زر I want to send : في حال اراد ال client الارسال مجددا يضغط على هذا الزر فتظهر له نافذة الارسال .

-الواجهة الرسومية لل server :

يوضح الشكل التالي الواجهة التي ستظهر للسيرفر عند اتصال client به:



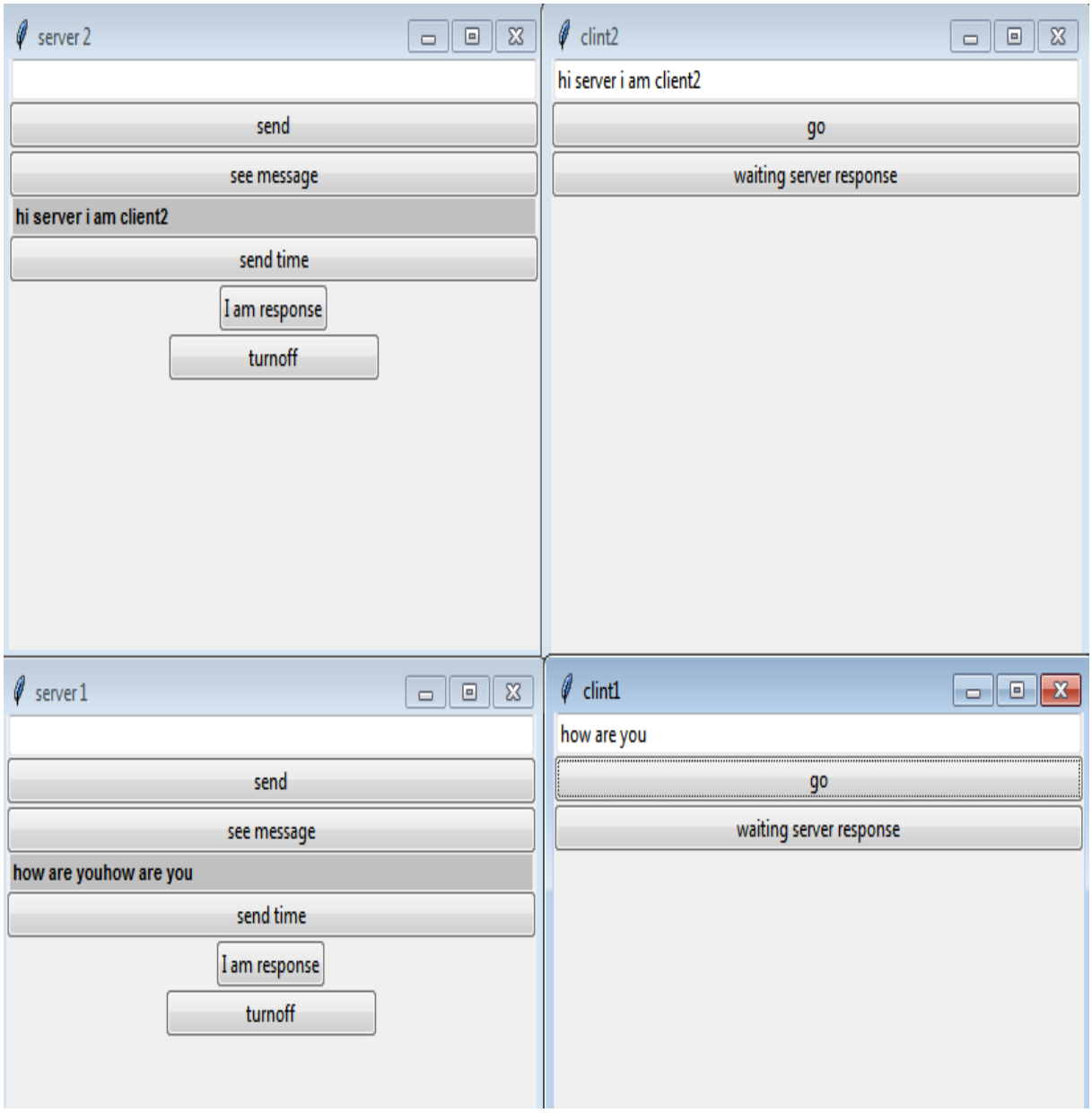
الشكل (7): الواجهة الرسومية لل server

محتويات هذه الواجهة:

- حقل الادخال: يكتب فيه السيرفر الرسالة التي يريد ارسالها الى ال client
- زر send : بعد كتابة الرسالة في حقل الادخال يضغط على هذا الزر لكي يتم ارسالها.
- زر see message : عند الضغط عليه تظهر رسالة ال client في الحقل اسفله مباشرة
- حقل اظهار : تظهر فيه الرسالة القادمة من ال client بمجرد الضغط على زر see message
- زر send time : عند الضغط عليه يتم ارسال الوقت مباشرة الى ال client
- زر I am response : الضغط عليه يؤدي الى اختفاء واجهة السيرفر ويعني ان السيرفر قد استجاب لرسالة ال client وهو جاهز لاستقبال رسائل اخرى
- زر turn off : بالضغط عليه يقوم السيرفر بقطع اتصاله مع هذا ال client

اتصال اكثر من client مع ال server :

يوضح الشكل التالي كيف ان السيرفر لديه واجهتين بنفس الوقت لكل client هناك واجهة يرد عليه من خلالها وهذا ناتج عن تعدد المسارات كما ذكرنا سابقا.



الشكل(8) : الواجهات الرسومية لاتصال client 2 مع ال server

المناقشة والتوصيات:

- إن استخدام تعدد المسارات وفر الكثير من الوقت الذي كان سينظره كل client حتى تتاح له فرصة الاتصال مع ال server وذلك لأن ال server بإمكانه تخديم أكثر من client معاً وهذا ما يوفر الوقت.
- بالإضافة إلى الوقت فإن هنالك إيجابية مهمه جداً لعملية تعدد المسارات وهي التكلفة المادية ففي التطبيقات أو المواقع التي يصل لها الزبائن بعدد كبير يومياً، فإنها لا تستطيع إنشاء سيرفر لكل client بسبب التكلفة الكبيرة لذلك.
- مع استخدام multi threading أصبح بإمكان كل server أن يخدم عدد كبير من الزبائن في وقت واحد.
- أضافة الواجهة الرسومية GUI مرونة و تفاعلية أكثر في التعامل مع كل من ال server وال client ، فبدلاً من استخدام محاكي ال python لكتابة وقراءة الرسائل، أصبحنا نستخدم الواجهات الرسومية كأي تطبيق موجود لدينا على الحاسب.
- ان استخدام ال multithreading مفيد الى حد معين ولكن عند عدد كبير لل clients سوف يكون من الصعب على السيرفر انشاء مسار خاص بكل client

اقتراحات لتطوير المشروع:

- اتاحة امكانية البدء بالمحادثة للسيرفر بعد ان يتصل به ال client .
- استخدام مكتبة scrolledtext في البايثون لانشاء مجدول للرسائل اي يصبح السيرفر قادرا على عرض جميع الرسائل التي ارسلها ال client اليه وجميع ردوده عليها, ونفس الامر بالنسبة لل client.
- تطوير السيرفر بحيث يملك العديد من ميزات الاستجابة السريعة (مشابهة للاستجابة للوقت الموجودة في مشروعنا)
- تطوير المشروع ليصبح غرفة دردشة.

المراجع.....

(J.ortega, learning python networking, 2015)

(J.ortega, learning python networking-second edation, 2019)