LAPORAN TUGAS
II2230 JARINGAN KOMPUTER
Chatting Application Based On
UDP Socket Programming

Dosen:

Hamonangan Situmorang, S.T, M.T.

Disusun Oleh :

Anindita Widya Santoso / 18222128

PROGRAM STUDI SISTEM DAN TEKNOLOGI INFORMASI

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

DAFTAR ISI

DAFTAR GAMBAR

1. Tujuan Dokumen

Tujuan dokumen ini adalah untuk memberikan panduan terperinci tentang pengembangan aplikasi *chat* menggunakan *socket* UDP dengan antarmuka grafis berbasis *library* tkinter di Python. Dokumen ini akan membahas secara mendalam tentang arsitektur dan desain aplikasi, menguraikan bagaimana mekanisme komunikasi *non-blocking* dapat dicapai melalui *threading*, yang memungkinkan operasi I/O paralel tanpa mengganggu responsivitas GUI. Selain itu, akan dijelaskan pula tentang penggunaan *socket* UDP untuk memfasilitasi komunikasi *real-time* yang efisien dan minim latensi. Dengan penjelasan konseptual dan contoh kode, dokumen ini bertujuan untuk memberikan pembaca pemahaman komprehensif tentang pembuatan aplikasi chat interaktif dengan Python.

2. Latar Belakang

Dalam era teknologi yang semakin canggih, aplikasi *chat* memegang peranan penting dalam kehidupan sehari-hari, memungkinkan individu berkomunikasi dengan mudah dan cepat, tanpa terikat oleh batasan geografis. Kemajuan ini menghasilkan sebuah tantangan bagi penulis untuk mengembangkan aplikasi *chat* yang tidak hanya fungsional tetapi juga efisien, memanfaatkan *socket* UDP untuk memastikan pengiriman pesan yang cepat dan handal, sangat sesuai dengan kebutuhan komunikasi zaman sekarang yang menuntut kecepatan dan efisiensi.

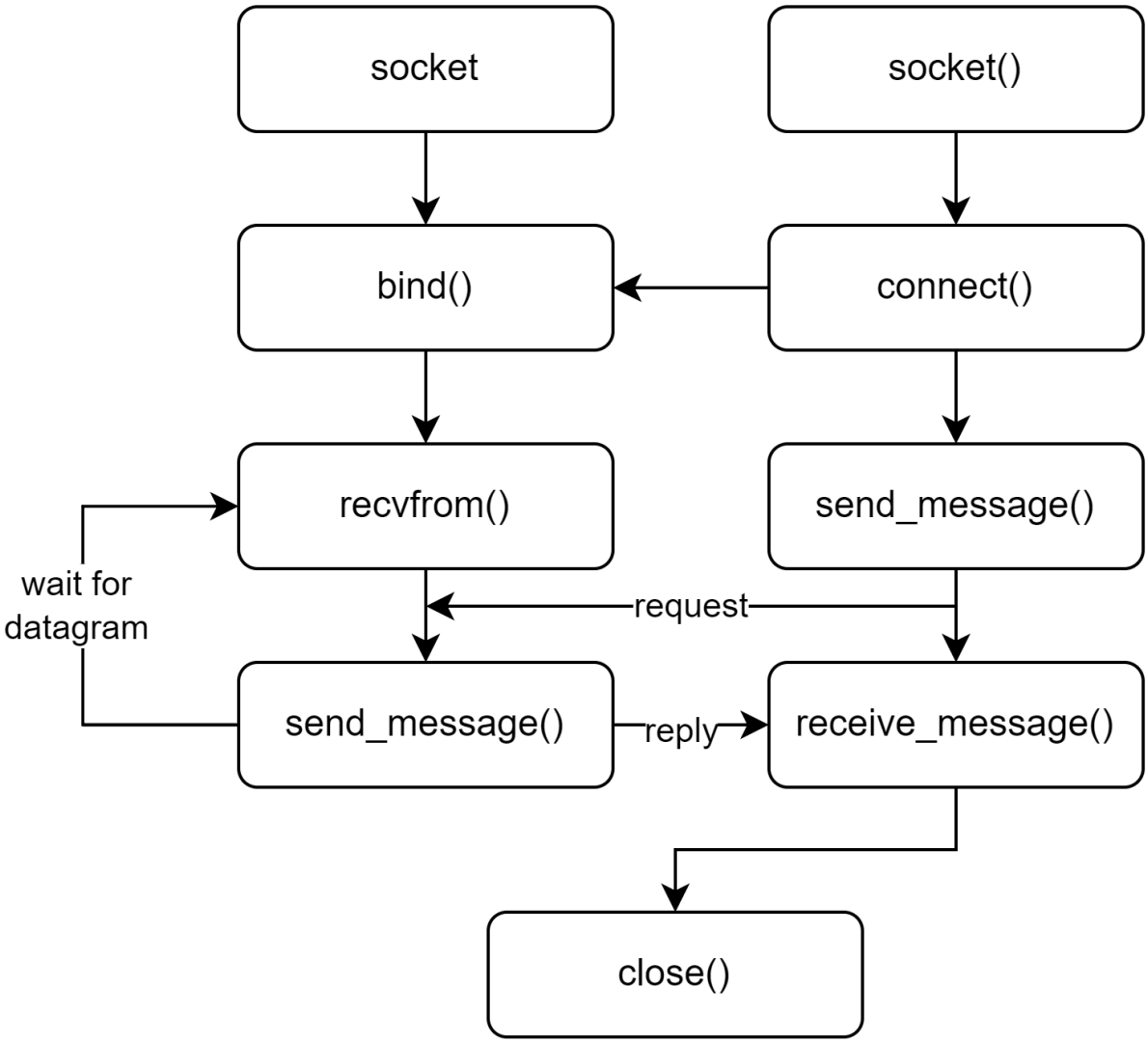
Memilih UDP sebagai pondasi aplikasi ini bukan tanpa alasan karena protokol ini menawarkan mekanisme yang ringan dan minim *overhead*, ideal untuk aplikasi chat yang ditujukan untuk komunikasi *real-time*. Dengan UDP, aplikasi dapat mengirim dan menerima pesan secara serempak tanpa perlu menunggu akumulasi pesan, memberikan responsivitas yang unggul dibandingkan protokol yang berorientasi koneksi. Namun, UDP merupakan sebuah protokol yang tidak menjamin integritas data, sehingga ada kemungkinan adanya ketidaksesuaian antara pesan yang dikirim dan yang diterima.

Python, dengan sintaksnya yang jelas dan *library* tkinter yang kaya fitur, dipilih sebagai bahasa pemrograman utama dalam proyek ini. Pilihan ini tidak hanya memudahkan proses pengembangan tetapi juga memungkinkan penyesuaian dan pemeliharaan yang lebih mudah di masa depan. Tkinter, dengan kemampuannya dalam membangun GUI yang intuitif, memastikan bahwa aplikasi ini dapat digunakan dengan nyaman oleh pengguna, tanpa memerlukan pengalaman teknis khusus.

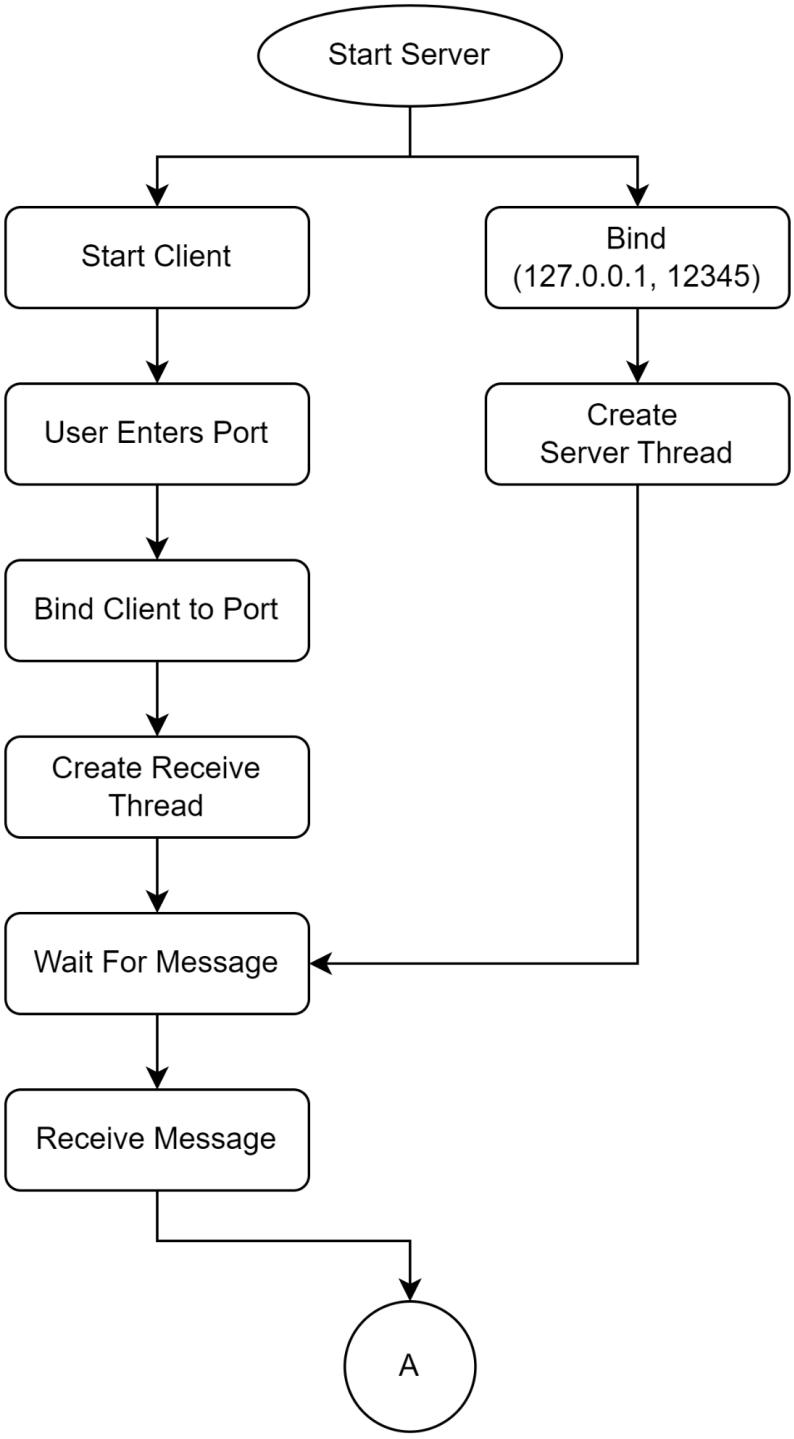
Proyek ini tidak hanya menjadi sarana untuk memenuhi tugas akademik tetapi juga sebagai langkah praktis dalam memahami lebih lanjut tentang jaringan komputer dan pemrograman aplikasi berbasis jaringan. Melalui pengembangan aplikasi *chat* ini, penulis berharap dapat menawarkan perspektif baru dalam desain dan implementasi sistem komunikasi yang efisien dan efektif.

3. *Flowchart* Program

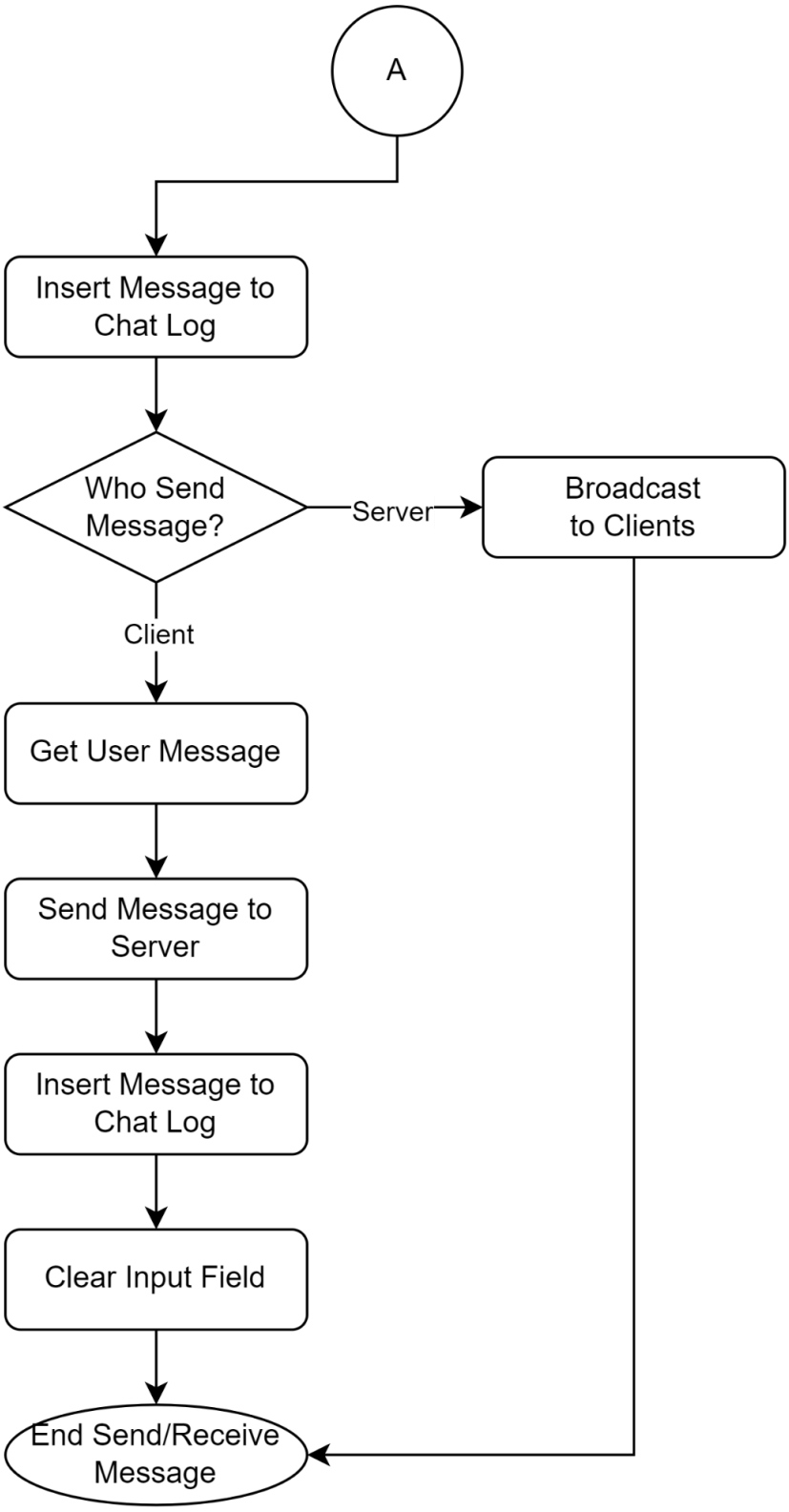
Program *chat* ini menggunakan protokol UDP untuk komunikasi antara *server* dan *client*, dengan antarmuka pengguna grafis yang dibuat menggunakan tkinter. *Server* mendengarkan pesan dari *client* dan mendistribusikannya ke semua *client* yang terhubung, sedangkan *client* dapat mengirim dan menerima pesan. Baik *server* maupun *client* menjalankan operasi jaringan mereka di *thread* terpisah untuk tidak mengganggu antarmuka pengguna, yang menampilkan *log chat* dan menyediakan kotak untuk mengirim pesan. Berikut merupakan *flowchart* dari aplikasi *chat* berbasis *socket* UDP.



Gambar 2.1 *Flowchart General* Kode UDP *Client Server*



Gambar 2.2 *Flowchart* Program Bagian 1



Gambar 2.3 *Flowchart* Program Bagian 2

Berdasarkan analisis terhadap gambar 2.2 dan gambar 2.3, dapat dipahami bahwa proses komunikasi dalam program ini diawali dengan pengaktifan *server*, yang merupakan langkah penting karena *client* tidak dapat beroperasi tanpa adanya *server* yang aktif. Langkah pertama dalam menjalankan program adalah menginisialisasi *server*, yang melibatkan pengikatan (*binding*) *server* ke alamat lokal 127.0.0.1 dan *port* 12345. Ini dilanjutkan dengan pembuatan *thread server* yang berfungsi untuk mendengarkan dan memproses pesan masuk dari *client*.

Di sisi *client*, proses dimulai dengan pengguna memasukkan nomor *port* yang akan digunakan untuk *binding*. Setelah itu, *client* di-*bind* ke *port* tersebut, dan *thread* penerimaan data dibuat untuk mendengarkan pesan dari *server*. *Client* berada dalam mode menunggu hingga pesan diterima dari *server*.

Setelah pesan diterima, baik di sisi *server* maupun *client*, pesan tersebut akan ditambahkan ke *log chat*. *Log* ini merupakan komponen penting yang menampilkan riwayat komunikasi. Jika *server* mengirim pesan, pesan tersebut akan di-*broadcast* ke semua *client* yang terhubung, memastikan bahwa setiap *client* mendapatkan pembaruan informasi secara *real-time*. Sebaliknya, jika *client* yang mengirim pesan, pesan tersebut akan dikirim ke *server*, yang kemudian akan memproses dan mendistribusikannya ke *client* lainnya.

Proses kirim-terima pesan ini merupakan inti dari aplikasi *chat*, di mana pesan yang dikirim dan diterima secara dinamis dikelola dan ditampilkan dalam antarmuka pengguna. Hal ini memungkinkan interaksi yang mulus antara pengguna dalam bentuk percakapan teks. Setiap transaksi data, mulai dari pengiriman hingga penerimaan pesan, dilakukan dengan memastikan integritas dan keakuratan informasi, menggarisbawahi keandalan sistem dalam mengelola komunikasi. Melalui pendekatan ini, aplikasi chat menawarkan platform komunikasi yang efektif dan interaktif bagi para penggunanya.

4. Kode Program

Kode proyek ini dituliskan menggunakan bahasa Python berdasarkan *flowchart* yang sudah dirancang dengan dua bagian utama, yaitu server.py dan client.py. Berikut adalah kode yang digunakan dalam proyek berikut.

Kode Server.py

```
import socket
import threading
from tkinter import *
from tkinter import scrolledtext

def start_server():
    while True:
        message, address = server.recvfrom(1024)
        message = message.decode('utf-8')
```

```
        clients.add(address)
        chat_log.insert(END, f"Client {address}: {message}\n")

def send_message():
    message = entry_message.get()
    chat_log.insert(END, "Server: " + message + "\n")
    for client in clients:
        server.sendto(message.encode('utf-8'), client)
    entry_message.delete(0, END)

server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server.bind(('127.0.0.1', 12345))

clients = set()

window = Tk()
window.title("Server")

chat_log = scrolledtext.ScrolledText(window, width=50, height=10)
chat_log.grid(column=0, row=0, padx=10, pady=10)

entry_message = Entry(window, width=40)
entry_message.grid(column=0, row=1, padx=10, pady=10)

send_button = Button(window, text="Kirim", command=send_message)
send_button.grid(column=0, row=2, padx=10, pady=10)

server_thread = threading.Thread(target=start_server, daemon=True)
server_thread.start()

window.mainloop()
```

Kode dalam *file* Server.py mengimpor *library socket* untuk pembuatan *socket* UDP, *threading* untuk inisialisasi *thread* baru, serta tkinter dan *scrolledtext* untuk pengembangan antarmuka pengguna grafis. Fungsi start_server bertugas mendengarkan pesan masuk, menyimpan alamat *client* dan pesan yang diterima ke dalam *log chat*. Fungsi send_message mengambil pesan dari entry_message dan mengirimkannya ke semua *client* yang telah terkoneksi. Inisialisasi *socket server* dilakukan untuk membuat *socket* UDP, sementara GUI diinisialisasi untuk menyiapkan *window*, dan *multithreading* digunakan untuk menjalankan *thread* baru. Proses mainloop dipanggil untuk menjaga jendela tetap aktif dan responsif terhadap interaksi pengguna.

Kode Client.py

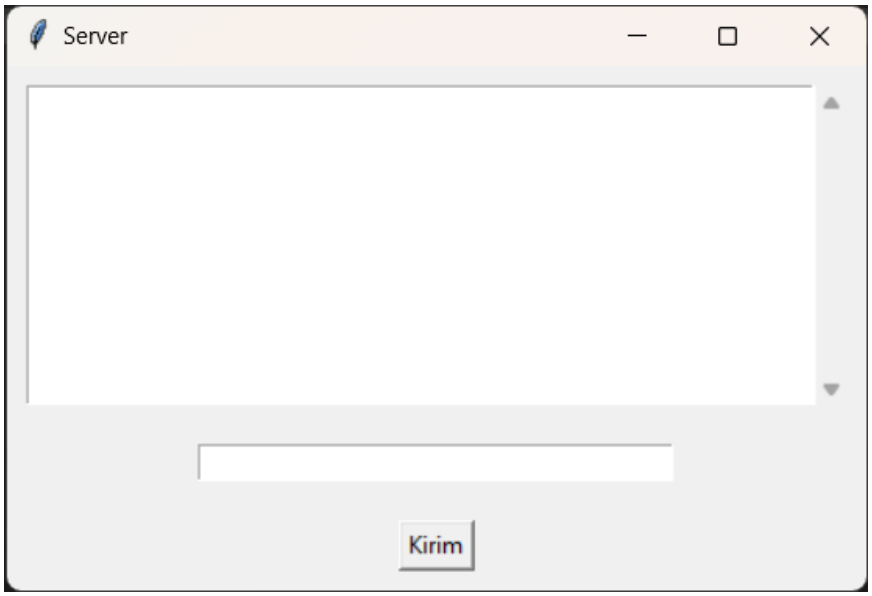
```
import socket
import threading
from tkinter import *
from tkinter import scrolledtext

def receive_message():
    while True:
        try:
            message, _ = client.recvfrom(1024)
            message = message.decode('utf-8')
            chat_log.insert(END, "Server: " + message + "\n")
        except Exception as e:
            print("Error receiving message: ", e)
            break

def send_message():
    message = entry_message.get()
    chat_log.insert(END, "Client: " + message + "\n")
    client.sendto(message.encode('utf-8'), server_address)
    entry_message.delete(0, END)

server_address = ('127.0.0.1', 12345)
client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
client.bind(('127.0.0.1', int(input("Enter port: "))))

window = Tk()
window.title("Client")

chat_log = scrolledtext.ScrolledText(window, width=50, height=10)
chat_log.grid(column=0, row=0, padx=10, pady=10)

entry_message = Entry(window, width=40)
entry_message.grid(column=0, row=1, padx=10, pady=10)

send_button = Button(window, text="Kirim", command=send_message)
send_button.grid(column=0, row=2, padx=10, pady=10)

receive_thread = threading.Thread(target=receive_message)
receive_thread.start()
```

```
window.mainloop()
```

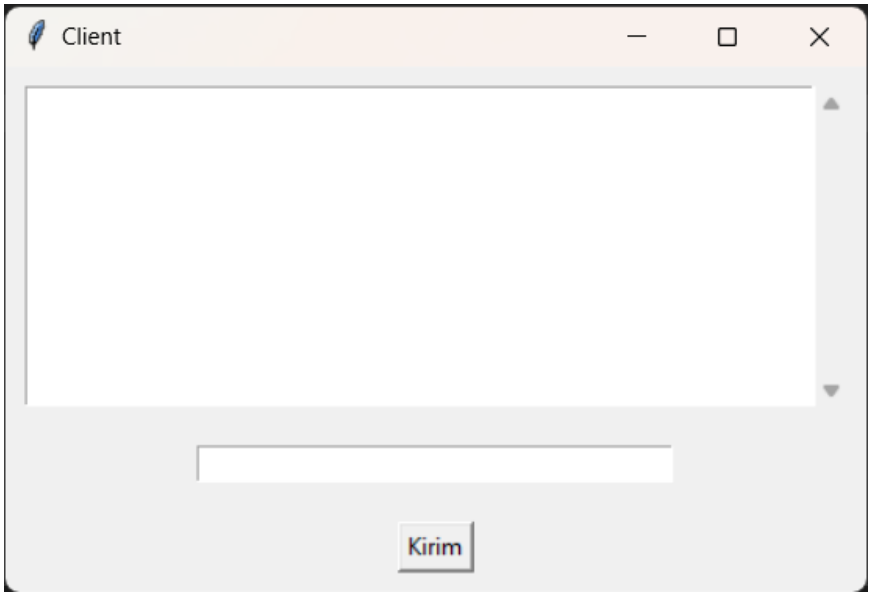
Kode dalam *file* Client.py mengimpor *library socket* untuk pembuatan *socket* UDP, *threading* untuk inisialisasi *thread* baru, serta tkinter dan *scrolledtext* untuk pengembangan antarmuka pengguna grafis. Fungsi receive_message terus menerus mendengarkan dan menerima pesan dari *server*, menampilkan pesan tersebut dalam *log chat.* Fungsi send_message mengambil teks dari entry_message, menampilkannya dalam *log chat,* dan mengirimkannya ke *server*. *Socket* UDP *client* diinisialisasi dan di-*bind* ke alamat IP lokal dan *port* yang ditentukan pengguna. GUI diinisialisasi untuk menampilkan *window*, *log chat*, kotak entri pesan, dan tombol kirim. *Multithreading* digunakan untuk memulai *thread* baru yang menjalankan receive_message, memungkinkan penerimaan pesan tanpa mengganggu interaksi GUI. Proses mainloop dipanggil untuk menjaga jendela tetap aktif dan responsif terhadap interaksi pengguna.

5. GUI Program

Berdasarkan kode yang disertakan, graphical user interface (GUI) untuk aplikasi chat yang dirancang menggunakan Tkinter dalam Python ini dirancang untuk fungsionalitas dan kemudahan penggunaan. Berikut adalah *graphical user interface* (GUI) aplikasi *chat* dalam percobaan ini.



Gambar 5.1 Tampilan Antarmuka Grafis Server.py



Gambar 5.2 Tampilan Antarmuka Grafis Client.py

6. Skenario Percobaan

Untuk memastikan kode percobaan yang dirancang sudah sesuai dengan tujuan, dilakukan perancangan skenario percobaan untuk menguji kode yang sudah ada. Penulis membuat dua buah skenario untuk memastikan kode sudah benar-benar berjalan dengan baik. Skenario percobaan tersebut adalah sebagai berikut.

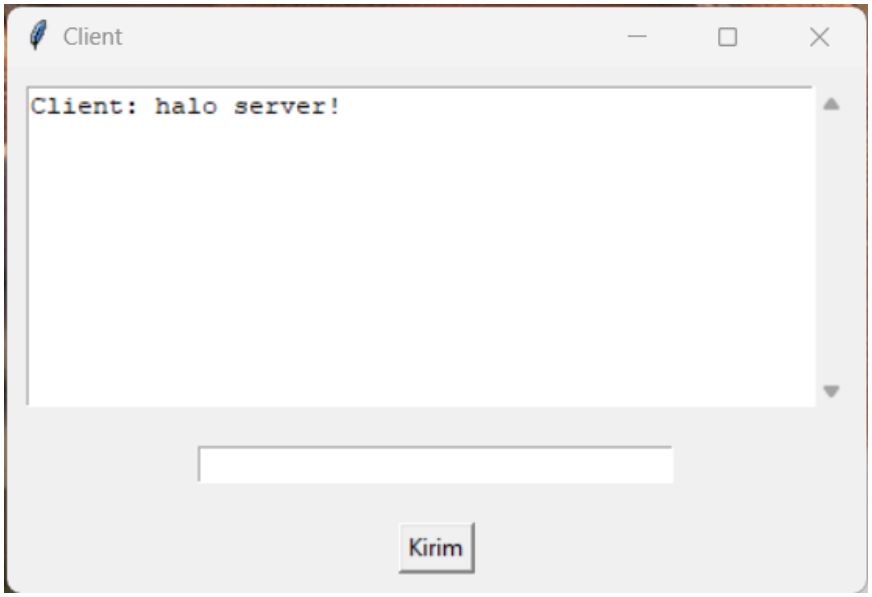
1. **Interaksi *server* dengan satu *client***
    a. *Server* dan *client* dijalankan
    b. *Client* mengirimkan “halo server!” sebagai inisialisasi hubungan komunikasi
    c. *Server* menerima pesan *client* beserta alamat dan *port* yang digunakan oleh *client*
    d. *Server* mengirimkan balasan pesan “halo juga client!” kepada *client*

2. **Interaksi *server* dengan *multiple client***
    a. *Server* dan kedua *client* dijalankan
    b. *Client* 2 mengirimkan “hi” sebagai inisialisasi hubungan komunikasi
    c. *Server* menerima pesan *client* 2 beserta alamat dan *port* yang digunakan oleh *client* 2
    d. *Client* 1 mengirimkan “halo server ^-^” sebagai inisialisasi hubungan komunikasi
    e. *Server* mengirimkan balasan pesan “halo clients!” dalam bentuk *broadcast* kepada semua *client* yang sudah memiliki hubungan komunikasi dengan *server*

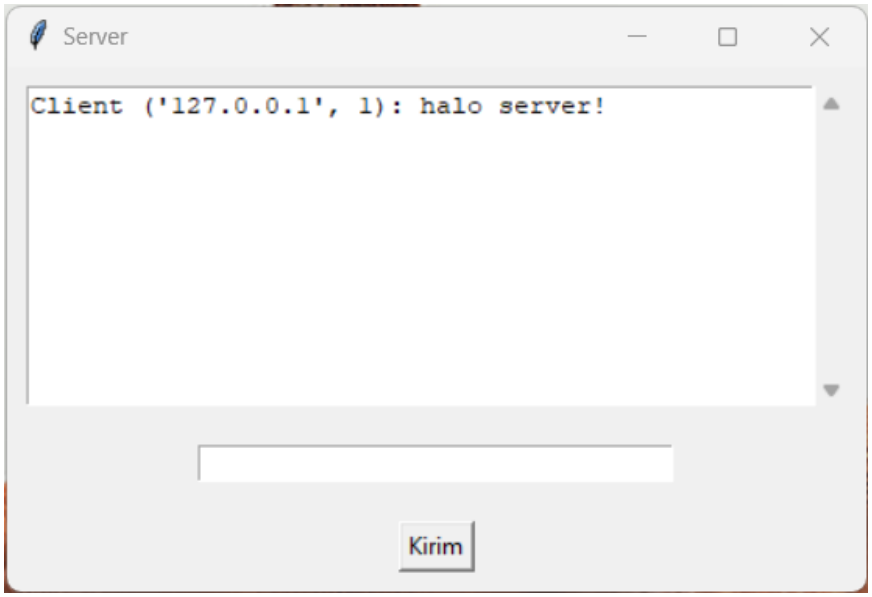
7. Hasil Pengujian

Sesuai dengan skenario percobaan yang telah dirancang pada bagian sebelumnya, didapatkan hasil pengujian sebagai berikut.

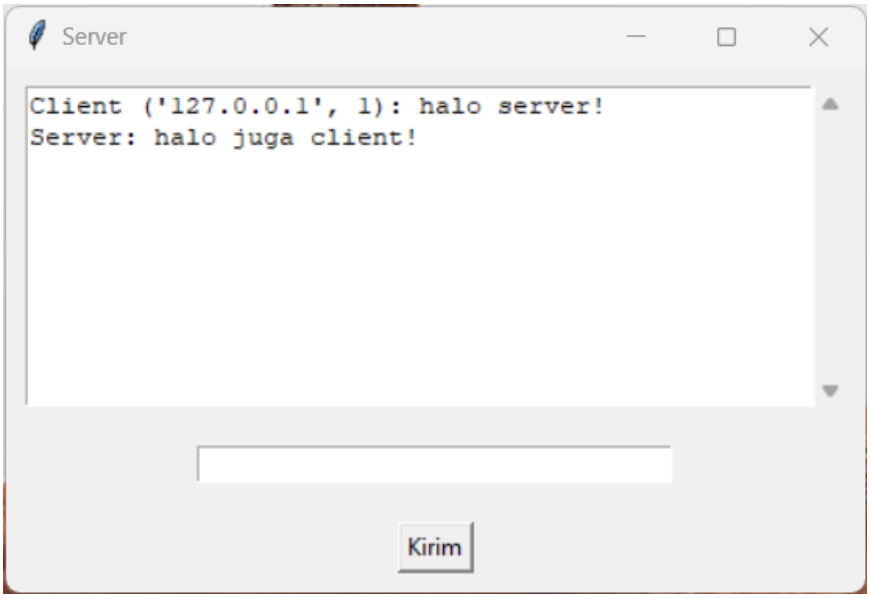
1. **Interaksi *server* dengan satu *client***
   Pada skenario ini, dibuat sebuah hubungan antara sebuah *server* dengan sebuah *client* yang komunikasinya diinisialisasi oleh *client.*



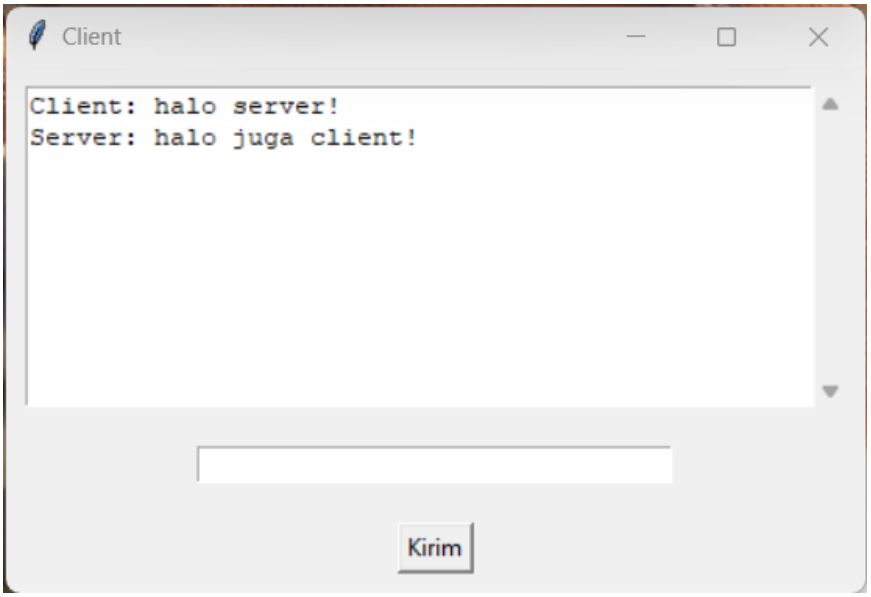
Gambar 7.1 *Client* melakukan inisialisasi hubungan (sudut pandang *client*)



Gambar 7.2 *Client* melakukan inisialisasi hubungan (sudut pandang *server*)



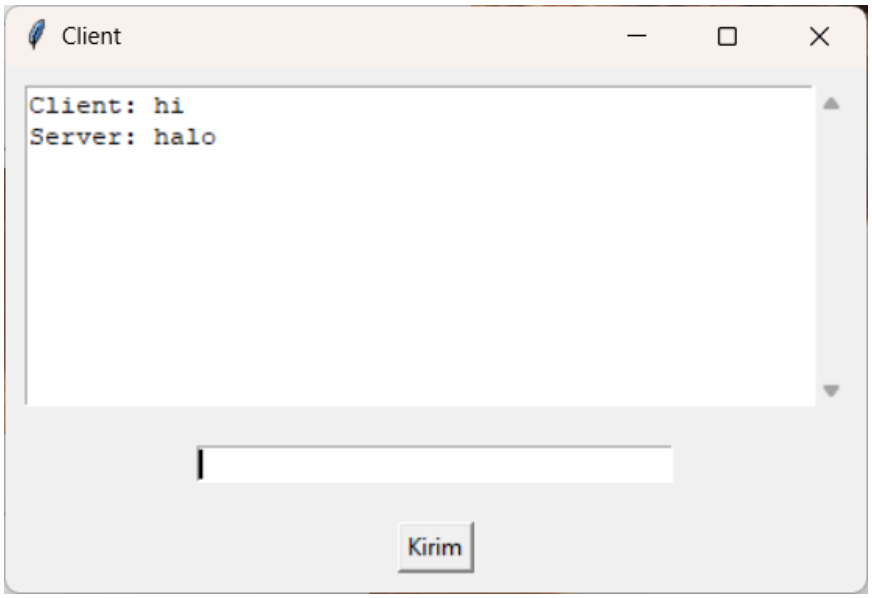
Gambar 7.3 *Server* mengirimkan balasan pada *client* (sudut pandang *server*)



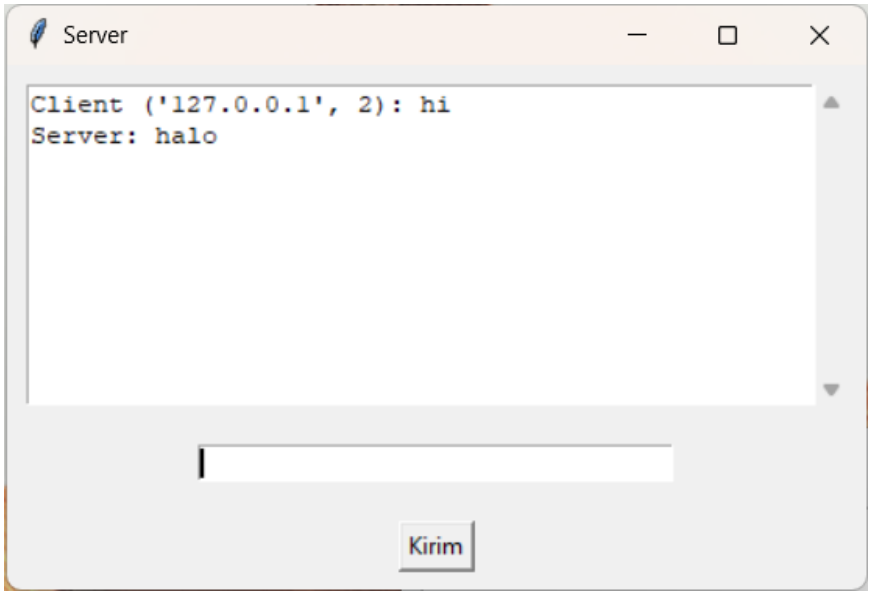
Gambar 7.4 *Server* mengirimkan balasan pada *client* (sudut pandang *client*)

2. **Interaksi *server* dengan *multiple client***

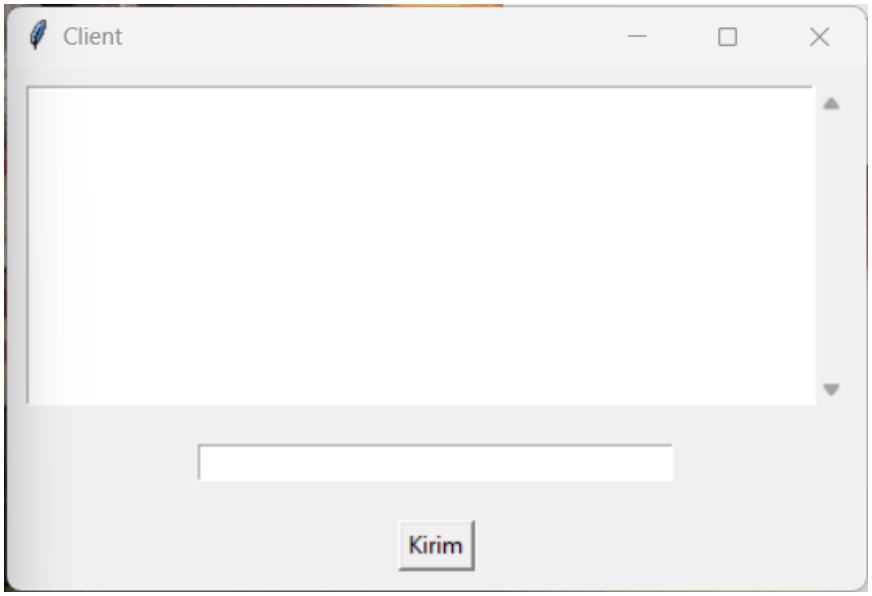
   Berbeda dengan skenario sebelumnya, pada skenario ini, dibuat hubungan antara sebuah *server* dengan beberapa *client* (pada percobaan ini dibuat hubungan dengan dua *client*, seharusnya bisa lebih).



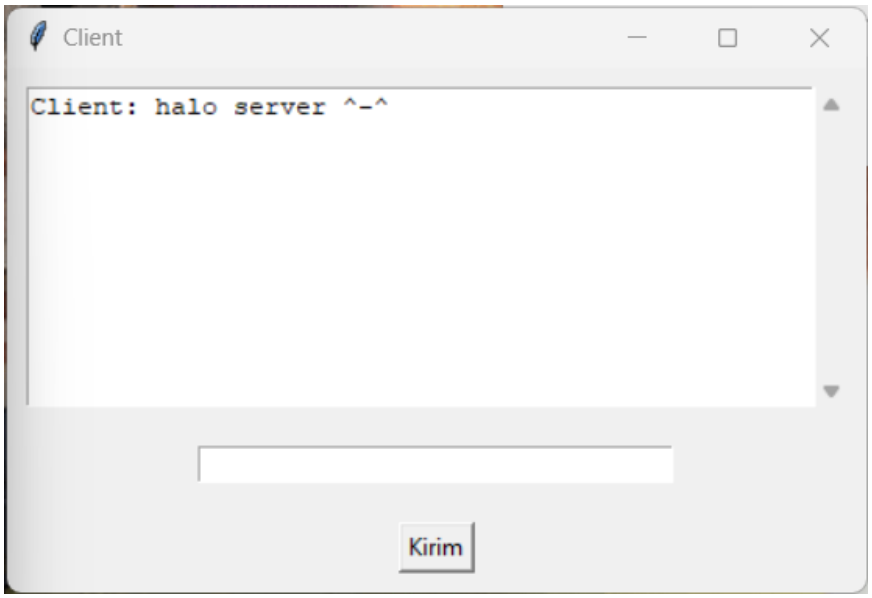
Gambar 7.5 *Client* 2 melakukan inisialisasi hubungan yang dibalas oleh *server* (sudut pandang *client* 2)



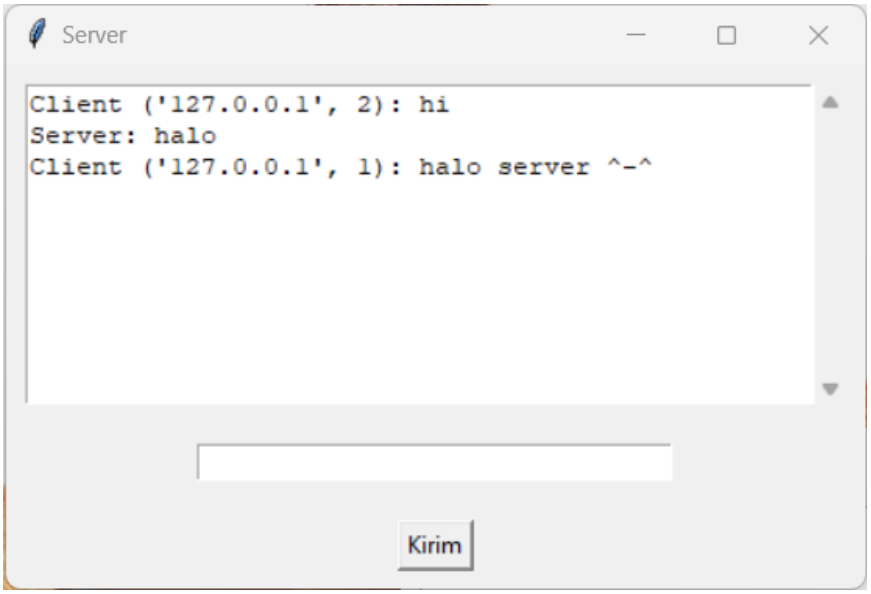
Gambar 7.6 *Client* 2 melakukan inisialisasi hubungan yang dibalas oleh *server* (sudut pandang *server*)



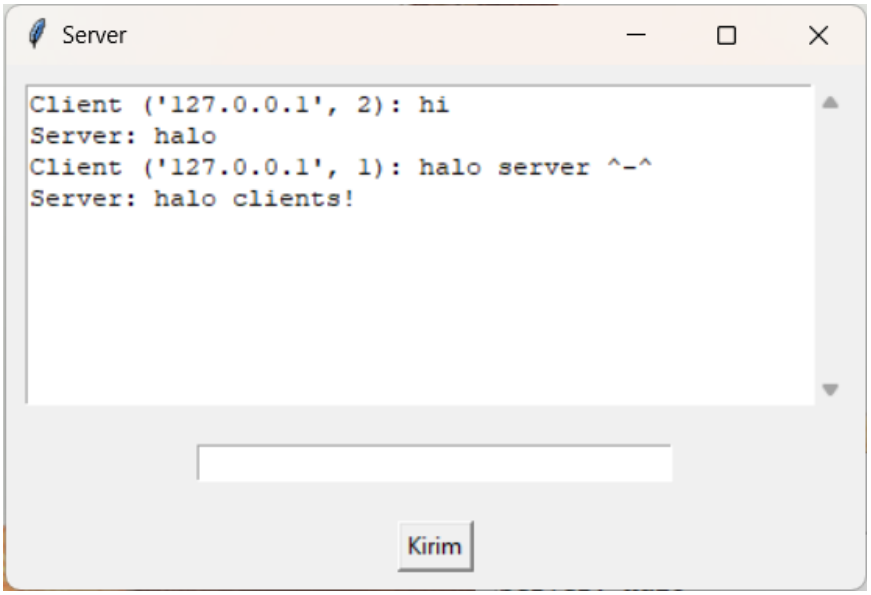
Gambar 7.7 Tampilan yang muncul pada *client* 1 yang belum melakukan inisialisasi hubungan dengan *server*



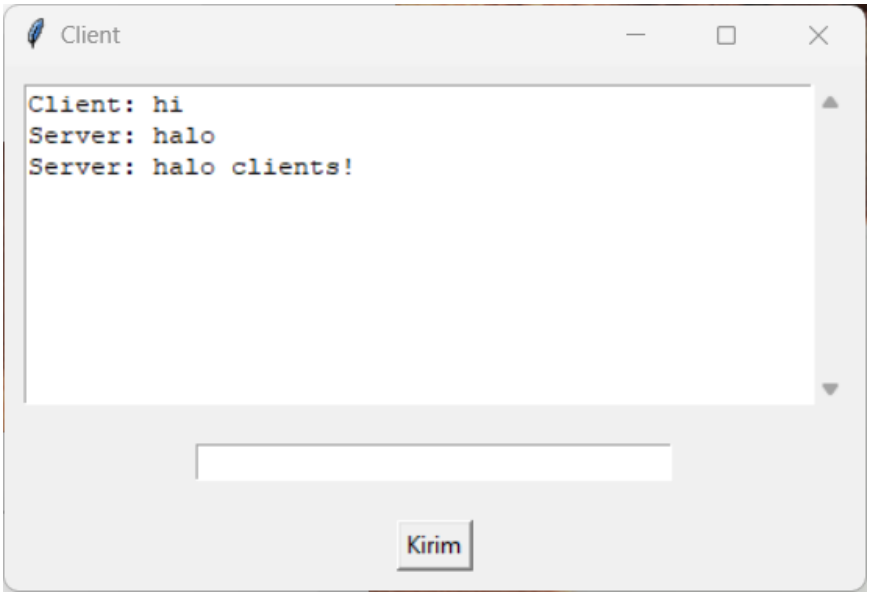
Gambar 7.8 Inisialisasi hubungan dilakukan oleh *client* 1 (sudut pandang *client* 1)



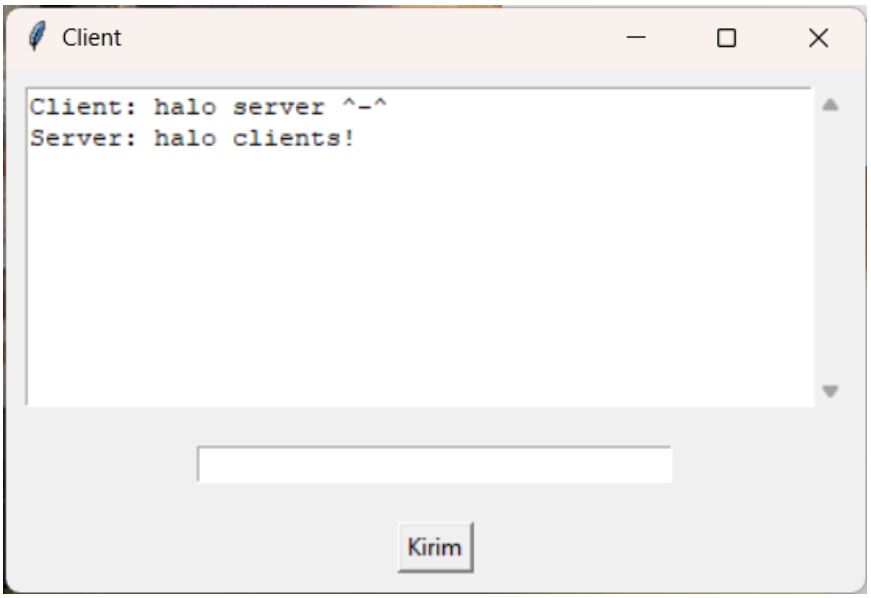
Gambar 7.9 Inisialisasi hubungan dilakukan oleh *client* 1 (sudut pandang *server*)



Gambar 7.10 *Server* mengirimkan pesan secara *broadcast* pada *client* 1 dan 2 (sudut pandang *server*)



Gambar 7.11 *Server* mengirimkan pesan secara *broadcast* pada *client* 1 dan 2 (sudut pandang *client* 2)



Gambar 7.12 *Server* mengirimkan pesan secara *broadcast* pada *client* 1 dan 2 (sudut pandang *client* 1)

REFERENSI

GfG. (2018, April 17). UDP client server using CONNECT: C implementation. GeeksforGeeks. https://www.geeksforgeeks.org/udp-client-server-using-connect-c-implementation/

Situmorang, H. (n.d.). Diktat Kuliah Pemrograman Berorientasi Objek. Bandung; Edunex.