Socket Programming 作業

110403518 林晉宇

一、摘要

在本次作業中，我使用 Python 實現了一個多人聊天室。基於 TCP 協議，結合了 multithreading 和 Non-blocking I/O（在 client 部分）的技術來實現高效的傳輸和處理。此外，我還為這個聊天室應用設計 GUI，提供更直觀和用戶友好的交互體驗。在報告的接下來部分，我將分別詳細介紹各個部分的實現方式。首先，我將講述如何利用 Python 的 socket 模組搭建基於 TCP 的 server 和 client，並探討如何通過 multithreading 在服務器端同時處理來自多個 client 的請求。接著，我將說明在客戶端如何實現 non-blocking I/O 來提高應用的響應速度和效率。最後，我將展示如何使用 Tkinter 庫構建聊天室的圖形用戶界面，並解釋 GUI 元件是如何與後端 socket 通信進行交互的。

程式架構是按照物件導向的方向去實作的，server 跟 client 都有各自的 class，所有 function 接包含在 class 中。

二、TCP

1. **Server 建立 tcp 連接:**

```
def __init__(self, host, port):
        self.host = host
        self.port = port
        self.clients = []
        self.server_socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        self.server_socket.setsockopt(socket.SOL_SOCKET,
socket.SO_REUSEADDR, 1)
        self.server_socket.bind((host, port))
        self.server_socket.listen()
```

- 使用 socket.socket(socket.AF_INET, socket.SOCK_STREAM)創建了一個連接。AF_INET 指定了地址族為 IPv4，而 SOCK_STREAM 指明使用了 TCP 協議。
- 通過 self.server_socket.bind((host, port))綁定到指定的主機和端口。
- 調用 self.server_socket.listen()開始監聽來自客戶端的連接請求。

2. **Client 建立 tcp 連接:**

```
def __init__(self, host, port, userid):
        self.host = host
        self.port = port
```

```
        self.userid = userid
        self.socket = socket.socket(socket.AF_INET,
socket.SOCK_STREAM)
        self.socket.connect((host, port))
        self.socket.send(f"{userid} 加入了聊天室".encode())
        self.socket.setblocking(False)
```

- 同樣使用 socket.socket(socket.AF_INET, socket.SOCK_STREAM)創建 TCP 連接。
- self.socket.connect((host, port))與伺服器建立 TCP 連接。
- 使用 self.socket.send(message.encode())向伺服器發送消息。
- 通過 self.socket.setblocking(False)，client 被設置為非阻塞模式。在這種模式下，操作不會阻塞流程，而是在無法立即完成時立即返回。

三、 Multithreading / Non-blocking

我在 server 端使用 multithreading，client 端使用 non-blocking socket；server 端因為要同時處理來自多個 client 端的連接，所以使用 multithreading 可以為每個客戶分配一個獨立的 thread，從而使伺服器能夠同時處理多個客戶端請求，而不會互相干擾；client 端因為包含 GUI，為了保持介面的響應性，所以使用 non-blocking socket 以確保即使在等待網路數據時，仍然可以響應用戶的操作，而在 client 端，通常不用處理來自多個來源的連接，因此無須創建額外的 thread。

1. **Server Multithreading:**

```
def run(self):
        print(f"Server listening on {self.host}:{self.port}")
        while True:
            client_socket, client_address
             = self.server_socket.accept()
            self.clients.append(client_socket)
            client_thread =
threading.Thread(target=self.client_handler,args=(client_socket,))

client_thread.start()
```

- 每當有新的客戶端連接到伺服器（即每當 self.server_socket.accept()方法返回一個新的連接時），伺服器為這個連接創建一個新的 thread。
- 通過 threading.Thread 類來實現的，將 client_handler 函數作為目標函數（target function），並將新連接的客戶端作為參數傳遞給它。
- 在 client_handler 函數中，伺服器讀取來自該 client 端的數據並作出回應。由於每個 client 連接都在自己的 thread 運行，伺服器可以同時處理

來自多個 client 的請求。

- 調用 start()方法來開始新的 thread。這樣，每個 client 的處理代碼都在獨立的 thread 運行。

2. **Client Non-blocking:**

```
def __init__(self, host, port, userid):
        ……
        self.socket.setblocking(False)
```

- 通過 self.socket.setblocking(False)，client 被設置為非阻塞模式。在這種模式下，操作不會阻塞流程，而是在無法立即完成時立即返回。

```
def receive_messages(self):
        while True:
            read_sockets, _, _ = select.select([self.socket], [], [], 
0.1) # non blocking
            for sock in read_sockets:
                try:
                    message = sock.recv(1024).decode()
                    if message:
                        self.update_chat_window(message)
                except Exception as e:
                    print(f"Error: {e}")
                    sys.exit()
```

- 使用 select.select 方法來監視是否有數據可讀。這允許客戶端在數據可供讀取時進行讀取，而在沒有數據時繼續執行其他任務（如處理用戶界面事件）。

四、GUI

GUI 部分我使用 python 的 Tkinter 模組實作。

1. **setup_gui (class ChatClient):**

```
def setup_gui(self):
        self.window = tk.Tk()
        self.window.title(f"聊天室 - {self.userid}")

        # 右上角顯示在線人數
        self.online_count_label = tk.Label(self.window, text="在線人
數: 0")
        self.online_count_label.pack(side=tk.TOP, anchor='ne', 
padx=5, pady=5)
```

```
        # 聊天訊息顯示區域
        self.chat_text = tk.Text(self.window, state='disabled',
height=15, width=50)
        self.chat_text.pack(padx=5, pady=5, expand=True,
fill=tk.BOTH)

        # 輸入框
        self.msg_entry = tk.Entry(self.window)
        self.msg_entry.pack(side=tk.LEFT, padx=5, pady=5, fill=tk.X,
expand=True)
        self.msg_entry.bind("<Return>", self.on_enter_pressed)

        # 傳送按鈕
        self.send_button = tk.Button(self.window, text="傳送",
command=self.on_send_pressed)
        self.send_button.pack(side=tk.RIGHT, padx=5, pady=5)

        self.window.protocol("WM_DELETE_WINDOW", self.on_close)
```

- 使用 tk.Tk()創建了一個 Tkinter 窗口。
- 通過 self.window.title(f"聊天室 - {self.userid}")設置了窗口的標題，其中包括用戶 ID。
- 使用 tk.Label 創建了一個標籤來顯示當前在線人數。這個標籤被放置在窗口的右上角。
- 通過 tk.Text 創建了一個文本區域用於顯示聊天歷史。這個文本區域被設置為禁用（state='disabled'），以便僅用於顯示而不允許用戶直接編輯。
- 使用 tk.Entry 創建了一個輸入框，讓用戶輸入消息。當用戶按下回車鍵時，與之關聯的 on_enter_pressed 方法會被觸發，發送消息。
- 一個 tk.Button 按鈕被添加到窗口中，用戶可以點擊它來發送消息。點擊按鈕會調用 on_send_pressed 方法。
- 通過 self.window.protocol("WM_DELETE_WINDOW", self.on_close)，應用設置了一個事件處理器，當用戶嘗試關閉窗口時，會調用 on_close 方法。
- 應用在一個單獨的線程中運行 receive_messages 方法來接收從伺服器發來的消息。當接收到消息時，應用會更新聊天歷史區域，顯示新消息。
- 最後，通過調用 self.window.mainloop()開始了 Tkinter 的主事件循環，這保持了窗口的開啟並響應用戶操作。

2. **on_close (關閉視窗的 function):**

```
def on_close(self):
        # 當窗口關閉時發送離開消息
        try:
            self.socket.send(f"{self.userid} 離開了聊天室".encode())
            self.socket.close()
        except Exception as e:
            print(f"Error sending leave message: {e}")
        finally:
            self.window.destroy()  # 關閉窗口
```

- 當使用者點擊關閉視窗按鈕時，client 傳送”使用者離開聊天室”訊息到 server 端。
- 使用 self.destroy()關閉視窗。

五、聊天功能相關 function

1. **Server 端**

```
def client_handler(self, client_socket):
        userid = None
        try:
            userid_message = client_socket.recv(1024).decode()
            if userid_message:
                userid = userid_message.split(" ")[0]  # 假設消息格式是 "userid 加入了聊天室"
                print(f"{userid_message} - Connected")
                self.broadcast(userid_message, client_socket)
                time.sleep(0.001)
                self.update_online_count()

            while True:
                message = client_socket.recv(1024).decode()
                if message:
                    self.broadcast(message, client_socket)
                    if message == f"{userid} 離開了聊天室":
                        break
        except Exception as e:
            print(f"Error: {e}")
        finally:
            if userid:
                leave_message = f"{userid} 離開了聊天室"
```

```
                print(leave_message)  # 打印離開消息
            client_socket.close()
            if client_socket in self.clients:
                self.clients.remove(client_socket)
            time.sleep(0.001)
            self.update_online_count()
```

- 函數首先嘗試從客戶端接收數據。
- 從這條消息中提取 userid，並在 server print 連接通知。
- 使用 broadcast 方法將加入聊天室的消息廣播給所有其他客戶端。
- 調用 update_online_count 方法更新並廣播當前在線人數。
- 進入一個無限循環，不斷從客戶端接收消息。
- 每當接收到新消息，就使用 broadcast 方法將其廣播給其他客戶端。
- 如果接收到的消息是用戶離開聊天室的通知（格式為"userid 離開了聊天室"），則跳出循環。
- 如果在接收或處理消息過程中發生任何異常，則會捕獲並打印錯誤信息。(error handling)
- 最終，當使用者離開或是發生異常，會讓使用者斷線，print 使用者離開的通知，更新在線人數。

```
def broadcast(self, message, source_socket):
        for client_socket in self.clients:
            if client_socket != source_socket:
                try:
                    client_socket.send(message.encode())
                except Exception as e:
                    print(f"Error: {e}")
                    client_socket.close()
                    self.clients.remove(client_socket)
```

- 遍歷 self.clients 列表，這個列表包含了所有當前連接的客戶端。
- 對於每個客戶端，首先檢查它是否是發送消息的源頭（source_socket）。如果是，則跳過，因為發送者不需要收到自己發送的消息。
- 對於非發送源的客戶端，函數嘗試將消息發送給它。消息首先被編碼成字節串（message.encode()），以便通過網絡傳輸。
- 如果在發送過程中出現任何異常（如網絡問題或客戶端已斷開連接），則捕獲該異常並打印錯誤信息。
- 如果在發送過程中出現任何異常（如網絡問題或客戶端已斷開連接），則捕獲該異常並打印錯誤信息。

```
def update_online_count(self):
```

```
        count_message = f"在線人數: {len(self.clients)}"
        for client_socket in self.clients:
            try:
                client_socket.send(count_message.encode())
            except Exception as e:
                print(f"Error: {e}")
                client_socket.close()
                self.clients.remove(client_socket)
```

- 函數首先計算 self.clients 列表的長度，即當前連接的客戶端數量。
- 使用 client_socket.send(count_message.encode())向每個客戶端發送 encode 後的在線人數消息。
- 如果在發送過程中遇到任何異常（如網絡問題或客戶端已斷開連接），則會捕獲並打印錯誤信息。

2. **Client 端**

```
def receive_messages(self):
        while True:
            read_sockets, _, _ = select.select([self.socket], [], [],
0.1) # non blocking
            for sock in read_sockets:
                try:
                    message = sock.recv(1024).decode()
                    if message:
                        self.update_chat_window(message)
                except Exception as e:
                    print(f"Error: {e}")
                    sys.exit()
```

- select.select([self.socket], [], [], 0.1)是非阻塞監聽套接字的關鍵部分。這個調用檢查 self.socket 是否有數據可讀，而不會阻塞整個程序。0.1 秒的超時時間意味著即使沒有數據，函數也會在 0.1 秒後返回，讓循環繼續。
- 如果發現有數據可讀（即 read_sockets 非空），則使用 sock.recv(1024).decode()從套接字讀取數據。這裡的 1024 字節是接收緩衝區的大小。
- 接收到的消息被解碼為字符串，然後檢查是否為空。非空消息表示有效的聊天數據，隨後調用 self.update_chat_window(message)將這些消息顯示在 GUI 的聊天歷史區域。

```
def send_message(self):
        message = self.msg_entry.get()
```

```
        if message:
            self.socket.send(f"{self.userid}: {message}".encode())
            self.update_chat_window(f"{self.userid}: {message}")
            self.msg_entry.delete(0, tk.END)
```

- 使用 self.msg_entry.get()從消息輸入框（由 tk.Entry 創建）中獲取用戶輸入的文本。
- 使用 self.socket.send(f"{self.userid}: {message}".encode())將消息發送到伺服器。消息被格式化為"{userid}: {message}"的形式，並被編碼為字節串以進行網絡傳輸。
- 調用 self.update_chat_window(f"{self.userid}: {message}")將發送的消息添加到聊天歷史區域。這確保了用戶能夠在聊天窗口中看到自己發送的消息。
- 通過 self.msg_entry.delete(0, tk.END)清空消息輸入框，為輸入下一條消息做準備。

```
def update_chat_window(self, message):
        if message.startswith("在線人數:"):
            # 更新在線人數標籤
            self.online_count_label.config(text=message)
        else:
            self.chat_text.config(state='normal')
            self.chat_text.insert(tk.END, message + '\n')
            self.chat_text.config(state='disabled')
            self.chat_text.see(tk.END)
```

- 函數首先檢查消息是否以特定的字符串（"在線人數:"）開頭，以此判斷消息是否是在線人數更新。
- 如果消息是在線人數更新，則使用 self.online_count_label.config(text=message)更新在線人數。這讓用戶可以看到最新的在線人數。
- 如果消息是普通聊天消息，則執行以下步驟來更新聊天歷史
- 將新消息插入到聊天文本區域的末尾（self.chat_text.insert(tk.END, message + '\n')）。
- 禁用文本框（self.chat_text.config(state='disabled')）來防止用戶直接編輯聊天歷史。
- 自動滾動到文本區域的底部，以顯示最新消息（self.chat_text.see(tk.END)）。

六、總結

使用 python 實作 socket programming 作業，作出使用 tcp 連接的簡易多人

聊天室；server 端使用 multithreading 處理多用戶連接的問題，client 端使用 non-blocking socket 達成 gui 即時響應的效果。