MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

Mini Project Report
of
Computer Networks Lab Lab (CSE 2262)

Chat System

SUBMITTED BY

Samarth Parashar 210905206 – Roll No. 40
Aditya Singhvi 210905216 – Roll No. 41
Pranav Mohan 210905204 – Roll No. 39

Department of Computer Science and Engineering Manipal Institute of Technology, Manipal.

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that this report is a bonafide work done by Samarth Parashar, Aditya Singhvi and Pranav Mohan submitted in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology (B.Tech.) in COMPUTER SCIENCE & ENGINEERING of Manipal Institute of Technology, Manipal, Karnataka, (A Constituent Institute of Manipal Academy of Higher Education), during the academic year 2022-2023.

Name and Signature of Examiners:

1. Mr. Manamohana Krishna

TABLE OF CONTENTS

ABSTRACT

This project presents a C-based client-server chat application designed to facilitate secure text communication over a network. Multiple users can connect to the server and engage in real-time chats while benefiting from basic message encryption using a Caesar cipher. The server is equipped to manage multiple client connections, maintain chat history, and execute specific chat-related tasks.

The server code establishes a listening socket on a specified port and awaits incoming client connections. For each connected client, a dedicated thread is created to handle interactions. To prevent data conflicts during concurrent client access, the server employs a mutex for synchronization. Messages sent between clients are encrypted and decrypted using a fixed-shift Caesar cipher, providing rudimentary data security.

Clients connect to the server, establish a separate thread for message reception, and enter a unique username. Users can communicate with others, request chat history, or send private messages. The client code also calculates the Round-Trip Time (RTT) for each message, offering insights into network performance.

This project leverages socket programming, multithreading, and basic encryption techniques to offer the foundation of a secure chat application. While the application is functional, further development is required to enhance encryption methods and strengthen error handling. This report explores the architecture and capabilities of the client-server chat system, discusses its limitations, and suggests avenues for future improvement.

INTRODUCTION

In today's fast-paced digital world, instant communication is a fundamental aspect of our daily lives. With the surge in online chat applications, this project introduces a client-server chat system developed in C. This system allows multiple users to connect to a central server and engage in text-based conversations, all while prioritizing the security of their messages.

The project comprises two main components: the server and the client. The server listens on a designated port, eagerly awaiting connections from clients. When a client connects, a unique thread is assigned to manage their interaction. This multithreaded approach ensures that multiple clients can join the conversation concurrently, creating a dynamic and interactive chat environment.

Security is a critical concern in the world of online communication. To address this, we've implemented a basic encryption technique known as the Caesar cipher for message encryption and decryption. While not the most sophisticated method, it provides an essential layer of security to protect message content.

On the client side, users are prompted to create a personalized username upon connecting to the server. This individualizes the chat experience, making it more engaging. Clients can then send and receive messages, engage in private conversations, request chat history, and even measure the Round-Trip Time (RTT) for messages to gauge network performance.

This project, while a significant first step into the world of client-server chat applications, acknowledges that there's room for improvement. The Caesar cipher encryption method could be bolstered with more robust security measures, and error handling could be further enhanced for greater reliability and safety.

This report will explore the architecture and functioning of the client-server chat system. It will also discuss the role of the Caesar cipher in ensuring secure communication, its limitations, and possible future improvements. The ultimate aim is to create a more robust, secure, and user-friendly chat platform in the future.

OBJECTIVES

1. **Develop a Client-Server Chat Application:** The primary objective is to create a functional client-server chat application that allows users to connect, send, and receive text-based messages in real-time.
2. **Enable Secure Communication:** Implement basic message encryption using a Caesar cipher to provide a fundamental level of security for user messages.
3. **Support Multiple Concurrent Users:** Develop a multithreaded server to handle multiple client connections simultaneously, creating a dynamic and interactive chat environment.
4. **User Personalization:** Prompt users to create unique usernames upon connecting to the server, enhancing the chat experience by distinguishing individual users.
5. **Private Messaging:** Allow users to engage in private conversations by implementing the ability to send messages to specific users, identified by their usernames.
6. **Chat History:** Store and manage chat history, enabling users to request and review past messages for reference and context.
7. **Round-Trip Time (RTT) Measurement:** Implement RTT measurement for sent messages to provide insights into network performance and response times.
8. **Basic Message Broadcast:** Enable users to send messages to all connected clients for group communication.

IMPLEMENTATION

CLIENT CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <pthread.h>
#include <sys/time.h>
#define PORT 10200;
typedef struct Message{
    char username[100];
    char text[1024];
}Message;
void *receive_messages(void *arg) {
    int client_socket = *((int *)arg);
    Message msg;

    while (1) {
        int n = recv(client_socket, &msg, sizeof(Message), 0);
        if (n <= 0) {
            printf("Server disconnected. Exiting...\n");
            exit(0);
        }
        printf("%s: %s",msg.username, msg.text);
    }
}
int main() {
    int client_socket;
    struct sockaddr_in server_addr;
    pthread_t tid;
    client_socket = socket(AF_INET, SOCK_STREAM, 0);
    server_addr.sin_family = AF_INET;
```

```
    server_addr.sin_port = PORT;
    server_addr.sin_addr.s_addr = INADDR_ANY;
    connect(client_socket, (struct sockaddr *)&server_addr, sizeof(server_addr));
    pthread_create(&tid, NULL, receive_messages, &client_socket);
    char username[100];
    printf("Enter Username : ");
    //fgets(username, sizeof(username), stdin);
    scanf("%s",username);
    username[strlen(username)] = '\0';
    char message[1024];
    while (1) {
        struct timeval start, end;
        Message newmsg;
        gettimeofday(&start, NULL);
        fgets(message, sizeof(message), stdin);
        strcpy(newmsg.username , username);
        strcpy(newmsg.text, message);
        send(client_socket, &newmsg, sizeof(Message),0);
        gettimeofday(&end, NULL);
        long seconds = end.tv_sec - start.tv_sec;
        long microseconds = end.tv_usec - start.tv_usec;
        double elapsed = seconds + microseconds / 1e6;
        printf("RTT for message: %f seconds\n", elapsed);
    }

    return 0;
}
```

SERVER CODE:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <pthread.h>
#include <sys/time.h>
```

```
#include <ctype.h>

#define PORT 10200
#define MAX_CLIENTS 10
#define MAX_HISTORY_SIZE 100
typedef struct Message{
    char username[100];
    char text[1024];
}Message;

typedef struct {
    int socket;
    struct sockaddr_in address;
} Client;

Client clients[MAX_CLIENTS];
int client_count = 0;
char usernames[MAX_CLIENTS][100];
int usernames_count = 0;

pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
Message chat_history[MAX_CLIENTS][MAX_HISTORY_SIZE];
int chat_history_size[MAX_CLIENTS] = {0};

void encrypt(char *message, int shift) {
    int length = strlen(message);
    for (int i = 0; i < length; i++) {
        if (isalpha(message[i])) {
            char base = islower(message[i]) ? 'a' : 'A';
            message[i] = (message[i] - base + shift) % 26 + base;
        }
    }
}
// Caesar cipher decryption function
void decrypt(char *message, int shift) {
    encrypt(message, 26 - shift); // Decryption is just shifting in the opposite direction
}
void print_all_messages(int client_no) {
    printf("All Messages from client %d:\n", client_no);
```

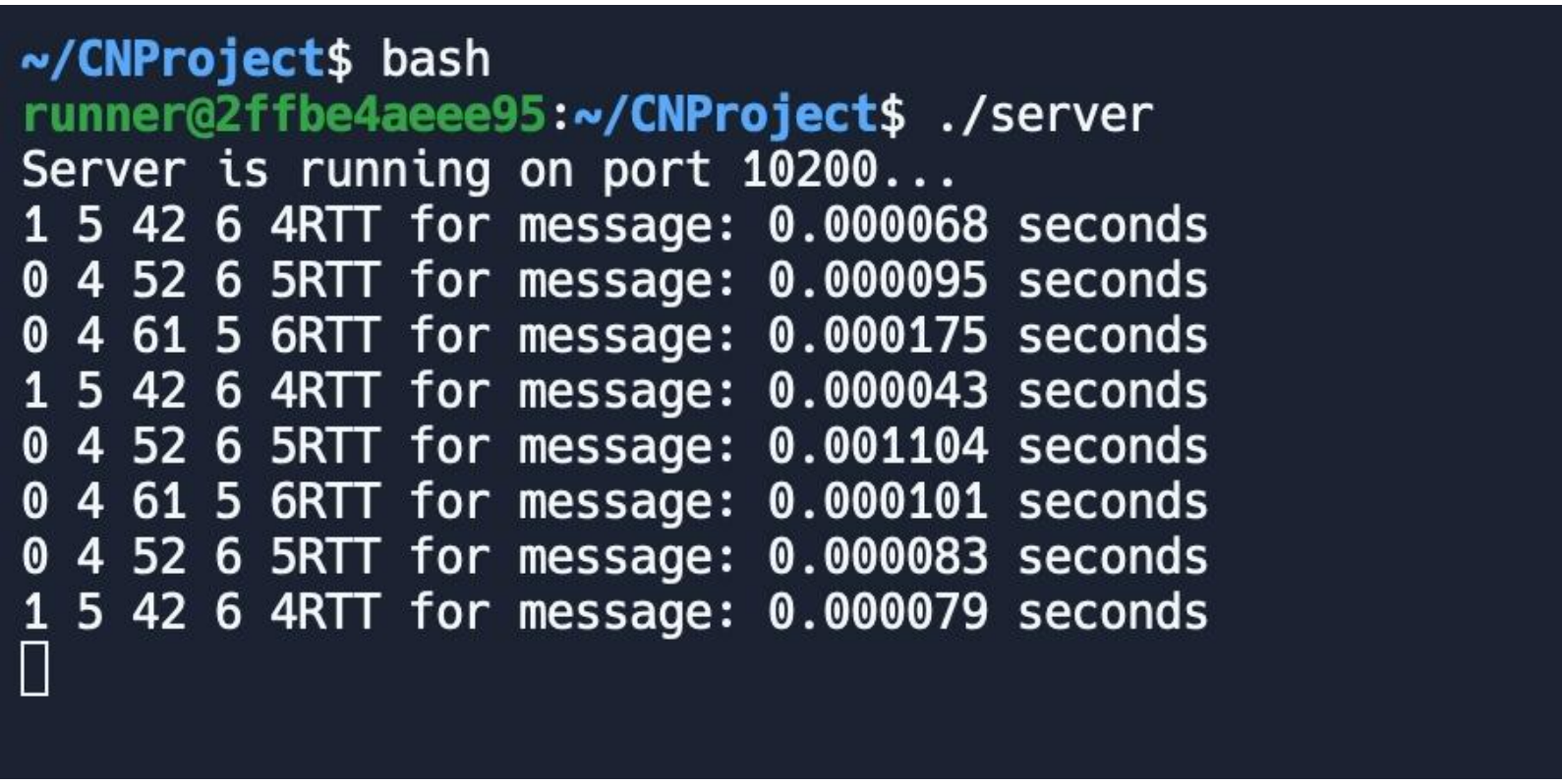
```
    for (int i = 0; i < chat_history_size[client_no]; i++) {
        printf("%s: %s\n", chat_history[client_no][i].username,
chat_history[client_no][i].text);
    }
    printf("End of Messages\n");
}
void add_message_to_history(Message msg, int client_no,int shift) {
    encrypt(msg.text,shift);
    if (chat_history_size[client_no] < MAX_HISTORY_SIZE) {
        chat_history[client_no][chat_history_size[client_no]] = msg;
        chat_history_size[client_no]++;
    } else {
        // Remove the oldest message to make space for the new one
        for (int i = 0; i < MAX_HISTORY_SIZE - 1; i++) {
            chat_history[client_no][i] = chat_history[client_no][i + 1];
        }
        chat_history[client_no][MAX_HISTORY_SIZE - 1] = msg;
    }
}

void send_chat_history(int client_socket, int client_no, int shift) {
    for (int i = 0; i < chat_history_size[client_no]; i++) {
        Message history_msg = chat_history[client_no][i];
        // Decrypt the message before sending it
        decrypt(history_msg.text, shift);
        send(client_socket, &history_msg, sizeof(Message), 0);
        // Encrypt it again in the chat history
        encrypt(chat_history[client_no][i].text, shift);
    }
}
//void send_to_all(char *message, int current_client) {
void send_to_all(Message msg,int current_client){
    pthread_mutex_lock(&mutex);
    if(strlen(msg.username)<1)return;
    for (int i = 0; i < client_count; i++) {
        if (clients[i].socket != current_client) {
            //send(clients[i].socket, message, strlen(message), 0);
            printf("%d %d %d",i,clients[i].socket,current_client);
            send(clients[i].socket, &msg, sizeof(Message), 0);
```

```
        }
    }
    pthread_mutex_unlock(&mutex);
}
void send_to_user(Message msg,int client_no){
    pthread_mutex_lock(&mutex);

    char newText[1024];
    strcpy(newText, msg.text+2);
    strcpy(msg.text, newText);
    send(clients[client_no].socket, &msg, sizeof(Message), 0);

    pthread_mutex_unlock(&mutex);
}
void *handle_client(void *arg) {
    int new_socket = *((int *)arg);
    char buffer[1024];
    int n;
    //char username[100];
    //send(new_socket,"Enter Username :",sizeof("Enter Username :"),0);
    //n = recv(new_socket, username, sizeof(username), 0);
    //username[n] = '\0';
    Message msg;
    while ((n = recv(new_socket, &msg, sizeof(Message), 0)) > 0) {
        //buffer[n] = '\0';

        struct timeval start, end;
        gettimeofday(&start, NULL);
        //send_to_all(buffer, new_socket);

        //msg.username = username;
        //msg.text = buffer;
        if(msg.text[0]=='#'){
            send_chat_history(new_socket, client_count,6);
        }
        else if(msg.text[0] == '@' || isdigit(msg.text[1])){
            send_to_user(msg, (msg.text[1]-'0'));
        }
        else if (msg.text[0]=='%') {
```

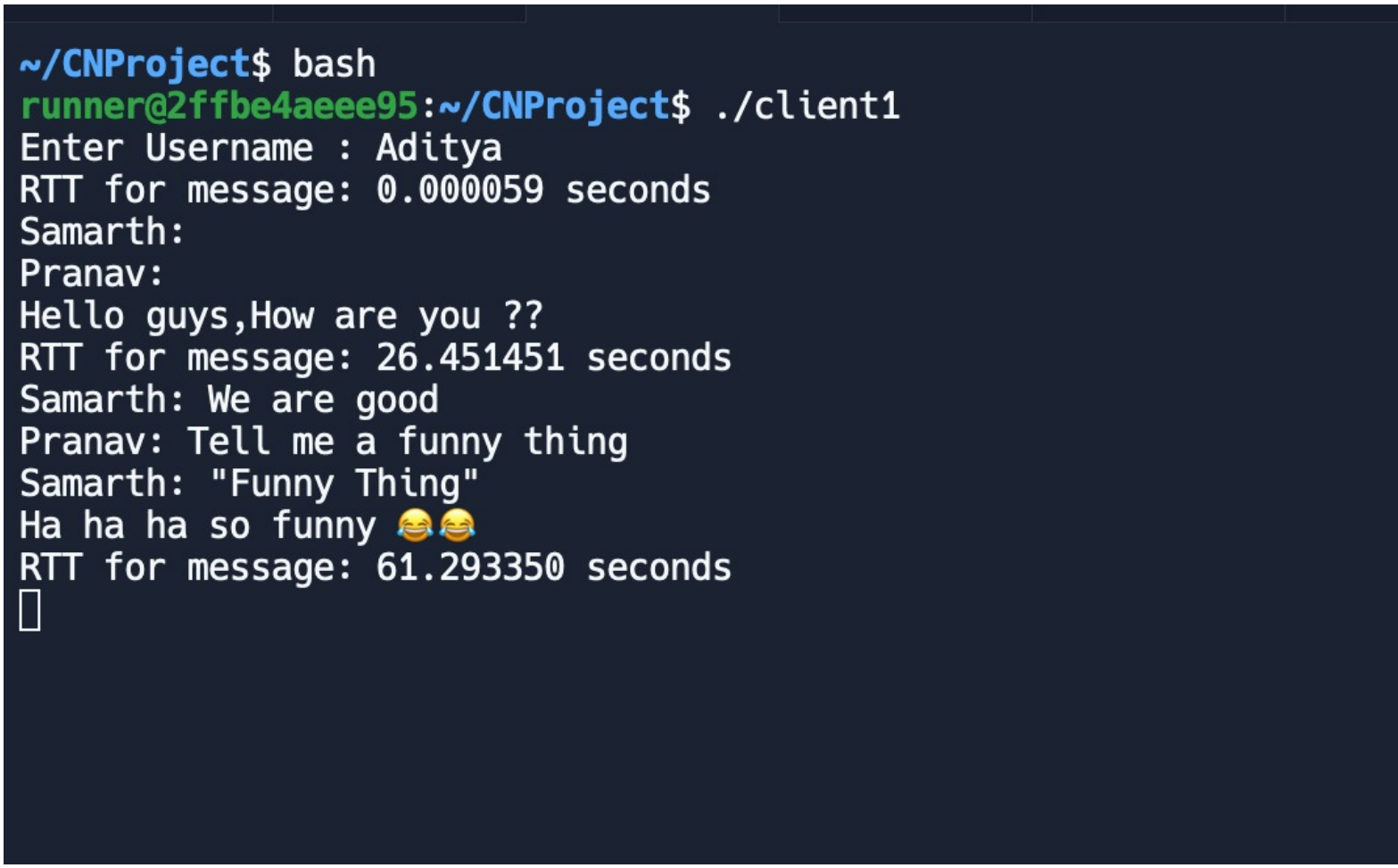
```
            // Print all messages when requested
            print_all_messages(client_count);
        }
        else {send_to_all(msg, new_socket);}
        add_message_to_history(msg, client_count,6);

        gettimeofday(&end, NULL);
        long seconds = end.tv_sec - start.tv_sec;
        long microseconds = end.tv_usec - start.tv_usec;
        double elapsed = seconds + microseconds / 1e6;

        printf("RTT for message: %f seconds\n", elapsed);
    }
    pthread_mutex_lock(&mutex);
    for (int i = 0; i < client_count; i++) {
        if (clients[i].socket == new_socket) {
            memmove(clients + i, clients + i + 1, (client_count - i - 1) * sizeof(Client));
            client_count--;
            break;
        }
    }
    pthread_mutex_unlock(&mutex);
    close(new_socket);
    pthread_exit(NULL);
}
int main() {
    int server_socket, new_socket;
    struct sockaddr_in server_addr, new_addr;
    socklen_t addr_size;

    server_socket = socket(AF_INET, SOCK_STREAM, 0);

    server_addr.sin_family = AF_INET;
    server_addr.sin_port = PORT;
    server_addr.sin_addr.s_addr = INADDR_ANY;

    bind(server_socket, (struct sockaddr *)&server_addr, sizeof(server_addr));
    listen(server_socket, MAX_CLIENTS);
    printf("Server is running on port %d...\n", PORT);
```

```
    while (1) {
        addr_size = sizeof(new_addr);
        new_socket = accept(server_socket, (struct sockaddr *)&new_addr, &addr_size);

        pthread_t tid;
        pthread_create(&tid, NULL, handle_client, &new_socket);

        pthread_mutex_lock(&mutex);
        clients[client_count].socket = new_socket;
        clients[client_count].address = new_addr;
        client_count++;
        pthread_mutex_unlock(&mutex);
    }
    return 0;
}
```

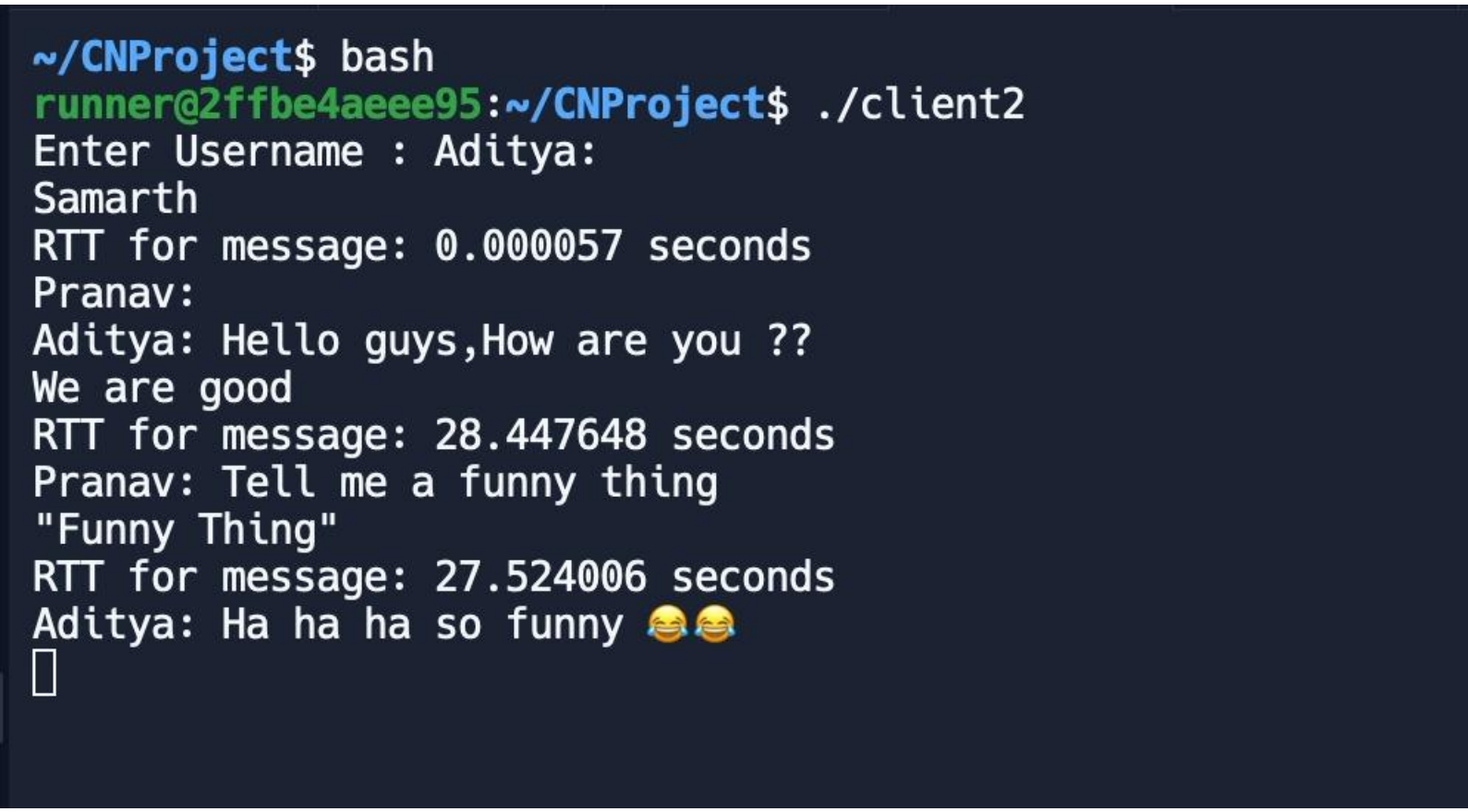
Some Snapshots:

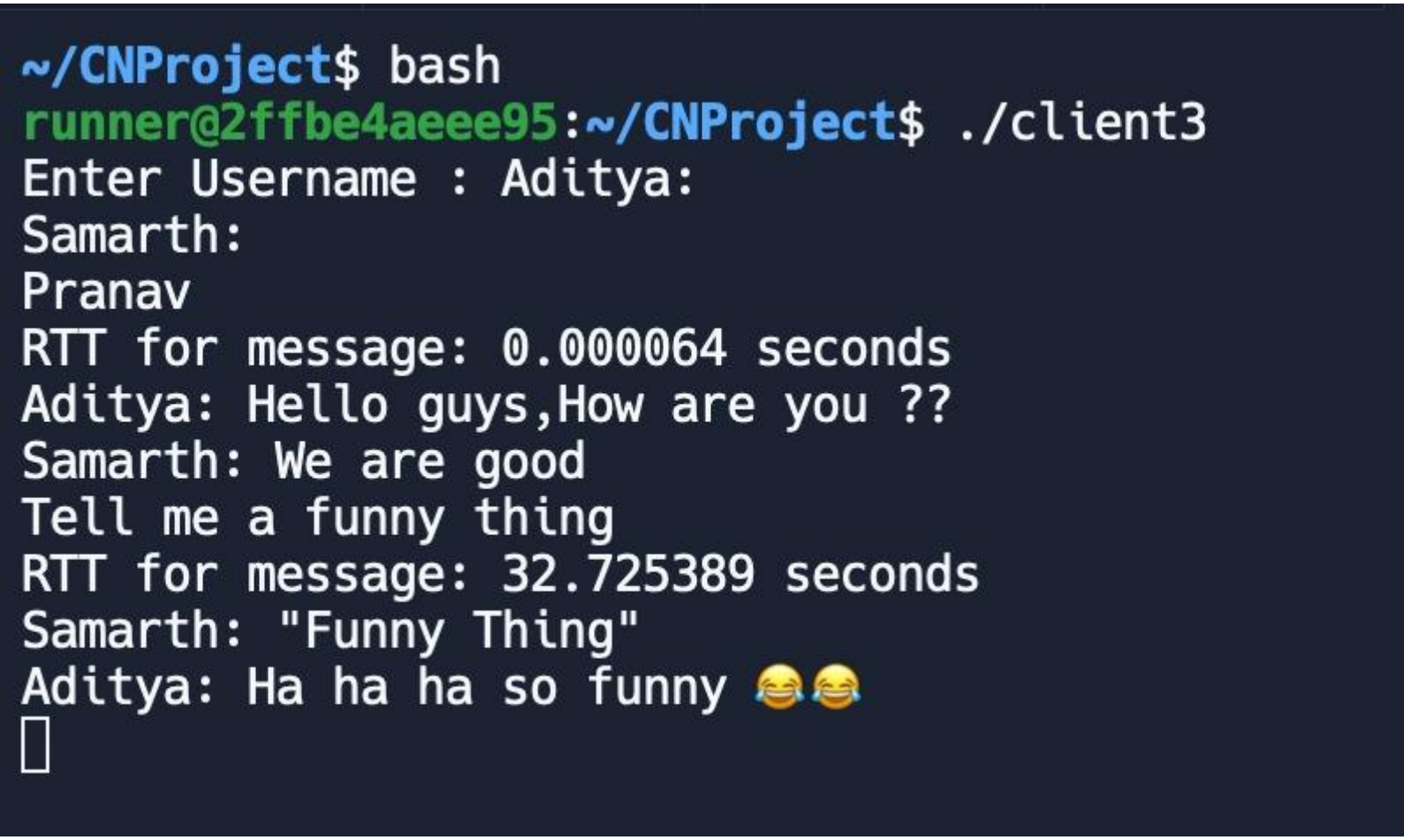


PRIVATE CHATTING:



Encrypted and Decrypted chatbackup:




~/CNProject$ bash
runner@2ffbe4aeee95:~/CNProject$ ./client3
Enter Username : Aditya:
Samarth:
Pranav
RTT for message: 0.000064 seconds
Aditya: Hello guys,How are you ??
Samarth: We are good
Tell me a funny thing
RTT for message: 32.725389 seconds
Samarth: "Funny Thing"
Aditya: Ha ha ha so funny 😂😂

LIMITATIONS AND FUTURE WORK

Limitations:

1. **Basic Encryption:** The project utilizes a basic Caesar cipher for message encryption, which may not offer the robust security needed for sensitive communications.
2. **User Authentication:** The application lacks user authentication, leaving it open to unauthorized access.
3. **Scalability:** While the system handles multiple users, further scalability considerations are necessary for supporting a larger number of concurrent users effectively.
4. **Cross-Platform Compatibility:** The client application is not designed for multiple operating systems, limiting its accessibility

FUTURE WORK:

1. **Advanced Encryption:** Implement more secure encryption methods like end-to-end encryption for heightened message security.
2. **User Authentication:** Integrate user authentication mechanisms to ensure secure user identification.
3. **Scalability Enhancements:** Optimize the server to handle a higher volume of concurrent users and improve overall performance.
4. **Cross-Platform Support:** Develop client applications for various platforms, such as mobile devices and web browsers, to broaden accessibility.
5. **Additional Features:** Add multimedia sharing, group chats, and support for file transfers to enhance the user experience.
6. **Enhanced Security:** Explore advanced security measures like digital signatures and encryption key management.

CONTRIBUTION

1.) Aditya Singhvi- helped in creating basic framework for chat system . Created the functionality where chat backup was being returned and backup was being encrypted and decrypted.Also helped in RTT with Samarth.

2.) Pranav Mohan – Helped in creating basic framework for chat system. Created the functionality of private messaging in the chatbot. Helped in creating and displaying and storing the username function of the clients.

3.) Samarth Parashar – Helped in creating the function to return RTT for the chats. Also helped in overall development process for the project.

CONCLUSION

The client-server chat application project has successfully laid the foundation for a secure and interactive platform that allows users to exchange text-based messages in real-time. While the application meets its initial objectives, it does so with some limitations.

The project has demonstrated the development of a multithreaded server and client application, incorporating basic message encryption using a Caesar cipher. This encryption method provides a foundational level of security but should be enhanced in the future.

In the course of this project, we have also identified areas for improvement. Security can be strengthened through the adoption of advanced encryption techniques and the implementation of user authentication mechanisms. Scalability should be a focus for accommodating a larger user base, and cross-platform compatibility should be explored for broader accessibility.

In conclusion, the client-server chat application serves as a stepping stone towards a more robust and user-friendly communication platform. The journey continues with the pursuit of security, scalability, and enhanced features, aiming to create a seamless and secure messaging experience for users. This project provides a strong foundation for further development and innovation in the realm of real-time chat applications.

References

[1] https://stackoverflow.com/questions/62694256/server-client-chatting-program

[2] https://www.grafiati.com/en/literature-selections/chat-application/

[3] https://www.geeksforgeeks.org/socket-programming-cc/