Messenger Application

Socket Programming

این Application برای انتقال پیام از یک Client به Client دیگر با اتصال به یک Server و استفاده از پروتکل Socket طراحی شده و دارای GUI می‌باشد.

همچنین در این Application از Multi Threading برای مدیریت همزمان محیط گرافیکی Clientها نیز استفاده شده.

نمادهایی که در این Document استفاده شده:

Function(inputs)

[FileName, LineNumber]

فهرست

توضیحات و نحوه عملکرد Server

این بخش، هسته‌ی برنامه است و وظیفه‌ی دریافت پیام و انتقال آن به Clientهای دیگر را دارد.

این بخش روی localhost و پورت ۹۰۹۰ اجرا می‌شود.

```
# AF_INET ==> internet socket
# SOCK_STREAM ==> TCP socket
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind((HOST, PORT))

server.listen()

clients = []
nickNames = []
```

Socket

در ابتدا یک سرور از نوع TCP Socket همچنین لیستی از Clientها و Nicknameها ایجاد شده.این سرور وظیفه مدیریت ارسال‌ها و دریافت‌ها را دارد.

```
# Broadcast messages
def broadcast(message):
  """
  Sends one message to all the clients
  Arguments: message
  """
  for client in clients:
    client.send(message)
```

broadcast(message) - [server.py, 23]

این تابع، وظیفه‌ی توزیع پیام‌های ورودی از یک Client به دیگر Clientها را دارد. ورودی این تابع message است و خروجی ندارد.

```
# Handle
def handle(client):
  """ Handle Connections """
  while True:
    try:
      message = client.recv(1024) # 1024 Bytes
      print(f'{nickNames[clients.index(client)]} says {message}')
      broadcast(message) # Broadcast to all users in server
    except:
      # if does not connected
      index = clients.index(client)
      clients.remove(client)
      client.close()

      nickname = nickNames[index]
      nickNames.remove(nickname)
      break
```

handle(client) - [server.py, 23]

این تابع، عملیات اتصال Clientها را مدیریت می‌کند. یک ورودی client نیز دارد.

به این صورت که:

۱. اگر پیام ورودی از یک Client ثبت شده یا شناخته شده باشد؛ آن پیام را با تابع ()recv که در سمت Client نوشته شده، دریافت کرده و سپس با استفاده از تابع ()broadcast که در سمت Server نوشته شده، به بقیه Clientها ارسال می‌کند.

۲. در غیر این صورت، اگر در ارسال پیام یا دریافت پیام، مشکلی رخ دهد؛ به این معنی‌ست که Client، معتبر نیست و باید حذف شود که در قسمت exception این عملیات حذف انجام میگیرد.

receive() - [server.py, 56]

```python
# Receive messages
def receive():
  """ Accept new client connections """

  while True:
    # Accept connection
    client,  address = server.accept()
    print(f'Connected with {str(address)}')

    # Give it a nickname
    client.send('NIC'.encode('utf-8'))
    nickname = client.recv(2048)

    # Add to database
    clients.append(client)
    nickNames.append(str(nickname))

    print(f'Nickname of the client is {nickname}') # server-side message
    broadcast(f'{nickname} joined the chat 🤩\n'.encode('utf-8'))
    client.send('Connected to the server.'.encode('utf-8')) # Just for this client send

    thread = threading.Thread(target=handle, args=(client,))
    thread.start()
```

این تابع، وظیفه‌ی ایجاد Client در سمت Server در صورت دریافت درخواست از سمت Client را دارد.

ابتدا با تابع accept() منتظر یک Client است و پس از دریافت اولین درخواست، آن را به لیست Clientها اضافه میکند. همچنین از Client درخواست وارد کردن یک nickname برای استفاده از آن در قسمت UI می‌کند.

در نهایت، به Clientهای دیگر پیغامی مبنی بر وارد شدن این Client می‌دهد و همچنین این Client را به Threadها اضافه می‌کند.

توضیحات و نحوه عملکرد Client

در این قسمت یک کلاس به نام Client داریم که در initialize کردن آن، دو پارامتر ورودی HOST و PORT مورد نیاز است.

__init__(self, host, port) - [client.py, 14]

```
def __init__(self, host, port):
  """ Initialization an instance """
  self.socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
  self.socket.connect((host, port))

  msg = tkinter.Tk() # Main window
  msg.withdraw()

  self.nickname = simpledialog.askstring('Nickname', 'Please choose a nickname:', parent=msg)

  self.gui_done = False
  self.running = True # Default is True and means app is running

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

  gui_thread.start()
  receive_thread.start()
```

در این قسمت نیز مانند Server، ابتدا یک Socket ایجاد کرده و با استفاده از تابع connect((host, port)) و پاس دادن HOST و PORT به آن، به سرور متصل می‌شویم.

سپس یک GUI با استفاده از پکیج tkinter ایجاد کرده و یک پنجره برای دریافت nickname از کاربر ایجاد میکنیم. پس از دریافت nickname شروع به ساخت GUI در یک Thread می‌کنیم و همچنین یک Thread هم برای دریافت پیام‌های سرور ایجاد کرده و هردو را اجرا می‌کنیم.

gui_loop(self) - [client.py, 33]

```
def gui_loop(self):
  """ GUI builder """

  # Main Window
  self.window = tkinter.Tk()
  self.window.configure(bg='lightgrey')

  # Chat Area Label
  self.chat_label = tkinter.Label(self.window, text='Chat: ', bg='lightgrey')
  self.chat_label.config(font=('Calibri', 12)) # config = configure
  self.chat_label.pack(padx=20, pady=5) # Adding some paddings

  # Chat Area
  self.text_area = tkinter.scrolledtext.ScrolledText(self.window)
  self.text_area.pack(padx=20, pady=5)
  self.text_area.config(state='disabled') # config = configure # Can't change input


  # Message Label
  self.message_label = tkinter.Label(self.window, text='Message: ', bg='lightgrey')
  self.chat_label.config(font=('Calibri', 12))
  self.chat_label.pack(padx=20, pady=5)

  # Message Area
  self.input_area = tkinter.Text(self.window, height=3)
  self.input_area.pack(padx=20, pady=5)
```

این تابع در یک Thread جداگانه پردازش می‌شود و تداخلی با بقیه توابع ندارد.

در این قسمت، UI ساخته می‌شود.

به این صورت که یک Object از tkinkter.TK ساخته می‌شود و بقیه موارد به آن اضافه می‌شود.

نحوه اضافه کردن هر اِلمان به این صورت است که ابتدا در یک متغیر، یک Widget از tkinter تعریف کرده و در تابع سازنده‌ی آن سپس با تابع config()، آن را configure کرده و با تابع pack() کنار بقیه اِلمان‌ها قرار میدهیم.

در ادامه‌ی تابع بالایی، یک دکمه نیز برای ارسال پیام قرار می‌دهیم.

```
# Button
self.send_button = tkinter.Button(self.window, text='Send', command=self.write)
self.send_button.config(font=('Calibri', 12))
self.send_button.pack(padx=20, pady=5)

# Send signal to other functions that UI is up now
self.gui_done = True

# if window is closed
self.window.protocol('WM_DELETE_WINDOW', self.stop)

self.window.mainloop()
```

همچنین برای توقف GUI از تابع protocol() استفاده میکنیم و در صورت دریافت ‘WM_DELETE_WINDOW’، تابع stop از همین کلاس Client را فراخوانی میکنیم.

write (self) - [client.py, 73]

```
def write(self):
  """ Get the text from message box and send it to the server and then clean the message area """

  message = f"{self.nickname}: {self.input_area.get('1.0', 'end')}" # ('1.0', 'end') ==> get the whole text
  self.socket.send(message.encode('utf-8'))
  self.input_area.delete('1.0', 'end')
```

این تابع، برای ارسال پیام از Client به Server استفاده می‌شود. البته باید توجه داشت که ابتدا پیام باید encode شده و سپس به سمت Server ارسال شود. بعد از آن نیز برای نوشتن پیام جدید، قسمت input_area پاک میشود.

stop(self) - [client.py, 82]

این تابع برای خاتمه‌ی GUI استفاده میشود.

مورد استفاده در تابع gui_loop

```
def stop(self):
  """ Stop and Close the Client window """

  self.running = False
  self.window.destroy()
  self.socket.close()
  exit(0)
```

```python
def receive(self):
    """ Handle functions for receiving and showing messages """


    while self.running:
        try:
            message = self.socket.recv(1024).decode('utf-8') # 1024 Bytes

            if message == 'NIC':
                self.socket.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') # Always scroll down to the end with getting messages
                    self.text_area.config(stat='disabled')
        except ConnectionAbortedError:
            break
        except:
            print('ERROR')
            self.socket.close()
            break
```

receive(self) - [client.py, 93]

این تابع در یک Thread جداگانه پردازش می‌شود و تداخلی با بقیه توابع ندارد.

این تابع در قسمت Client، مدیریت نمایش پیام‌های ورودی در قسمت text_area در UI را دارد.

به این صورت که ابتدا GUI منتظر دریافت پیامی از سمت سرور Socket می‌باشد و زمانی که پیام دریافت شد، آن را encode کرده و سپس نمایش می‌دهد.

نحوه نمایش پیام به این صورت است که ابتدا قسمت text_area که به دلیل پیشگیری از وارد کردن متن در این ناحیه، disable(غیر فعال شده بود) را فعلا میکنیم و به حالت normal برمیگردانیم و پیام وارده را با استفاده از تابع insert و ورودی های end(پیام را به آخر این ناحیه وارد می‌کند) و message(پیام ورودی‌ای که encode شده) در این قسمت چاپ میکنیم. و در نهایت؛ قسمت text_area را دوباره به حالت disabled(غیر فعال) برمیگردانیم.

در صورت وجود مشکل اتصال، این عملیات متوقف می‌شود.

همچنین در صورت وجود مشکلات دیگر غیر از اتصال، پیام ERROR در کنسول نمایش داده شده و اتصال قطع میگردد.

نحوه اجرای برنامه

برای اجرای برنامه کافی‌ست در محیط cmd و در دایرکتوری server.py، دستور زیر را اجرا کنید:

```
> python server.py
```

با اجرای این کد، Server اجرا شده و با دریافت پیغام "Server is running..." می‌توانید، Client را اجرا کرده و لذت ببرید... 😄

به دلیل استفاده از Threading، همزمان امکان استفاده از چندین Client است.

برای اجرای Client کافی‌ست در محیط cmd و در دایرکتوری client.py، دستور زیر را اجرا کنید:

```
> python client.py
```

با هربار اجرای این Command، یک Client ایجاد می‌شود.

تصاویر اجرای برنامه

SERVER

```
PS D:\A Uni\4001\Internet.Engineering(HamedRahimi)\Files\Project> python.exe .\server.py
Server is running...
```

CLIENT