

Socket programming

資工三 A 110403516 李倬安

我使用 python 的 socket 套件完成 TCP socket，並實現基本聊天應用程序，並且結合了使用 Tkinter 製作的圖形用戶界面（GUI）。以下是其功能介紹：

- ✧ 基本 TCP socket 連線
- ✧ 為 client、server 創立各自 thread
- ✧ 可處理多 client 連線(multithreading)
- ✧ Non blocking socket
- ✧ GUI 介面
- ✧ 以登入確認方式才可以進行建立 client thread(目前預設用戶為 admin, amy, bob)
- ✧ 分別記錄每個 client 和 server 的聊天歷史紀錄，各個 client 只能看到自己的聊天歷史紀錄
- ✧ 如果登入錯誤，或是 message submit 空白的話，皆會跳出警告視窗
- ✧ 每則 client 和 server 的聊天訊息皆會加上時間戳
- ✧ Terminal 會記錄何時 server 成功建立，或是何時 client 登入成功

Simple System Demo:



1.

```
import socket
import tkinter as tk
from tkinter import messagebox
import threading
import select
import datetime
import time

client_credentials = {"username": "", "password": ""}
admin_message_history = []
amy_message_history = []
bob_message_history = []
```

一開始先導入了需要的 python 套件，之後定義 global 變數和相關的數據結構以便紀錄計算，用於存儲 client 的身份信息和聊天歷史記錄。

2.

```
def server():
    host = ""
    port = 12343
    print("Server started!")
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((host, port))
    server_socket.listen(5)

    while True:
        readable, _, _ = select.select([server_socket], [], [], 0.1)
        for s in readable:
            if s is server_socket:
                c, addr = server_socket.accept()
                new_thread = threading.Thread(
                    target=handle_client, args=(c,)
                ) # create new thread for client
                new_thread.start()
```

- host 和 port 變數：這裡定義了服務器的 host 和 port。host 設置為空字串 ""，表示服務器將監聽所有可用的網路介面。port 設置為 12343，表示服務器將在該 port 上監聽連接請求。
- 創建服務器 socket：使用 socket.socket(socket.AF_INET, socket.SOCK_STREAM) 創建了一個 TCP socket server_socket，它使用 IPv4 地址 (socket.AF_INET) 和 TCP 協議 (socket.SOCK_STREAM)。
- 綁定 socket 到 host 和 port：通過調用 server_socket.bind((host, port))，將服務器 socket 綁定到指定的 host 和 port 上。將所有傳入的連接請求路由到指定的 host 和 port。
- 啟動服務器：通過調用 server_socket.listen(5) 啟動服務器，開始監聽傳入的連接請求，5 表示服務器可以排隊等待的最大連接數。
- 進入服務器主循環：進入一個無限循環，這個循環用於不斷地等待 client 的連接請求。
- 使用 select.select 監聽連接請求：使用 select.select 函數監聽 socket 是否有可讀事件。這裡傳入了 server_socket 列表作為待監聽的 socket，空列表表示不關注可寫和異常事件，最後一個參數 0.1 表示超時時間為 0.1 秒，即非阻塞等待連接請求。
- 處理新連接：如果 server_socket 變為可讀（即有新的連接請求到來），則進入循環。使用 server_socket.accept() 來接受 client 的連接，返回一個新的 socket，c 和 client 的地址 addr。
- 創建 new thread 處理 client 通信：為了能夠同時處理多個 client 連接，通過創建新線程的方式來處理每個 client。這裡使用 threading.Thread 創建了一個新的線程，目標函數

為 `handle_client`，傳遞了 `client socket c` 作為參數。然後啟動線程。

3.

```
def handle_client(client_socket):
    while True:
        data = client_socket.recv(1024)
        if not data:
            break

        if data == b"hello":
            response = "Hi! What's your name?"
        else:
            response = "Hello, " + str(data.decode("ascii")) + "!"
        current_time = datetime.datetime.now().strftime("%m-%d %H:%M:%S")
        if client_credentials["username"] == "admin":
            admin_message_history.append(
                current_time + "\t\t" + "server" + " : " + response
            )
        elif client_credentials["username"] == "amy":
            amy_message_history.append(
                current_time + "\t\t" + "server" + " : " + response
            )
        elif client_credentials["username"] == "bob":
            bob_message_history.append(
                current_time + "\t\t" + "server" + " : " + response
            )
        update_message_history()
        client_socket.send(response.encode("ascii"))
        client_socket.close()
```

- `def handle_client(client_socket)`：這是一個函數定義，它接受 `client_socket` 作為參數，用於與 `client` 進行通信。
- `while True`：進入一個無限循環，用於不斷地接收來自 `client` 的消息並進行處理，直到 `client` 斷開連接或發送空消息。
- `data = client_socket.recv(1024)`：使用 `client_socket.recv(1024)` 接收來自 `client` 的消息，每次最多接收 1024 位元組的數據。接收到的數據被存儲在 `data` 變量中。
- `if not data`：檢查是否接收到了空消息（消息長度為 0），或可以表示 `client` 已斷開連接，跳出循環。
- `if data == b"hello"`：檢查接收到的消息是否為 "hello"，如果是，表示 `client` 向服務器發送了 "hello"。如果接收到的消息是 "hello"，則構造一個回應消息 `response` 為 "Hi! What's your name?"。如果接收到的消息不是 "hello"，則構造一個回應消息 `response` 為 "Hello, [用戶名]!"，其中 [用戶名] 由 `client` 發送的消息解碼得到的。
- 獲取當前時間 `current_time`：使用 `datetime.datetime.now().strftime("%m-%d %H:%M:%S")` 獲取當前時間，並格式化為 "月-日 時:分:秒" 的字元串。
- 根據 `client` 用戶名更新消息歷史記錄：根據 `client_credentials["username"]` 的值判斷 `client` 的用戶名（admin、amy、bob）。將當前時間、服務器、響應消息構成的字元串添加到相應用戶的消息歷史記錄中。例如，如果用戶名是 "admin"，則將記錄添加到 `admin_message_history` 中。

- 調用 `update_message_history()`：這個函數用於更新消息歷史記錄，將服務器的回答添加到 GUI 中顯示的消息歷史文本框中。
- `client_socket.send(response.encode("ascii"))`：將回應消息 `response` 編碼為 ASCII 並通過 `client_socket` 發送給 client。
- `client_socket.close()`：關閉 client socket，結束與 client 的通信。

4.

```
def client(message):
    host = "127.0.0.1"
    port = 12343
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.setblocking(False)
    try:
        s.connect((host, port))
    except BlockingIOError:
        pass
    counter = 0
    max_iterations = 5
    while True:
        readable, _, _ = select.select([s], [], [], 0.1)
        if not readable:
            print("No new connections during this interval.")
            time.sleep(1)
            counter += 1
            if counter >= max_iterations:
                break
        else:
            for sock in readable:
                if sock is s:
                    try:
                        data = s.recv(1024)
                        if not data:
                            s.close()
                            print("Connection closed")
                            return
                    except ConnectionResetError:
                        s.close()
                        print("Connection reset")
                        return
                    try:
                        s.send(message.encode("ascii"))
                    except BrokenPipeError:
                        s.close()
                        print("Broken pipe")
                        return
```

- `def client(message)`：這是一個函數定義，它接受一個參數 `message`，表示要發送給服務器的 parameter。
- `host` 和 `port` 變量：這裡定義了服務器的主機地址為 "127.0.0.1"，表示連接到本地計算機上運行的服務器。`port` 設置為 12343，表示要連接到服務器的埠號。
- 創建客戶端套接字：使用 `socket.socket(socket.AF_INET, socket.SOCK_STREAM)` 創建了一個 TCP 套接字 `s`，它使用 IPv4 地址 (`socket.AF_INET`) 和 TCP 協議 (`socket.SOCK_STREAM`)。
- 設置非阻塞模式：通過 `s.setblocking(False)` 設置套接字為非阻塞模式。這意味著在沒有數據可讀或可寫時，不會阻塞線程，而是立即返回，以便可以執行其他任務。
- 連接到服務器：通過 `s.connect((host, port))` 嘗試連接到指定的 `host` 和 `port`。如果連接成功，繼續後續的通信；如果連接失敗，拋出 `BlockingIOError` 異常，但非阻塞模式下連接可能不會立即成功。
- 進入客戶端主循環：進入一個無限循環，用於不斷地確認可讀事件，以便接收來自服務器的

消息，並嘗試發送消息給服務器。

- 使用 `select.select` 監聽可讀事件：使用 `select.select` 函數監聽套接字是否有可讀事件。這裡傳入了 `[s]` 列表作為待監聽的套接字，空列表表示不關注可寫和異常事件，最後一個參數 `0.1` 表示超時時間為 `0.1` 秒，即非阻塞等待可讀事件。
- 處理不可讀事件：如果沒有可讀的套接字，會印出 "No new connections during this interval." 通知使用者，之後會確認時間，達到最大循環次數就退出循環。
- 處理可讀事件：如果套接字 `s` 變為可讀（即有數據從服務器傳輸到客戶端），則進入循環。使用 `s.recv(1024)` 接收來自服務器的消息，每次最多接收 `1024` 位元組的數據。如果接收到的數據為空（消息長度為 `0`），表示服務器已斷開連接，關閉套接字並返回，結束通信。
- 嘗試發送消息：如果套接字可以寫入（即可發送消息給服務器），則嘗試發送 `message` 給服務器，通過 `s.send(message.encode("ascii"))` 將消息編碼為 ASCII 並發送。
- 異常處理：如果發生連接重置（`ConnectionResetError`）或管道中斷（`BrokenPipeError`）等異常，都會關閉套接字並列印相應的錯誤信息，並退出循環。
- `BrokenPipeError`：如果一方關閉了套接字（通常是先關閉的一方），而另一方仍然嘗試向該套接字發送數據，就會觸發 `BrokenPipeError`。這表示通信的一方已經不再接受數據，發送數據會失敗。或是說如果發送端發送數據速度過快，而接收端無法處理這麼快的速度，套接字緩沖區可能會被填滿。當發送端嘗試將數據寫入已滿的緩沖區時，也會觸發 `BrokenPipeError`。

5.

```
def send_message():
    message = message_entry.get()
    current_time = datetime.datetime.now().strftime("%m-%d %H:%M:%S")
    if message:
        if client_credentials["username"] == "admin":
            admin_message_history.append(
                current_time + "\t\t" + "admin" + " : " + message
            )
        elif client_credentials["username"] == "amy":
            amy_message_history.append(current_time + "\t\t" + "amy" + " : " + message)
        elif client_credentials["username"] == "bob":
            bob_message_history.append(current_time + "\t\t" + "bob" + " : " + message)
        update_message_history()
        # update_message_history(client_credentials["username"] + " : " + message)
        threading.Thread(target=client, args=(message,)).start()
        message_entry.delete(0, tk.END)
    else:
        messagebox.showwarning("Input Error", "Please enter a message!")
```

- `def send_message()`：這是一個函數定義，用於發送消息到服務器。
- `message = message_entry.get()`：從 GUI 界面中的輸入框 `message_entry` 中獲取用戶輸入的消息，並將其存儲在 `message` 變數中。
- `current_time = datetime.datetime.now().strftime("%m-%d %H:%M:%S")`：獲取當前時間，並以 "月-日 時:分:秒" 的格式將其存儲在 `current_time` 變數中。
- `if message`：檢查用戶輸入的消息是否非空。
- 如果消息非空，執行以下操作：使用 `client_credentials["username"]` 獲取當前用戶的用戶名。根據不同的用戶名將消息添加到不同的消息歷史記錄中（`admin_message_history`、`amy_message_history` 或 `bob_message_history`），根據用戶名、當前時間、消息內容構建一條消息記錄，並將其添加到相應的歷史記錄中。

- 調用 `update_message_history()` 函數更新消息歷史記錄，以便在 GUI 中顯示最新的消息。創建一個新的線程，使用 `threading.Thread()`，並將 `client` 函數作為目標，同時傳遞 `message` 參數給 `client` 函數。這個新線程用於在後台與服務器進行通信，以避免阻塞 GUI 界面。
- 清空消息輸入框：使用 `message_entry.delete(0, tk.END)`，以便用戶可以輸入下一條消息。
- 如果用戶輸入的消息為空：使用 `messagebox.showwarning()` 彈出一個警告框，提醒用戶輸入有效的消息。

6.

```
def login():
    login_window = tk.Toplevel(root)
    login_window.title("Login")

    username_label = tk.Label(login_window, text="Username:")
    username_label.pack(pady=10)
    username_entry = tk.Entry(login_window, width=30)
    username_entry.pack(pady=5)

    password_label = tk.Label(login_window, text="Password:")
    password_label.pack()
    password_entry = tk.Entry(login_window, width=30, show="*")
    password_entry.pack(pady=10)

    def authenticate():
        username = username_entry.get()
        password = password_entry.get()

        if (
            (username == "admin" and password == "admin")
            or (username == "amy" and password == "amy")
            or (username == "bob" and password == "bob")
        ):
            print("Login successful!")
            message_history_text.delete(1.0, tk.END)
            client_credentials["username"] = username
            client_credentials["password"] = password
            login_window.destroy()
            send_message_button.config(state="normal")
            message_history_text.config(state="normal")
            message_history_text.delete(1.0, tk.END)
            message_history_text.config(state="disabled")
        else:
            messagebox.showerror(
                "Authentication Error", "Invalid username or password."
            )

    login_button = tk.Button(login_window, text="Login", command=authenticate)
    login_button.pack()
```

- `def login()`：這是一個函數定義，用於處理用戶登錄。
- `login_window = tk.Toplevel(root)`：創建一個頂級視窗 `login_window`，作為登錄視窗的容器，並將其關聯到主視窗 `root`。
- `login_window.title("Login")`：設置登錄視窗的標題為 "Login"。
- 創建用戶名和密碼輸入框：`username_label` 是一個標籤，顯示 "Username:" 文本，用於指示用戶輸入用戶名。`username_entry` 是一個文本輸入框，用戶可以在其中輸入用戶名。
- `password_label` 是一個標籤，顯示 "Password:" 文本，用於指示用戶輸入密碼。
- `password_entry` 是一個密碼輸入框，用戶可以在其中輸入密碼，輸入的字元會被顯示為 "*"，以保護密碼的安全性。
- 創建 `authenticate()` 函數：這個函數用於進行用戶身份驗證，驗證用戶輸入的用戶名和密

碼是否有效。通過 `username_entry.get()` 和 `password_entry.get()` 獲取用戶輸入的用戶名和密碼。使用條件語句檢查用戶名和密碼是否匹配已知的用戶名和密碼組合（在這里，用戶名和密碼設定為 "admin"、"amy"、"bob"）。如果用戶名和密碼匹配其中一個組合，顯示 "Login successful!" 並執行以下操作：清空消息歷史記錄文本框，以便在登錄後顯示新的聊天記錄。更新 `client_credentials` 字典中的用戶名和密碼。銷毀登錄視窗，關閉登錄對話框。啟用發送消息按鈕 `send_message_button` 和消息歷史記錄文本框 `message_history_text`，以使用戶可以開始發送消息。清空消息歷史記錄文本框以準備顯示新的消息記錄。如果用戶名和密碼不匹配任何已知的組合，使用 `messagebox.showerror()` 顯示身份驗證錯誤的消息框，提示用戶輸入有效的用戶名和密碼。創建登錄按鈕 `login_button`，當用戶點擊該按鈕時，將執行 `authenticate()` 函數進行身份驗證。

7.

```
def update_message_history():
    message_history_text.config(state="normal")
    current_user_message_history = []
    if client_credentials["username"] == "admin":
        current_user_message_history = admin_message_history
    elif client_credentials["username"] == "amy":
        current_user_message_history = amy_message_history
    elif client_credentials["username"] == "bob":
        current_user_message_history = bob_message_history

    message_history_text.delete(1.0, tk.END)
    for message in current_user_message_history:
        message_history_text.insert(tk.END, f"{message}\n")

    message_history_text.config(state="disabled")
```

- `def update_message_history()`：這是一個函數定義，用於更新消息歷史記錄的顯示。
- `message_history_text.config(state="normal")`：設置消息歷史記錄文本框 `message_history_text` 的狀態為 "normal"，以允許對其進行修改。
- `current_user_message_history = []`：創建一個空列表 `current_user_message_history` 用於存儲當前用戶的消息歷史記錄。使用條件語句檢查當前用戶的用戶名（存儲在 `client_credentials["username"]` 中），並根據不同的用戶名將相應的消息歷史記錄賦值給 `current_user_message_history` 變量。例如，如果用戶名為 "admin"，則將 `admin_message_history` 賦值給 `current_user_message_history`。
- `message_history_text.delete(1.0, tk.END)`：清空消息歷史記錄文本框中的所有內容，以便準備顯示新的消息記錄。使用循環遍歷 `current_user_message_history` 列表中的每條消息，並將其插入到消息歷史記錄文本框中。每條消息都以新的行結束（`\n`），以確保它們在文本框中按行顯示。
- `message_history_text.config(state="disabled")`：將消息歷史記錄文本框的狀態設置為 "disabled"，以防止用戶修改或編輯消息歷史記錄。這樣，用戶只能查看歷史消息，而不能編輯它們。

8.

```

root = tk.Tk()
root.title("110403516 Socket Programming")

frame = tk.Frame(root)
frame.pack(pady=50)

message_label = tk.Label(frame, text="Enter a message:")
message_label.grid(row=0, column=0, padx=10)

message_entry = tk.Entry(frame, width=30)
message_entry.grid(row=0, column=1, padx=10)

send_message_button = tk.Button(
    frame, text="Send Message", command=send_message, state="disabled"
)
send_message_button.grid(row=0, column=2, padx=10)

start_server_button = tk.Button(
    frame, text="Start Server", command=lambda: threading.Thread(target=server).start()
)
start_server_button.grid(row=1, column=0, padx=10)

login_button = tk.Button(frame, text="Login", command=login)
login_button.grid(row=1, column=1, padx=10)

# Create a multi-line text widget for message history
message_history_text = tk.Text(frame, state="disabled", wrap=tk.WORD)
message_history_text.grid(row=2, columnspan=3, padx=10, pady=10, sticky="nsew")
scrollbar = tk.Scrollbar(frame, command=message_history_text.yview)
scrollbar.grid(row=2, column=3, sticky="ns")
message_history_text.config(yscrollcommand=scrollbar.set)

root.mainloop()

```

- `root = tk.Tk()`：創建一個 Tkinter 視窗的根（root）部件，這是 GUI 應用程式的主視窗。
- `root.title("110403516 Socket Programming")`：設置視窗的標題為 "110403516 Socket Programming"。
- `frame = tk.Frame(root)`：創建一個 Frame 部件，用於放置其他 GUI 部件。
- `frame.pack(pady=50)`：將 Frame 部件放置在主視窗中，並設置垂直填充（pady=50）以在頂部添加一些空白空間。
- `message_label = tk.Label(frame, text="Enter a message:")`：創建一個標籤部件，顯示文本 "Enter a message:"。
- `message_label.grid(row=0, column=0, padx=10)`：將標籤部件放置在 Frame 中的第一行第一列，並設置水平填充（padx=10）以添加一些水平間距。
- `message_entry = tk.Entry(frame, width=30)`：創建一個文本輸入框部件，用於輸入聊天消息，設置寬度為 30 個字元。
- `message_entry.grid(row=0, column=1, padx=10)`：將文本輸入框部件放置在 Frame 中的第一行第二列，並設置水平填充（padx=10）以添加一些水平間距。
- `send_message_button = tk.Button(frame, text="Send Message",`

`command=send_message, state="disabled")`：創建一個按鈕部件，顯示文本 "Send Message"，並將其與 `send_message` 函數關聯。此按鈕初始狀態為禁用 (`state="disabled"`)。

- `send_message_button.grid(row=0, column=2, padx=10)`：將按鈕部件放置在 Frame 中的第一行第三列，並設置水平填充 (`padx=10`) 以添加一些水平間距。
 - `start_server_button = tk.Button(frame, text="Start Server", command=lambda: threading.Thread(target=server).start())`：創建一個按鈕部件，顯示文本 "Start Server"，並將其與啟動服務器的函數 `server` 關聯。通過 `threading.Thread` 創建一個新線程來啟動服務器。
 - `start_server_button.grid(row=1, column=0, padx=10)`：將 "Start Server" 按鈕部件放置在 Frame 中的第二行第一列，並設置水平填充 (`padx=10`) 以添加一些水平間距。
 - `login_button = tk.Button(frame, text="Login", command=login)`：創建一個按鈕部件，顯示文本 "Login"，並將其與登錄函數 `login` 關聯。
 - `login_button.grid(row=1, column=1, padx=10)`：將 "Login" 按鈕部件放置在 Frame 中的第二行第二列，並設置水平填充 (`padx=10`) 以添加一些水平間距。
 - `message_history_text = tk.Text(frame, state="disabled", wrap=tk.WORD)`：創建一個多行文本框部件，用於顯示消息歷史記錄。初始狀態為禁用 (`state="disabled"`)，以防止用戶編輯文本。
 - `message_history_text.grid(row=2, columnspan=3, padx=10, pady=10, sticky="nsew")`：將消息歷史記錄文本框放置在 Frame 中的第三行，並跨足 3 列，設置水平和垂直填充，以填充可用空間。
 - `scrollbar = tk.Scrollbar(frame, command=message_history_text.yview)`：創建一個滾動條部件，並將其與消息歷史記錄文本框的垂直滾動聯動。
 - `scrollbar.grid(row=2, column=3, sticky="ns")`：將滾動條部件放置在 Frame 中的第三行第四列，並設置垂直粘性以保持滾動條與文本框對齊。
 - `message_history_text.config(yscrollcommand=scrollbar.set)`：將消息歷史記錄文本框的垂直滾動命令與滾動條的設置關聯，以使滾動條可以滾動文本。
- `root.mainloop()`：啟動 Tkinter 的主事件循環，使 GUI 視窗顯示並響應用戶交互。