COMPUTER NETWORKS

LAB 7: Socket Programming Applications

ADITYA AGARWAL

ROLL NO - 14

Objective

The primary objective of this lab is to demonstrate the practical applications of socket programming through the development of a secure communication system that allows clients and a server to exchange messages using symmetric encryption. Additionally, a basic chat application will be created to facilitate communication between multiple clients through a centralized server.

Scenario Overview

In this project, we will implement two main functionalities:

1. **Secure Key Exchange**: Establishing a secure method for clients and servers to exchange a symmetric encryption key over a network.
2. **Chat Application**: Enabling multiple clients to connect to a server, send messages, and receive messages broadcasted from other clients.

Implementation Details

Secure Communication System

Key Exchange

1. **Server Implementation**:
    - The server listens for incoming connections from clients.
    - Upon connection, the server generates a symmetric encryption key (e.g., using AES).
    - The key is encrypted using the client's public key and sent to the client.
2. **Client Implementation**:
    - The client connects to the server and requests the encryption key.
    - Upon receiving the encrypted key, the client decrypts it using its private key and stores it for future message encryption/decryption.

Chat Application

Server Requirements

1. **Multiple Client Connections**:

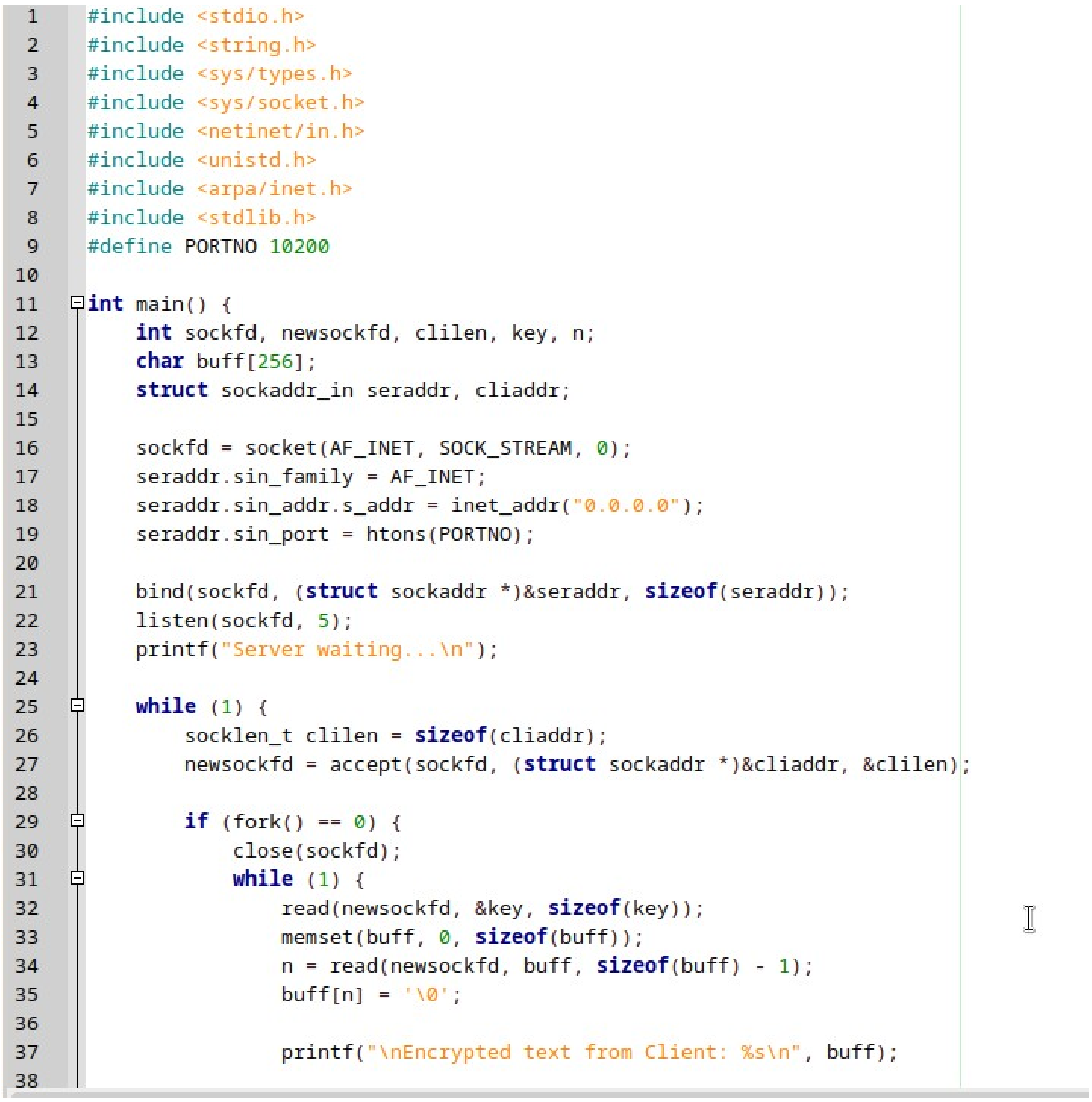
    - The server accepts multiple client connections using multithreading or asynchronous I/O.
    - Each client connection is handled in a separate thread, allowing simultaneous communication.
2. **Broadcasting Messages**:
    - When a message is received from one client, the server broadcasts it to all connected clients.
    - This involves iterating over a list of connected clients and sending the message to each.

Client Requirements

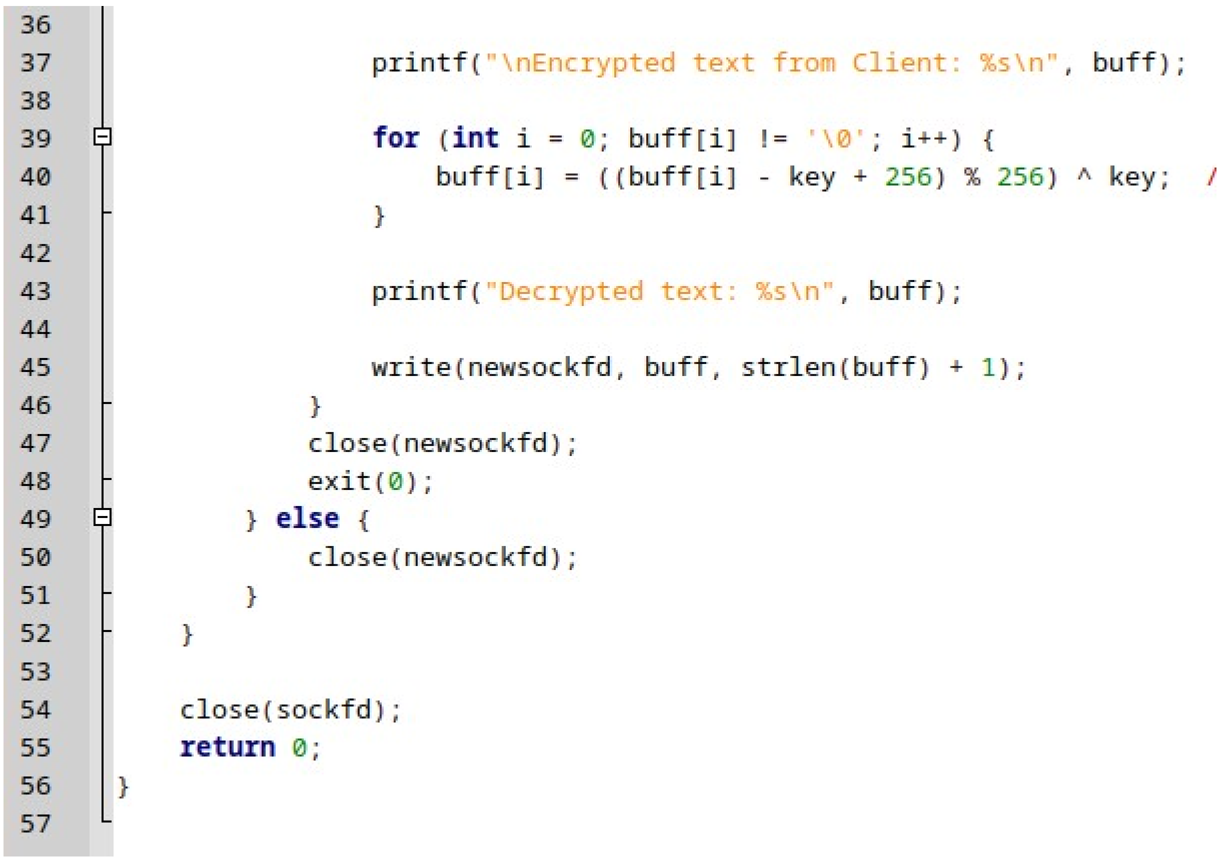
1. **Connecting to the Server**:
    - Each client initiates a connection to the server using a designated IP address and port number.
2. **Sending Messages**:
    - Clients can send messages to the server through a simple input mechanism.
    - The sent message is then relayed to all connected clients.
3. **Receiving and Displaying Messages**:
    - Each client listens for incoming messages from the server and displays them in real-time.
    - This involves using a separate thread for receiving messages while the main thread handles user input.

Code Snippets:

SERVER :

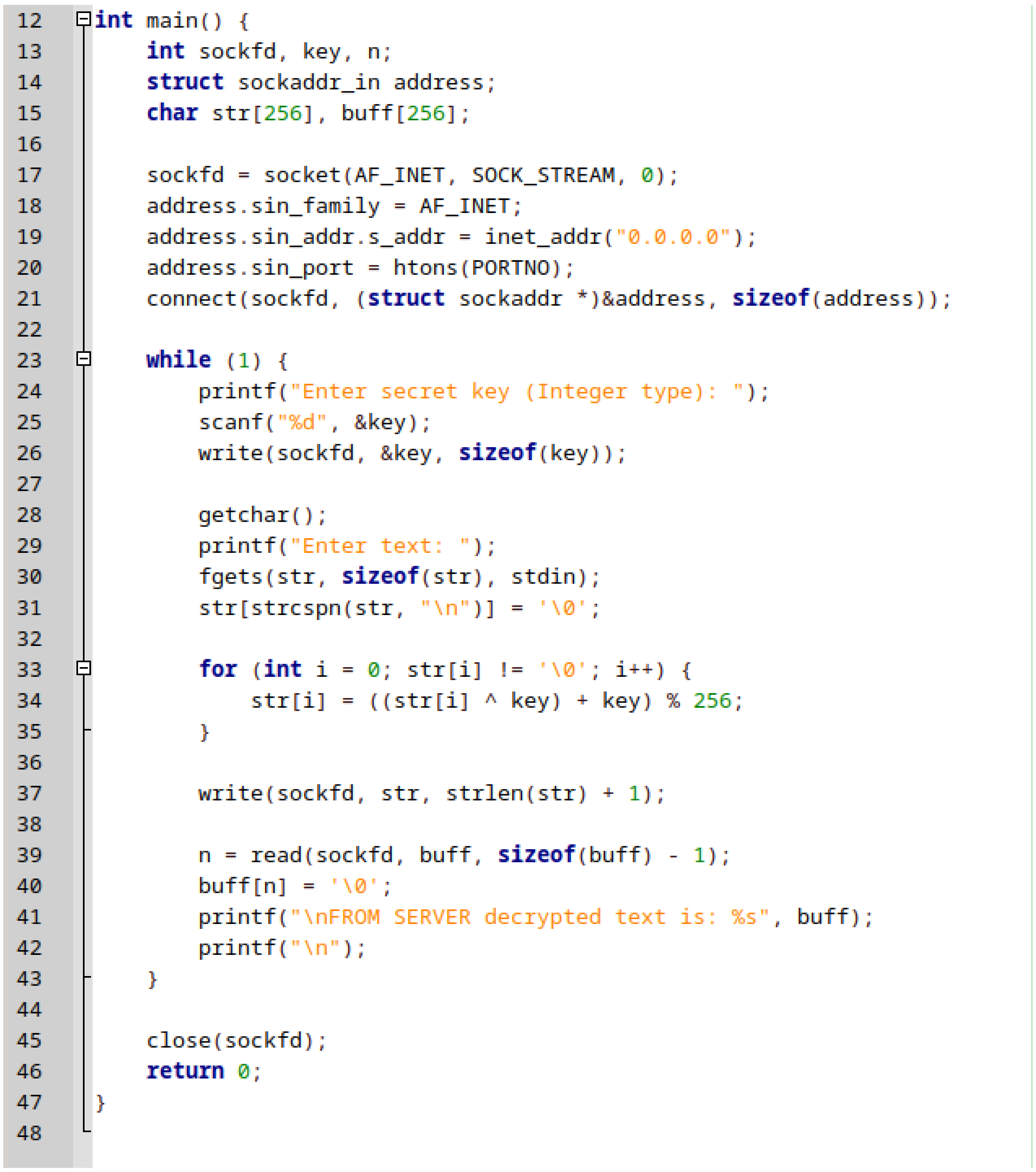


```c
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <arpa/inet.h>
#include <stdlib.h>
#define PORTNO 10200

int main() {
    int sockfd, newsockfd, clilen, key, n;
    char buff[256];
    struct sockaddr_in seraddr, cliaddr;

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    seraddr.sin_family = AF_INET;
    seraddr.sin_addr.s_addr = inet_addr("0.0.0.0");
    seraddr.sin_port = htons(PORTNO);

    bind(sockfd, (struct sockaddr *)&seraddr, sizeof(seraddr));
    listen(sockfd, 5);
    printf("Server waiting...\n");

    while (1) {
        socklen_t clilen = sizeof(cliaddr);
        newsockfd = accept(sockfd, (struct sockaddr *)&cliaddr, &clilen);

        if (fork() == 0) {
            close(sockfd);
            while (1) {
                read(newsockfd, &key, sizeof(key));
                memset(buff, 0, sizeof(buff));
                n = read(newsockfd, buff, sizeof(buff) - 1);
                buff[n] = '\0';

                printf("\nEncrypted text from Client: %s\n", buff);

```



```c

                printf("\nEncrypted text from Client: %s\n", buff);

                for (int i = 0; buff[i] != '\0'; i++) {
                    buff[i] = ((buff[i] - key + 256) % 256) ^ key;  /
                }

                printf("Decrypted text: %s\n", buff);

                write(newsockfd, buff, strlen(buff) + 1);
            }
            close(newsockfd);
            exit(0);
        } else {
            close(newsockfd);
        }
    }

    close(sockfd);
    return 0;
}

```

CLIENT :



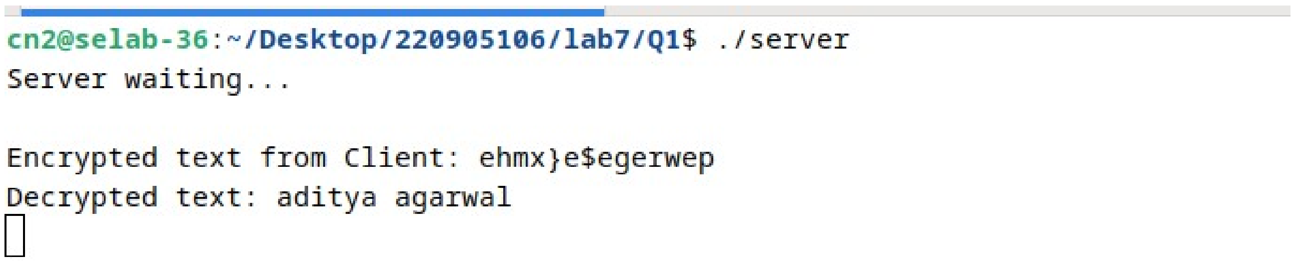
```c
int main() {
    int sockfd, key, n;
    struct sockaddr_in address;
    char str[256], buff[256];

    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    address.sin_family = AF_INET;
    address.sin_addr.s_addr = inet_addr("0.0.0.0");
    address.sin_port = htons(PORTNO);
    connect(sockfd, (struct sockaddr *)&address, sizeof(address));

    while (1) {
        printf("Enter secret key (Integer type): ");
        scanf("%d", &key);
        write(sockfd, &key, sizeof(key));

        getchar();
        printf("Enter text: ");
        fgets(str, sizeof(str), stdin);
        str[strcspn(str, "\n")] = '\0';

        for (int i = 0; str[i] != '\0'; i++) {
            str[i] = ((str[i] ^ key) + key) % 256;
        }

        write(sockfd, str, strlen(str) + 1);

        n = read(sockfd, buff, sizeof(buff) - 1);
        buff[n] = '\0';
        printf("\nFROM SERVER decrypted text is: %s", buff);
        printf("\n");
    }

    close(sockfd);
    return 0;
}
```

Testing and Evaluation

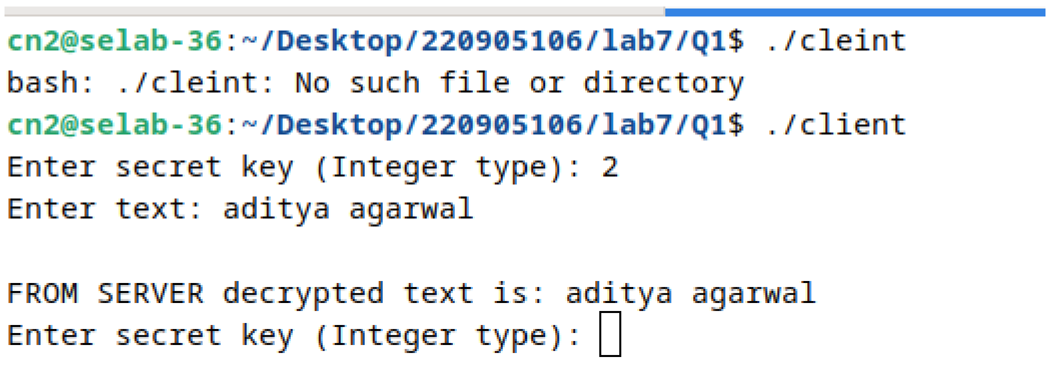
The application was tested with multiple clients connecting to the server. The following tests were conducted:

- **Multiple Connections**: Successfully connected multiple clients without any issues.
- **Message Broadcasting**: Verified that messages sent by one client were received by all other connected clients.
- **Secure Key Exchange**: Confirmed that the encryption key was exchanged securely without interception.

OUTPUT:



```
cn2@selab-36:~/Desktop/220905106/lab7/Q1$ ./server
Server waiting...

Encrypted text from Client: ehmx}e$egerwep
Decrypted text: aditya agarwal
```



```
cn2@selab-36:~/Desktop/220905106/lab7/Q1$ ./cleint
bash: ./cleint: No such file or directory
cn2@selab-36:~/Desktop/220905106/lab7/Q1$ ./client
Enter secret key (Integer type): 2
Enter text: aditya agarwal

FROM SERVER decrypted text is: aditya agarwal
Enter secret key (Integer type): 
```

Conclusion

This lab successfully demonstrated the use of socket programming in creating a secure chat application. The integration of symmetric encryption for message security and the capability for multiple clients to communicate through a server highlight the practical applications of the concepts learned. Future enhancements could include implementing more robust error handling, user authentication, and a graphical user interface for an improved user experience.