***A Project Report***

***On***

**The SUMMER TRAINING**

# At

**Hindustan Aeronautics Limited**

**Transport Aircraft Division, Kanpur Nagar**



# Submitted By

**Abhishek Kumar Yadav**

**(2100040100003)**

**Raja Balwant Singh Engineering Technical Campus**

**Bichpuri Agra**

**Department of Computer Science & Engineering**

**Dr. APJ Abdul kalam Technical University**

# Certificate

*This is to certify that “****Chat Box using Socket Programming****” is projects done by* ***Abhishek Kumar Yadav*** *to fulfil the requirements of Industrial training program at* ***Hindustan Aeronautics Limited, Transport Aircraft Division, Kanpur*** *under our supervision and guidance, during the period of 9th July, 2024 to 8th August, 2024.*

***Mr. Raman Kumar***

***Chief Manager (IT)***

***HAL, Transport Aircraft Division***

**Acknowledgements**

*We would like to extend my heartfelt gratitude to* ***Mr. Raman Kumar, Chief Manager (IT) of HAL, Transport Aircraft Division, Kanpur*** *for providing me with the opportunity to work on projects of* ***“Chat Box using Socket Programming”****. His expert guidance and continuous encouragement throughout the project have been invaluable in helping me stay focused and motivated from its inception to its successful completion. Specifically, his support has been instrumental in the development of my projects,* ***" Chat Box using Socket Programming "****.*

**Abhishek Kumar Yadav**

**(2100040100003)**

## **Table of Content**

|  |  |  |
| --- | --- | --- |
| Serial No. | Topic | Page No. |
| 1 | History of HAL | 5 – 6 |
| 2 | Transport Aircraft Division, HAL | 7 |
| 3 | Abstract | 8 |
| 4 | Chat Box using Socket Programming | 9 - 12 |
| 5 | Algorithm | 13 |
| 5 | Flow chart | 14 |
| 6 | Source Code | 15-16 |
| 7 | Running the project / Output | 17 |
| 8 | Conclusion | 18-19 |
| 9 | References | 20 |

***History of HAL***

HAL (Hindustan Aeronautics Limited):



Hindustan Aeronautics Limited (HAL) based in

Bangalore, India, is one of Asia's largest aerospace

companies. Under the management of the Indian

Ministry of Defence, this state-owned company is

mainly involved in aerospace industry, which includes

manufacturing and assembling aircraft, navigation and related communication equipment.

HAL built the first military aircraft in South Asia and is currently involved in the design,

fabrication and assembly of aircraft, jet engines, and helicopters, as well as their components

and spares. It has several facilities spread across several cities in India including Nasik,

Korwa, Kanpur, Koraput, Lucknow, Bangalore and Hyderabad. The German engineer Kurt

Tank designed the HF-24 Marut fighter-bomber, the first fighter aircraft made in India.

Hindustan Aeronautics has a long history of collaboration with several other international

and domestic aerospace agencies such as Airbus, Boeing, Sukhoi Aviation Corporation etc.

History:



Hindustan Aeronautics Limited (HAL) came into existence on 1st

October 1964. The Company was formed by the merger of Hindustan

Aircraft Limited with Aeronautics India Limited and Aircraft Manufacturing

Depot, Kanpur.

The Company traces its roots to the pioneering efforts of an industrialist with

extraordinary vision, the late Seth Walchand Hirachand, who set up Hindustan Aircraft Limited at Bangalore in association with the erstwhile princely State of Mysore in December 1940. The Government of India became a shareholder in March 1941 and took over the Management in 1942.

Today, HAL has 20 Production Division and 10 Research & Design Centres in 8 locations in India. The Company has an impressive product track record - 15 types of

Aircraft/Helicopters manufactured with in-house R & D and 14 types produced under license. HAL has manufactured over 3658 Aircraft/Helicopters, 4178 Engines, and Upgraded 272 Aircraft and overhauled over 9643Aircraft and 29775 Engines.

During the 1980s, HAL's operations saw a rapid increase which resulted in the development of new indigenous aircraft such as the HAL Tejas and HAL Dhruv. HAL also developed an advanced version of the Mikoyan-Gurevich MiG-21, known as MiG-21 Bison, which increased its life-span by more than 20 years. HAL has also obtained several multimillion-dollar contracts from leading international aerospace firms such as Airbus, Boeing and Honeywell to manufacture aircrafts spare parts and engines.

2

By 2012, HAL was reportedly bogged down in the details of production and has been slipping on its schedules. On 1 April 2015, HAL reconstituted its Board with TS Raju as CMD, S Subrahmanyan as Director (Operations), VM Chamola as Director (HR), CA Ramana

Rao as Director (Finance) and D K Venkatesh as Director (Engineering & R&D). There are two government nominees in the board and six independent directors.

In March 2017, HAL's chairman and managing director T Suvarna Raju announced that the company had finalised plans for an indigenisation drive. The company plans to produce nearly 1, 000 military helicopters, including, LCH (Light Combat Helicopter) ALH (Advanced Light Helicopter), and over 100 planes over the next 10 years.

HAL will carry out major upgrade of almost the entire fighter fleet of Indian Air Force including Su-30MKI, Jaguars, Mirage and Hawk jets to make them "more lethal". The company will also deliver 123 Tejas Light Combat Aircraft to the IAF from 2018 to 2019, at a rate of 16 jets per year. LCH production will now take place in a newly built Light Combat Helicopter Production Hangar at Helicopter Division in HAL Complex.

.

### **Transport Aircraft Division, HAL**

HAL TAD Kanpur refers to the Transport Aircraft Division (TAD) of Hindustan Aeronautics Limited (HAL) located in Kanpur, India.

The Transport Aircraft Division was established in 1960 as a part of Hindustan Aircraft Limited, which later merged with Aeronautics India Limited and Aircraft Manufacturing Depot, Kanpur to form Hindustan Aeronautics Limited (HAL) in 1964.

The TAD Kanpur is one of the major divisions of HAL and is responsible for the design, development, production, and overhaul of transport aircraft, including military transporters, trainers, and helicopters. The division has played a significant role in the development of India's aerospace industry and has contributed to the country's self-reliance in defense production.

Some of the notable projects undertaken by HAL TAD Kanpur include:

HS-748 Avro: A transport aircraft developed in collaboration with the UK-based Hawker Siddeley Aviation.

Hindustan 228: A light transport aircraft developed by HAL TAD, Kanpur.

AN-32: A medium-lift transport aircraft developed in collaboration with the Ukrainian company Antonov.

The TAD Kanpur has also been involved in the development of indigenous aircraft, such as the HAL HTT-40 basic trainer and the HAL LUH (Light Utility Helicopter).

Today, HAL TAD Kanpur continues to play a vital role in India's aerospace industry, with a focus on design, development, production, and overhaul of transport aircraft, helicopters, and other aerospace systems.

### **Abstract**

This project implements a basic chat server using Python's socket programming framework, designed to facilitate real-time communication over a network through the Transmission Control Protocol (TCP). The server listens for incoming connections on a specific IP address and port number, acting as a central hub for client-server interactions.

Upon initialization, the server binds to the specified IP address and port, then begins accepting connection requests. Once a client connects, the server establishes a communication channel and enters a continuous loop to handle data exchange. During this loop, the server listens for messages from the client and prints them to the console for monitoring. The server operator is then prompted to either respond to the client or terminate the session. If responding, the operator inputs a payload, which is sent back to the client.

The communication continues until the client sends a termination signal ('END') or the operator decides to end the session. Exception handling is integrated to manage potential communication errors, such as connection issues or data transmission failures, ensuring that the server remains stable and connections are properly closed.

This chat server serves as a foundational model for more complex network applications, providing a basic yet effective framework for client-server communication. It offers a starting point for incorporating advanced features like authentication, encryption, or multi-client support. By demonstrating fundamental concepts of socket programming and TCP communication, this project provides a practical basis for further exploration and development in network programming and real-time data exchange.

### **Chat Box using Socket Programming**

**INTRODUCTION**

In the realm of network programming, the ability to create real-time communication systems is a fundamental skill. This project introduces a basic chat server implemented using Python's socket programming capabilities, demonstrating the core principles of client-server communication over a network.

The chat server is designed to use the Transmission Control Protocol (TCP) to establish reliable and ordered communication channels between a server and one or more clients. Utilizing Python's built-in `socket` library, this project showcases the essential steps involved in setting up a network server, managing client connections, and facilitating message exchange.

Upon initiation, the server binds to a specific IP address (10.50.1.124) and port number (5555), preparing to listen for incoming connection requests. Once a client establishes a connection, the server accepts it and creates a communication channel. The server operates in a continuous loop, receiving messages from the client and displaying them to the console. It also provides a simple interactive interface for the server operator to respond to client messages or terminate the session.

The communication process is straightforward: the server waits for data from the client, and the operator can choose to send a response or end the interaction. This loop continues until a termination signal ('END') is received from the client or the operator decides to end the session manually. The implementation includes basic error handling to manage potential issues such as connection interruptions, ensuring a stable and reliable operation.

This basic chat server serves as an educational tool for understanding the fundamental aspects of socket programming and TCP communication. It provides a foundation upon which more complex features, such as multi-client support, authentication, or encryption, can be built. By demonstrating essential networking concepts and practical implementation, this project lays the groundwork for further exploration into advanced network programming and real-time communication systems.

* **Initialization:** The server sets up and configures a socket to listen for client connections.
* **Connection Handling:** It accepts connections from clients, communicates with them by receiving and sending messages, and allows the operator to control the interaction.
* **Robust Operation:** The server handles errors and ensures that each client connection is properly closed after use.

This process enables basic chat functionality where a server can interact with a client in real-time, handling messages and providing a way for the server operator to communicate back.

**What are the benefits of chat box using socket programming?**

Using socket programming to create a chat box (or chat server) provides several benefits. Here’s an overview of the advantages:

**1. Real-Time Communication:**

* **Instant Messaging:** Socket programming facilitates real-time, two-way communication between the client and server, allowing instant messaging and updates.
* **Low Latency:** Socket-based communication typically has lower latency compared to other methods because it operates at a lower level and avoids additional overhead.

2. **Full Control Over Communication:**

* **Custom Protocols:** You can design and implement custom communication protocols tailored to your needs, giving you control over data formats, message handling, and connection management.
* **Direct Data Handling:** Unlike higher-level abstractions, socket programming allows direct access to data transmission and reception, enabling more sophisticated control over the messaging process.

**3. Scalability:**

* **Handling Multiple Clients:** The server can be designed to handle multiple client connections, either sequentially or concurrently (using threading or asynchronous methods).
* **Extensible:** You can scale up the chat server by adding more features, such as chat rooms, private messaging, and user authentication.

**4. Flexibility and Customization:**

* **Protocol Design:** You can create and use custom data protocols suited to your specific requirements, such as implementing special commands or handling different types of messages.
* **Integration:** Sockets can be integrated into a wide range of applications, from simple command-line tools to complex graphical chat applications.

**5. Efficiency:**

* **Resource Utilization:** Socket programming allows efficient use of network resources with minimal overhead. For example, data is transmitted in binary format, which can be more compact and faster compared to text-based protocols.

**6. Learning and Skill Development:**

* **Understanding Networking:** Working with sockets provides a deeper understanding of network protocols, client-server architecture, and how data transmission works at a fundamental level.
* **Troubleshooting Skills:** You gain skills in diagnosing and debugging network-related issues, which can be valuable for more complex networking tasks.

**7. Cross-Platform Capability:**

* **Platform Independence:** Socket programming is supported on various platforms and operating systems, allowing you to create cross-platform chat applications that work on different environments.

**8. Security:**

* **Custom Security Measures:** You can implement custom security measures such as encryption and authentication to ensure secure communication between clients and servers.

### **How does chat box works**

The chat server is designed to handle communication between itself and a client using TCP sockets.

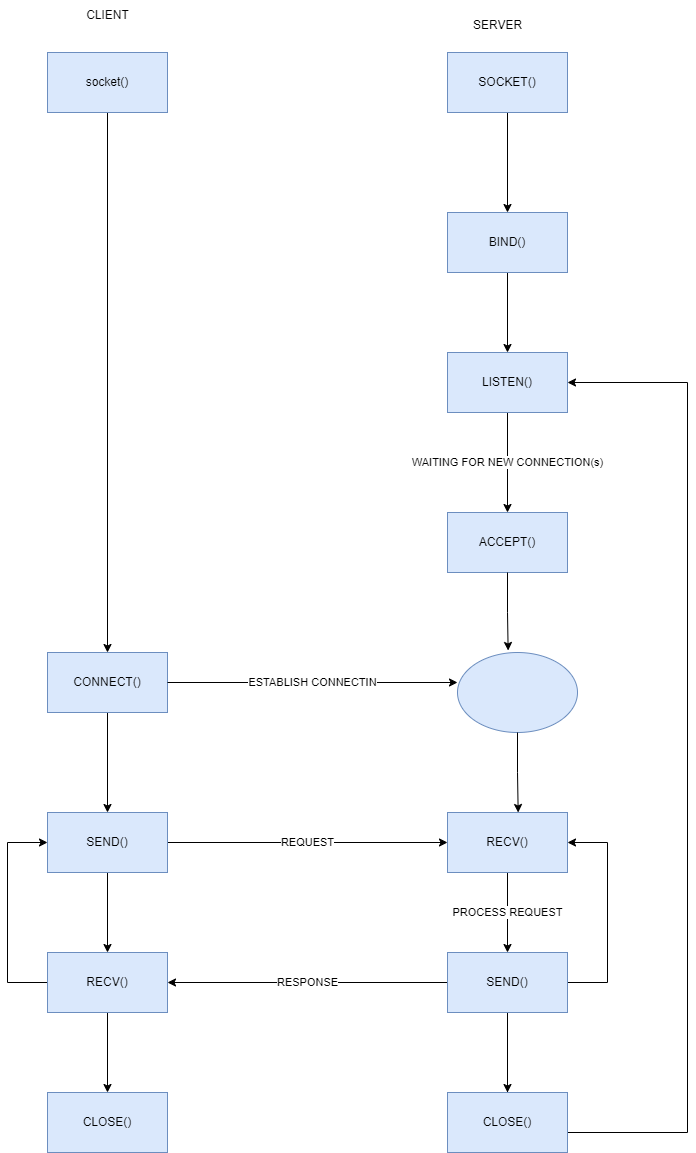
Here’s a step-by-step breakdown of how it operates:

1. **Creating and Configuring the Server Socket:**
   * **Setting Up the Socket:** The server starts by creating a socket using socket.socket(). This socket will use IPv4 addresses (AF\_INET) and TCP (SOCK\_STREAM), which are standard for reliable, connection-oriented communication.
   * **Binding to an Address and Port:** The server then binds this socket to a specific IP address and port number. This means the server will listen for connections on this address and port.
   * **Listening for Connections:** The server\_socket.listen(5) call tells the server to start listening for incoming connections. The number 5 indicates how many connections can be waiting to be processed (queued up) at any one time.
2. **Waiting for and Accepting Client Connections:**
   * **Accepting Connections:** The server enters an infinite loop where it waits for clients to connect. When a client tries to connect, the server accepts the connection using server\_socket.accept(). This method returns a new socket object (client\_socket) specifically for communicating with the connected client, and the client's address information (addr).
3. **Handling Communication:**
   * **Receiving Messages:** Inside another loop, the server waits to receive messages from the client using client\_socket.recv(1024). It reads up to 1024 bytes of data sent by the client. If no data is received or if the client sends a special 'END' message, the loop breaks, ending the communication.
   * **Displaying Messages:** The received messages are decoded from bytes to a string and printed to the server's console so the server operator can see them.
   * **Sending Responses:** The server asks the operator if they want to send a response back to the client. If the operator chooses 'yes', they type in a message (payload), which is then encoded to bytes and sent back to the client. If the operator chooses 'no', the server ends the interaction.
4. **Error Handling and Closing Connections:**
   * **Handling Errors:** If any issues occur during communication (e.g., connection errors), they are caught and printed out. This ensures the server can handle unexpected problems gracefully.
   * **Closing the Connection:** Regardless of whether an error occurred or not, the finally block ensures that the connection to the client is properly closed using client\_socket.close(). This frees up resources and prepares the server to accept new connections.

### **Algorithms**

* **Initialize Server Socket:**
* Create a server socket object using IPv4 (AF\_INET) and TCP (SOCK\_STREAM).
* **Bind Socket to Address:**
* Bind the server socket to the specified IP address (10.50.1.124) and port (5555).
* **Listen for Incoming Connections:**
* Set the server socket to listen for incoming connection requests. Allow up to 5 pending connections.
* **Accept Connections in a Loop:**
* Enter an infinite loop to continuously accept incoming connections.
* Print a message indicating that the server is waiting for a connection.
* Accept a connection from a client, which returns a new socket object (client\_socket) and the address of the client (addr).
* Print a message with the client's address indicating a successful connection.
* **Handle Client Communication:**
* Enter another infinite loop to handle communication with the connected client.
* Receive data from the client with a buffer size of 1024 bytes.
  + If no data is received or if the received data is "END", break out of the loop.
  + Print the received data after decoding it from bytes to a string.
* Prompt the user to decide whether to send data back to the client.
  + If the user inputs 'y', ask for the payload to send and send it to the client after encoding it to bytes.
  + If the user inputs anything other than 'y', break out of the loop.
* **Handle Exceptions:**
* If an exception occurs during communication, print the error message.
* **Close the Client Connection:**
* After exiting the communication loop, close the client socket to end the connection.
* **Repeat for New Connections:**
* Return to the beginning of the main loop to wait for a new connection.

### **Flow chart**



## **SOURCE CODE**

**Server**

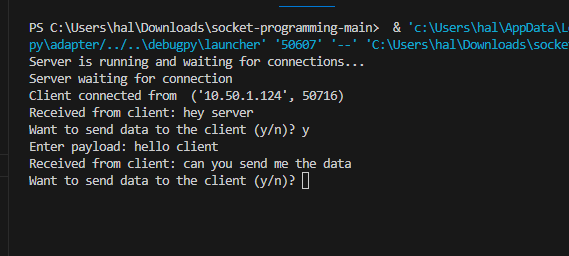


### **Client**

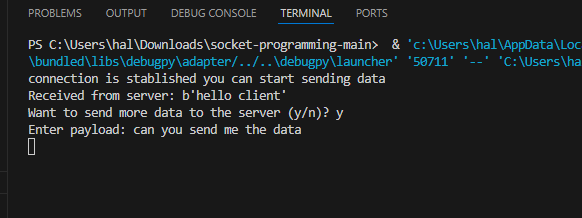
s

### **OUTPUT**

### **Server**



**Client**

****

### **Conclusion**

The chat box project using socket programming successfully demonstrates a fundamental client-server communication model. This project highlights key aspects of network programming, including setting up sockets, managing client connections, and exchanging data between a client and a server.

**Key Accomplishments:**

1. **Socket Initialization and Binding:**
   * The server socket is initialized, bound to a specific IP address and port, and set to listen for incoming connections. This setup establishes the foundation for network communication.
2. **Client-Server Communication:**
   * The server accepts client connections and enters a loop to handle communication. It receives data from clients and provides an interactive way to send responses back.
   * The client connects to the server and engages in sending and receiving messages, with user inputs guiding the interaction. This demonstrates real-time data exchange and client-server dialogue.
3. **Error Handling:**
   * Both the server and client handle exceptions gracefully, ensuring robustness against connection errors and interruptions. For instance, the server handles connection errors, and the client manages scenarios like connection resets and user interruptions.
4. **Interactive Data Exchange:**
   * The project allows interactive data exchange where users can decide whether to continue sending messages or terminate the session. This interactive capability enhances the usability of the chat application.

**Lessons Learned:**

1. **Understanding of Networking Basics:**
   * The project deepens understanding of TCP/IP socket programming, including the roles of client and server, socket binding, and data transmission.
2. **Error Management:**
   * Effective error handling is crucial for maintaining a stable and user-friendly application. The project illustrates how to manage exceptions and ensure reliable operation.
3. **User Interaction:**
   * Incorporating user input into networked applications requires careful consideration of user prompts and data handling, ensuring that the application remains responsive and intuitive.

**Future Improvements:**

1. **Enhanced Security:**
   * Implement encryption mechanisms to secure data transmitted between the client and server, protecting against eavesdropping and tampering.
2. **Concurrency Handling:**
   * Expand the server to handle multiple clients concurrently using threading or asynchronous techniques, allowing simultaneous interactions and scalability.
3. **User Authentication:**
   * Introduce user authentication and authorization features to manage access and personalize interactions.
4. **GUI Integration:**
   * Develop a graphical user interface (GUI) for a more user-friendly experience, moving beyond the command-line interface to a more interactive and visually appealing design.
5. **Robust Error Reporting:**
   * Improve error reporting and logging to better diagnose issues and enhance the troubleshooting process.

This chat box project provides a solid foundation in socket programming and client-server communication. By addressing core concepts and incorporating basic error handling, the project paves the way for more advanced network applications and real-time interactive systems. Future enhancements will further improve the application's functionality, security, and user experience, making it a more robust and versatile tool for real-time communication.

Top of Form

Bottom of Form

### 

### **References**

* <https://hal-india.co.in/home>
* <https://www.datacamp.com/tutorial/a-complete-guide-to-socket-programming-in-python>
* <https://realpython.com/python-sockets/>
* <https://www.geeksforgeeks.org/socket-programming-python/>
* <https://docs.python.org/3/howto/sockets.html>
* Geeksforgeeks.com