Single Client server chat application

```
    while True:

        client_socket, addr = server_socket.accept()

        print(f"Connected with {addr[0]}:{addr[1]}")

        # Start a new thread to handle the client

        client_thread = threading.Thread(target=handle_client, args=(client_socket,))

        client_thread.start()

if __name__ == '__main__':

    main()
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                                  Python

/bin/python3.9 /home/matlab/ClientServer/prac.py
(base) matlab@sjt516scope029:~/ClientServer$ /bin/python3.9 /home/matlab/ClientServer/prac.py
  File "/home/matlab/ClientServer/prac.py", line 14
    print s.recv(1024)
          ^
SyntaxError: invalid syntax
(base) matlab@sjt516scope029:~/ClientServer$ /bin/python3.9 /home/matlab/ClientServer/prac.py
Chat server started on port 8888
```

BIDERECTIONAL-

SERVER:

pracS.py pracC.py

pracS.py > ...

```
import socket
import threading

def handle_client(client_socket):

    while True:

        message = client_socket.recv(1024).decode('utf-8')

        if message == 'exit':

            break

        print(f"Received message: {message}")

    client_socket.close()

def main():

    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    server_socket.bind(('10.30.161.39', 8888))

    server_socket.listen(5)

    print("Chat server started on port 8888")

```

```
    client_socket, addr = server_socket.accept()

    print(f"Connected with {addr[0]}:{addr[1]}")



    # Start a new thread to handle the client

    client_thread = threading.Thread(target=handle_client, args=(client_socket,))

    client_thread.start()



if __name__ == '__main__':

    main()
```

CLIENT:

pracS.py pracC.py

pracC.py > ...

```
import socket

def main():

    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    client_socket.connect(('localhost', 8888))

    print("Connected to server")

    while True:

        message = input("Enter your message: ")

        client_socket.send(message.encode('utf-8'))

        if message == 'exit':

            break

    client_socket.close()

if __name__ == '__main__':

    main()
```

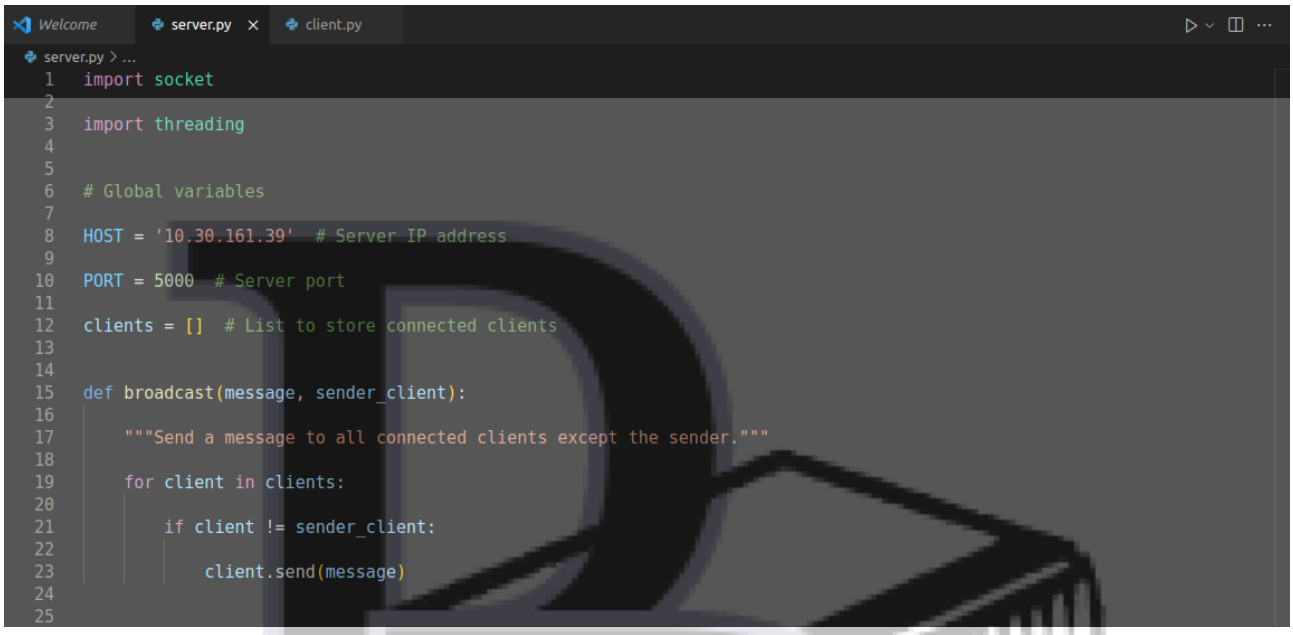
OUTPUT:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL Python

```
/bin/python3.9 /home/matlab/ClientServer/pracS.py
(base) matlab@sjt516scope029:~/ClientServer$ /bin/python3.9 /home/matlab/ClientServer/pracS.py
Chat server started on port 8888
Connected with 10.30.161.38:34770
Received message: HI!!!
Received message: HELLO
```

Multi client -server chat application

SERVER CODE:



```python
import socket

import threading


# Global variables

HOST = '10.30.161.39'  # Server IP address

PORT = 5000  # Server port

clients = []  # List to store connected clients


def broadcast(message, sender_client):

    """Send a message to all connected clients except the sender."""

    for client in clients:

        if client != sender_client:

            client.send(message)
```

```python
def handle_client(client):

    """Handle a client connection."""

    while True:

        try:

            message = client.recv(1024)

            if message:

                broadcast(message, client)

                print(message.decode())  # Display the message on the server

            else:

                remove_client(client)

                break

        except:

            remove_client(client)

            break
```

```python
def remove_client(client):

    """Remove a client from the list of connected clients."""

    if client in clients:

        clients.remove(client)


def start_server():

    """Start the server and listen for incoming connections."""

    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    server_socket.bind((HOST, PORT))

    server_socket.listen(5)

    print('Server started on {}:{}'.format(HOST, PORT))


    while True:

        client_socket, client_address = server_socket.accept()

        clients.append(client_socket)

        print('Connected to {}:{}'.format(client_address[0], client_address[1]))


        client_thread = threading.Thread(target=handle_client, args=(client_socket,))

        client_thread.start()


if __name__ == '__main__':

    start_server()
```

CLIENT CODE:

```python
import socket
import threading


# Global variables

HOST = '127.0.0.1'  # Server IP address
PORT = 5000  # Server port

def receive_messages(client_socket):

    """Receive messages from the server."""

    while True:

        try:

            message = client_socket.recv(1024).decode('utf-8')

            print(message)

        except:

            print('Error receiving messages from the server.')

            client_socket.close()

            break
```

```python
def send_message(client_socket):

    """Send messages to the server."""

    while True:

        try:

            message = input()

            client_socket.send(message.encode('utf-8'))

        except:

            print('Error sending messages to the server.')

            client_socket.close()

            break

def start_client():

    """Connect to the server and start sending/receiving messages."""

    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

    client_socket.connect((HOST, PORT))


    receive_thread = threading.Thread(target=receive_messages, args=(client_socket,))

    receive_thread.start()
```

```python

    send_thread = threading.Thread(target=send_message, args=(client_socket,))

    send_thread.start()


if __name__ == '__main__':

    start_client()
```

OUTPUT:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL                                                    Python

/bin/python3.9 /home/matlab/client-server/server.py
(base) matlab@sjt516scope029:~/client-server$ /bin/python3.9 /home/matlab/client-server/server.py
Server started on 10.30.161.39:5000
Connected to 10.30.161.38:48916
hey
Connected to 10.30.161.40:50590
hi
yayyy
```

Cyclic Redundancy Check

Sender Side

1. The task is to send string data to the server/receiver side.
2. The sender sends a string let us say “EVN”.
3. First, this string is converted to binary string “10001011010110100 1110” key is known to both the side sender and receiver here key used is 1001.
4. This data is encoded using the CRC code using the key on the client/sender side.
5. This encoded data is sent to the receiver.
6. Receiver later decodes the encoded data string to verify whether there was any error or not.

Receiver Side

1. The receiver receives the encoded data string from the sender.
2. Receiver with the help of the key decodes the data and finds out the remainder.
3. If the remainder is zero then it means there is no error in data sent by the sender to the receiver.
4. If the remainder comes out to be non-zero it means there was an error, a Negative Acknowledgement is sent to the sender. The sender then resends the data until the receiver receives the correct data.

```
def crc_encode(data, key):
# Append zeros to the data equal to the length of the key
data += '0' * (len(key) - 1)
# Convert the data and key to lists of bits
data = list(data)
key = list(key)
# Perform the CRC division
for i in range(len(data) - len(key) + 1):
if data[i] == '1':
for j in range(len(key)):
data[i + j] = str(int(data[i + j]) ^ int(key[j]))
# Return the encoded data (including the remainder)
return ''.join(data)
def crc_decode(encoded_data, key):
# Convert the encoded data and key to lists of bits
data = list(encoded_data)
key = list(key)
# Perform the CRC division
for i in range(len(data) - len(key) + 1):
if data[i] == '1':
for j in range(len(key)):
data[i + j] = str(int(data[i + j]) ^ int(key[j]))
# Check if the remainder is zero
if '1' in data:
return False # Error detected
else:
return True # No error

# Sender side
data = input("Enter the data to send: ")
key = "1001" # Assuming a fixed key value
encoded_data = crc_encode(data, key)
print("Encoded data: " + encoded_data)
```

```
# Receiver side
received_data = input("Enter the received data: ")
if crc_decode(received_data, key):
print("No error detected.")
else:
print("Error detected. Resend the data.")
```

OUTPUT

Hamming Code

Steps:

1. Enter the Data to be transmitted
2. Calculate the no of redundant bits required
3. Determine the parity bits
4. Create error data for testing
5. Check for errors

Examples:

Input:

1011001

Output:

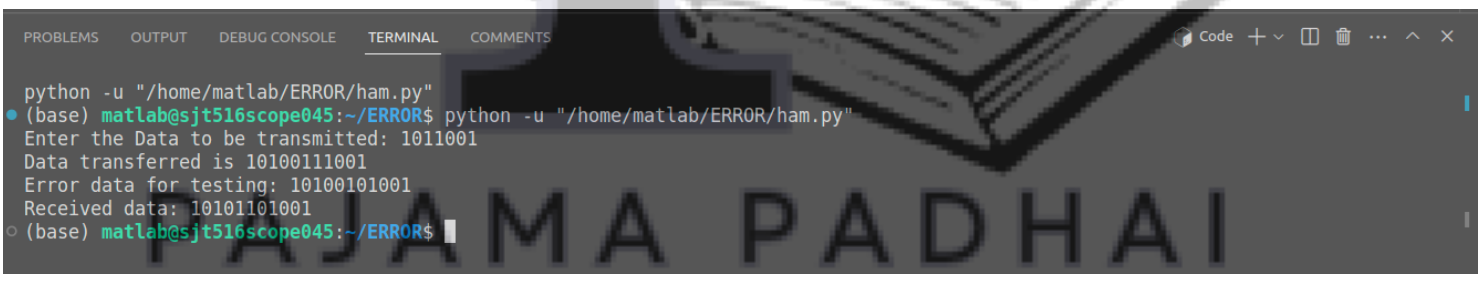
Data transferred is 10101001110

```
def calculate_parity_bits(data):
# Calculate the number of redundant bits required
r = 0
while 2**r < len(data) + r + 1:
r += 1
# Determine the positions of parity bits
positions = []
for i in range(r):
positions.append(2**i - 1)
# Initialize the parity bits to 0
hamming_data = [0] * (len(data) + r)
# Insert data bits into hamming_data at non-parity positions
j = 0
for i in range(len(hamming_data)):
if i in positions:
continue
hamming_data[i] = int(data[j])
j += 1
# Calculate parity bits
for pos in positions:
count = 0
for i in range(len(hamming_data)):
if (i+1) & (pos+1):
if hamming_data[i] == 1:
count += 1
hamming_data[pos] = count % 2
return hamming_data
def create_error_data(data):
# Create error data by flipping a random bit
import random
error_pos = random.randint(0, len(data)-1)
```

```
error_data = list(data)
error_data[error_pos] = 0 if error_data[error_pos] == 1 else 1
return error_data
def check_errors(received_data):
# Check for errors and correct if possible
error_pos = 0
r = 0
while 2**r < len(received_data):
r += 1
for i in range(r):
pos = 2**i - 1
count = 0
for j in range(len(received_data)):
if (j+1) & (pos+1):
if received_data[j] == 1:
count += 1
if count % 2 != 0:
error_pos += pos
if error_pos == 0:
return received_data
else:
error_data = list(received_data)
error_data[error_pos] = 0 if error_data[error_pos] == 1 else 1
return error_data

# Main program
data = input("Enter the Data to be transmitted: ")
data = list(map(int, data))
hamming_data = calculate_parity_bits(data)
print("Data transferred is", ".join(map(str, hamming_data)))
error_data = create_error_data(hamming_data)
print("Error data for testing:", ".join(map(str, error_data)))
corrected_data = check_errors(error_data)
print("Received data:", ".join(map(str, corrected_data)))
```

OUTPUT



CHECKSUM:

```
#include <stdio.h>
#include <string.h>

unsigned int calculateChecksum(const char* str) {
    unsigned int checksum = 0;
```

```c
    // Iterate over each character in the string
    for (int i = 0; str[i] != '\0'; i++) {
        checksum += str[i];
    }

    return checksum;
}

int main() {
    char str[100];

    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin);

    // Remove the trailing newline character
    str[strcspn(str, "\n")] = '\0';

    unsigned int checksum = calculateChecksum(str);

    printf("Checksum: %u\n", checksum);

    return 0;
}
```

Output:

```
PROBLEMS  OUTPUT  TERMINAL  COMMENTS  DEBUG CONSOLE                                   Code

PS C:\Users\Admin\Desktop\prac> cd "c:\Users\Admin\Desktop\prac\" ; if ($?) { gcc main.c -o main } ; if ($?) { .\main }
Enter a string: Hello, My name is Purva!
Checksum: 2066
PS C:\Users\Admin\Desktop\prac>
```

Flow Control:

CODE:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <time.h>

#define FRAME_SIZE 128
#define ACK 1
#define NAK 0

// Function to introduce random errors
void induce_error(char *data)
{
    int r1 = rand() % 2;
    if (r1 == 1)
    {
        int r2 = rand() % strlen(data);
        data[r2] = (data[r2] == '0') ? '1' : '0';
    }
}

// Function to calculate CRC
void calculate_crc(char *data, char *crc_generator, char *crc_result)
{
    strcpy(crc_result, data);

    while (strlen(crc_result) >= strlen(crc_generator))
    {
        for (int i = 0; i < strlen(crc_generator); i++)
        {
            crc_result[i] = (crc_result[i] == crc_generator[i]) ? '0' : '1';
        }

        if (crc_result[0] == '0')
            strcpy(crc_result, crc_result + 1);
        else
            strcpy(crc_result, crc_result);
    }
}

// Function to convert ASCII string to binary string
void ascii_to_bin(char *input, char *binary)
{
    int i, j;
    for (i = 0; i < strlen(input); i++)
    {
        for (j = 7; j >= 0; j--)
        {
            binary[(i * 8) + (7 - j)] = ((input[i] & (1 << j)) ? '1' : '0');
        }
    }
    binary[i * 8] = '\0';
}
```

```c
// Function to append zero bits
void append_zero(char *binary, int num_zeros)
{
    int len = strlen(binary);
    for (int i = 0; i < num_zeros; i++)
    {
        binary[len + i] = '0';
    }
    binary[len + num_zeros] = '\0';
}

// Function to send data packet
void send_packet(char *data, char *crc_result, int *ack)
{
    printf("Sending data packet: %s%s\n", data, crc_result);

    // Randomly introduce error
    induce_error(data);
    induce_error(crc_result);

    // Simulate receiver's response
    int r = rand() % 2;
    if (r == 0)
    {
        printf("Acknowledgement received: OK\n");
        *ack = ACK;
    }
    else
    {
        printf("Acknowledgement received: NAK\n");
        *ack = NAK;
    }
}

// Function to receive data packet
void receive_packet(char *packet, char *crc_generator, char *ack)
{
    // Simulate packet corruption
    int r = rand() % 2;
    if (r == 0)
    {
        printf("Received packet: %s\n", packet);

        // Check for corruption
        char crc_result[FRAME_SIZE];
        calculate_crc(packet, crc_generator, crc_result);
        if (strcmp(crc_result, "000") == 0)
        {
            printf("Packet is error-free.\n");
            *ack = ACK;
        }
        else
        {
            printf("Packet is corrupted.\n");
            *ack = NAK;
        }
```

```
    }
    else
    {
        printf("Received corrupted packet.\n");
        *ack = NAK;
    }
}

// Function to send file
void send_file(char *filename, char *crc_generator)
{
    FILE *file = fopen(filename, "r");
    if (file == NULL)
    {
        printf("Error opening file.\n");
        return;
    }

    char input[FRAME_SIZE];
    char binary[FRAME_SIZE * 8];
    char data[FRAME_SIZE];
    char crc_result[FRAME_SIZE];
    int ack;

    while (fgets(input, FRAME_SIZE, file))
    {
        // Convert ASCII to binary
        ascii_to_bin(input, binary);

        // Append zero bits
        append_zero(binary, strlen(crc_generator) - 1);

        // Generate CRC
        calculate_crc(binary, crc_generator, crc_result);

        // Send packet and receive acknowledgement
        ack = NAK;
        while (ack != ACK)
        {
            strncpy(data, binary, FRAME_SIZE - strlen(crc_generator) + 1);
            send_packet(data, crc_result, &ack);
            if (ack == ACK)
            {
                break;
            }
        }

        if (ack == NAK)
        {
            printf("Resending packet: %s%s\n", data, crc_result);
            send_packet(data, crc_result, &ack);
        }
    }

    // Send end-of-file flag
    printf("Sending end-of-file flag.\n");
    send_packet("", "", &ack);
```

```
    fclose(file);
}

// Function to receive file
void receive_file(char *filename, char *crc_generator)
{
    FILE *file = fopen(filename, "w");
    if (file == NULL)
    {
        printf("Error creating file.\n");
        return;
    }

    char packet[FRAME_SIZE];
    char ack = NAK;

    while (ack != ACK)
    {
        receive_packet(packet, crc_generator, &ack);
        if (ack == ACK)
        {
            break;
        }
        else
        {
            printf("Resending acknowledgement: NAK\n");
        }
    }

    while (strlen(packet) > 0)
    {
        // Extract data bits
        char data[FRAME_SIZE - strlen(crc_generator) + 1];
        strncpy(data, packet, FRAME_SIZE - strlen(crc_generator));
        data[FRAME_SIZE - strlen(crc_generator)] = '\0';

        // Convert binary to ASCII
        char output[FRAME_SIZE / 8];
        for (int i = 0; i < strlen(data) / 8; i++)
        {
            char byte[9];
            strncpy(byte, data + (i * 8), 8);
            byte[8] = '\0';
            output[i] = strtol(byte, NULL, 2);
        }

        fprintf(file, "%s", output);

        // Send acknowledgement
        ack = ACK;
        printf("Sending acknowledgement: OK\n");

        if (ack != NAK)
        {
            receive_packet(packet, crc_generator, &ack);
            if (ack == NAK)
```

```
            {
                printf("Resending acknowledgement: NAK\n");
            }
        }
    }

    fclose(file);
}
```

```
int main()
{
    srand(time(NULL));

    char crc_generator[] = "1011"; // CRC-4

    // Sender
    printf("Sender:\n");
    send_file("input.txt", crc_generator);

    printf("\n-----------------\n\n");

    // Receiver
    printf("Receiver:\n");
    receive_file("output.txt", crc_generator);

    return 0;
}
```

OUTPUT:

```
PROBLEMS  OUTPUT  TERMINAL  COMMENTS  DEBUG CONSOLE                                   Code

PS C:\Users\Admin\Desktop\cn> cd "c:\Users\Admin\Desktop\cn\" ; if ($?) { gcc cn.c -o cn } ; if ($?) { .\cn }
Sender:
Sending data packet: 0100100001100101011011000110110001101111001011000010000001010111011011110111001001101100011001000010000100001000
01010000001
Acknowledgement received: NAK
Sending data packet: 0100100001100101011011000110110001101111001011000010000001010111011011110111001001101100011001000010000100001000
01010000011
Acknowledgement received: OK
Sending data packet: 0101010001101000011010010111001100100000011010010111001100100000011000010010000001110100011001010111001101110
10100001011100000 1011
Acknowledgement received: OK
Sending data packet: 0101001101100101011011100110011000110100101110110011001110010000001100100011000010111010001100001001000000011
01111011101100110 0100
```