Developing chat application: introducing synchronization in multithreaded environment

General description

In this project, you'll be tasked with designing and implementing a multi-room chat application using TCP connections. The goal is to create a robust chat system that allows users to communicate in different rooms and even share files with other members within one room.

Task

It is strictly prohibited to use AI and other tools for generating the solution for your assignment! Your goal is to improve your programming skills, that's why please concentrate on learning instead of cheating.

Rework the client-server application from the assignments 1-2 to create a chat application with rooms support and file transferring feature.

Features to be implemented

1. Supporting multiple rooms, a client should send the room ID while connecting to the server. Example output "CLIENT X JOINED ROOM 1".
2. Message sharing between all clients within 1 room, example output "CLIENT X: Hello", for that it's desired to use Message Queues (see theory).
3. File sharing between all clients, example output "CLIENT X sends a file Test.txt…" and then "File Text.txt has been downloaded".
4. Client should be able to reject receiving the file, example output "CLIENT X wants to send Test.txt file which size is 1 MB, do you want to receive?".
5. Client should be able to rejoin another room.
6. **EXTRA**: the sender should wait until all other room members receive a file.

You should create a document (feel free to use the first page in github) with this task output explanation, it will contain:

1. General system description
2. Application protocol description
3. Screenshots of different use cases (join room, exit room, client force close, send file, accept file, decline file, change room)
4. Explanation of different use cases using UML diagrams and examples from code

Theory

1. Multithreading Overview:

- **Definition:** Multithreading involves the concurrent execution of multiple threads within the same process.
- **Purpose:** In a server application, multithreading is employed to handle multiple client connections simultaneously, providing responsiveness and scalability.

2. Multithreading in Server Applications:

- **Concurrency:** Multiple threads allow the server to manage multiple clients concurrently.
- **Thread Safety:** Ensuring thread safety is crucial to prevent race conditions and data inconsistencies.

3. Race Conditions:

- **Definition:** A race condition occurs when two or more threads access shared data concurrently, and at least one of them modifies the data.
- **Solution:** Use synchronization mechanisms like mutexes to protect critical sections.
- **Example:**

```
int sharedCounter = 0;

void incrementCounter() {
    for (int i = 0; i < 100000; ++i) {
        // Race condition here when calling this function from different
threads
        sharedCounter++;
    }
}
```

4. Mutexes (Mutual Exclusion):

- **Definition:** Mutexes ensure that only one thread can access a critical section of code at a time.
- **Example:**

```
#include <mutex>

int sharedCounter = 0;
std::mutex counterMutex;

void incrementCounter() {
    for (int i = 0; i < 100000; ++i) {
        std::lock_guard<std::mutex> lock(counterMutex);
        sharedCounter++;
    }
}
```

5. Deadlocks:

- **Definition:** A deadlock occurs when two or more threads are unable to proceed because each is waiting for the other to release a resource.
- **Solution:** Avoid circular waiting or use mechanisms like timeouts, use lock_guards not to forget release mutexes.
- **Example:**

```
std::mutex mutexA, mutexB;

void threadA() {
    std::unique_lock<std::mutex> lockA(mutexA);
    // ... some operations ...
    std::unique_lock<std::mutex> lockB(mutexB);  // Potential deadlock
}

void threadB() {
    std::unique_lock<std::mutex> lockB(mutexB);
    // ... some operations ...
    std::unique_lock<std::mutex> lockA(mutexA);  // Potential deadlock
}
```

6. Condition Variables:

- **Definition:** Condition variables are used for signaling between threads and avoiding busy waiting.
- **Example:**

```
#include <condition_variable>

std::mutex dataMutex;
std::condition_variable dataCondition;

int sharedData = 0;

void producer() {
    std::unique_lock<std::mutex> lock(dataMutex);
    sharedData = generateData();
    dataCondition.notify_one();
}

void consumer() {
    std::unique_lock<std::mutex> lock(dataMutex);
    dataCondition.wait(lock, [] { return sharedData != 0; });
    processData(sharedData);
}
```

7. Message Queue:

- **Scenario:** Each chat room may have a message queue where messages from different clients are stored before being broadcasted to others.

- **Usage: Use a condition variable** to signal when a new message is added to the queue, allowing the broadcasting thread to efficiently process and send messages to all clients.

```
std::mutex messageQueueMutex;
std::condition_variable messageAvailableCondition;
std::queue<Message> messageQueue;

void addMessageToQueue(const Message& message) {
    {
        std::lock_guard<std::mutex> lock(messageQueueMutex);
        messageQueue.push(message);
    }
    messageAvailableCondition.notify_one();
}

void broadcastMessages() {
    while (true) {
        std::unique_lock<std::mutex> lock(messageQueueMutex);
        messageAvailableCondition.wait(lock, [] { return
!messageQueue.empty(); });

        // Process and send messages to clients
        while (!messageQueue.empty()) {
            Message message = messageQueue.front();
            messageQueue.pop();
                    // Broadcast the message to all clients in the room
        }
    }
}
```

Examples

Windows

Server

```cpp
// Server.cpp
#include <iostream>
#include <thread>
#include <vector>
#include <string>
#include <mutex>
#include <winsock2.h>

#pragma comment(lib, "ws2_32.lib")

std::mutex consoleMutex;
std::vector<SOCKET> clients;

void broadcastMessage(const std::string& message, SOCKET senderSocket) {
    std::lock_guard<std::mutex> lock(consoleMutex);
    std::cout << "Client " << senderSocket << ": " << message << std::endl;

    for (SOCKET client : clients) {
        if (client != senderSocket) {
            send(client, message.c_str(), message.size() + 1, 0);
        }
    }
}

void handleClient(SOCKET clientSocket) {
    clients.push_back(clientSocket);

    char buffer[4096];
    while (true) {
        int bytesReceived = recv(clientSocket, buffer, sizeof(buffer), 0);
        if (bytesReceived <= 0) {
            std::lock_guard<std::mutex> lock(consoleMutex);
            std::cout << "Client " << clientSocket << " disconnected.\n";
            break;
        }

        buffer[bytesReceived] = '\0';
        std::string message(buffer);
        broadcastMessage(message, clientSocket);
    }

    closesocket(clientSocket);
}

int main() {
    WSADATA wsaData;
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        std::cerr << "WSAStartup failed.\n";
        return 1;
    }
```

```
    SOCKET serverSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (serverSocket == INVALID_SOCKET) {
        std::cerr << "Socket creation failed.\n";
        WSACleanup();
        return 1;
    }

    sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = INADDR_ANY;
    serverAddr.sin_port = htons(8080);

    if (bind(serverSocket, (sockaddr*)&serverAddr, sizeof(serverAddr)) ==
SOCKET_ERROR) {
        std::cerr << "Bind failed.\n";
        closesocket(serverSocket);
        WSACleanup();
        return 1;
    }

    if (listen(serverSocket, SOMAXCONN) == SOCKET_ERROR) {
        std::cerr << "Listen failed.\n";
        closesocket(serverSocket);
        WSACleanup();
        return 1;
    }

    std::cout << "Server is listening on port 8080...\n";

    while (true) {
        SOCKET clientSocket = accept(serverSocket, nullptr, nullptr);
        if (clientSocket == INVALID_SOCKET) {
            std::cerr << "Accept failed.\n";
            closesocket(serverSocket);
            WSACleanup();
            return 1;
        }

        std::lock_guard<std::mutex> lock(consoleMutex);
        std::cout << "Client " << clientSocket << " connected.\n";

        std::thread clientThread(handleClient, clientSocket);
        clientThread.detach(); // Detach the thread to allow handling multiple
clients concurrently
    }

    closesocket(serverSocket);
    WSACleanup();

    return 0;
}
```

Client

```
// Client.cpp
```

```cpp
#include <iostream>
#include <thread>
#include <string>
#include <winsock2.h>
#include <Ws2tcpip.h>


#pragma comment(lib, "ws2_32.lib")

void receiveMessages(SOCKET clientSocket) {
    char buffer[4096];
    while (true) {
        int bytesReceived = recv(clientSocket, buffer, sizeof(buffer), 0);
        if (bytesReceived <= 0) {
            std::cerr << "Server disconnected.\n";
            break;
        }

        buffer[bytesReceived] = '\0';
        std::cout << "Server: " << buffer << std::endl;
    }
}

int main() {
    WSADATA wsaData;
    if (WSAStartup(MAKEWORD(2, 2), &wsaData) != 0) {
        std::cerr << "WSAStartup failed.\n";
        return 1;
    }

    SOCKET clientSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (clientSocket == INVALID_SOCKET) {
        std::cerr << "Socket creation failed.\n";
        WSACleanup();
        return 1;
    }

    sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    InetPton(AF_INET, L"127.0.0.1", &serverAddr.sin_addr);
    serverAddr.sin_port = htons(8080);

    if (connect(clientSocket, (sockaddr*)&serverAddr, sizeof(serverAddr)) ==
SOCKET_ERROR) {
        std::cerr << "Connection failed.\n";
        closesocket(clientSocket);
        WSACleanup();
        return 1;
    }

    std::cout << "Connected to server.\n";

    // Start a thread to receive messages from the server
    std::thread receiveThread(receiveMessages, clientSocket);

    // Main thread to send messages to the server
    std::string message;
    while (true) {
```

```
        std::getline(std::cin, message);
        send(clientSocket, message.c_str(), message.size() + 1, 0);
    }

    // Close the socket and clean up
    closesocket(clientSocket);
    WSACleanup();

    return 0;
}
```

Linux

Server

```
// Server.cpp
#include <iostream>
#include <vector>
#include <cstring>
#include <algorithm> // Include this header for std::find
#include <unistd.h>
#include <arpa/inet.h>
#include <pthread.h>

std::vector<int> clients;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void broadcastMessage(const char* message, int senderSocket) {
    pthread_mutex_lock(&mutex);

    for (int clientSocket : clients) {
        if (clientSocket != senderSocket) {
            send(clientSocket, message, strlen(message), 0);
        }
    }

    pthread_mutex_unlock(&mutex);
}

void* handleClient(void* arg) {
    int clientSocket = *((int*)arg);
    char buffer[4096];

    pthread_mutex_lock(&mutex);
    clients.push_back(clientSocket);
    pthread_mutex_unlock(&mutex);

    std::cout << "Client " << clientSocket << " connected.\n";

    while (true) {
        int bytesReceived = recv(clientSocket, buffer, sizeof(buffer), 0);
        if (bytesReceived <= 0) {
            pthread_mutex_lock(&mutex);
            auto it = std::find(clients.begin(), clients.end(), clientSocket);
            if (it != clients.end()) {
```

```
                clients.erase(it);
            }
            pthread_mutex_unlock(&mutex);

            std::cout << "Client " << clientSocket << " disconnected.\n";
            close(clientSocket);
            pthread_exit(nullptr);
        }

        buffer[bytesReceived] = '\0';
        std::cout << "Client " << clientSocket << ": " << buffer << std::endl;

        broadcastMessage(buffer, clientSocket);
    }

    close(clientSocket);
    pthread_exit(nullptr);
}

int main() {
    int serverSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (serverSocket == -1) {
        std::cerr << "Socket creation failed.\n";
        return 1;
    }

    sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = INADDR_ANY;
    serverAddr.sin_port = htons(8080);

    if (bind(serverSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr)) == -
1) {
        std::cerr << "Bind failed.\n";
        close(serverSocket);
        return 1;
    }

    if (listen(serverSocket, SOMAXCONN) == -1) {
        std::cerr << "Listen failed.\n";
        close(serverSocket);
        return 1;
    }

    std::cout << "Server is listening on port 8080...\n";

    while (true) {
        sockaddr_in clientAddr;
        socklen_t clientAddrLen = sizeof(clientAddr);
        int clientSocket = accept(serverSocket, (struct sockaddr*)&clientAddr,
&clientAddrLen);

        if (clientSocket == -1) {
            std::cerr << "Accept failed.\n";
            close(serverSocket);
            return 1;
        }
```

```
        pthread_t thread;
        if (pthread_create(&thread, nullptr, handleClient, &clientSocket) != 0) {
            std::cerr << "Failed to create thread.\n";
            close(clientSocket);
        }

        pthread_detach(thread);
    }

    close(serverSocket);

    return 0;
}
```

Client

```
// Client.cpp
#include <iostream>
#include <cstring>
#include <unistd.h>
#include <arpa/inet.h>
#include <pthread.h>

void* receiveMessages(void* arg) {
    int clientSocket = *((int*)arg);
    char buffer[4096];

    while (true) {
        int bytesReceived = recv(clientSocket, buffer, sizeof(buffer), 0);
        if (bytesReceived <= 0) {
            std::cerr << "Server disconnected.\n";
            break;
        }

        buffer[bytesReceived] = '\0';
        std::cout << "Server: " << buffer << std::endl;
    }

    close(clientSocket);
    pthread_exit(nullptr);
}

int main() {
    int clientSocket = socket(AF_INET, SOCK_STREAM, 0);
    if (clientSocket == -1) {
        std::cerr << "Socket creation failed.\n";
        return 1;
    }

    sockaddr_in serverAddr;
    serverAddr.sin_family = AF_INET;
    serverAddr.sin_addr.s_addr = inet_addr("localhost");
    serverAddr.sin_port = htons(8080);
```

```
    if (connect(clientSocket, (struct sockaddr*)&serverAddr, sizeof(serverAddr))
== -1) {
        std::cerr << "Connection failed.\n";
        close(clientSocket);
        return 1;
    }

    std::cout << "Connected to server.\n";

    pthread_t receiveThread;
    if (pthread_create(&receiveThread, nullptr, receiveMessages, &clientSocket) !=
0) {
        std::cerr << "Failed to create receive thread.\n";
        close(clientSocket);
        return 1;
    }

    char message[4096];
    while (true) {
        std::cout << "Client: ";
        std::cin.getline(message, sizeof(message));

        send(clientSocket, message, strlen(message), 0);
    }

    // The main thread doesn't exit until Enter is pressed
    pthread_cancel(receiveThread); // Terminate the receiving thread

    // Close the socket and clean up
    close(clientSocket);
    pthread_join(receiveThread, nullptr);

    return 0;
}
```

Evaluation

Attention! The grade will be decreased in case you can't explain how the code works.

Chat application (share messages, message queue)	4
Rooms support (join, rejoin)	3
File transferring (send, receive, decline)	3
Extra: Using condition variable to let sender wait until all clients either receive or decline a file.	2
Total	10 + 2(extra)

Questions

1. What is the purpose of using threads in a server application, especially in the context of a multi-room chat application?
2. How can multithreading enhance the responsiveness and scalability of a server handling multiple client connections?
3. Define what a race condition is in the context of multithreading. Provide an example of a race condition that could occur in a server.
4. Explain the role of mutexes (mutual exclusion) in ensuring thread safety. Provide an example where a mutex is used to protect shared data in a multithreaded server.
5. Define a deadlock in the context of multithreading and try to provide an example scenario that could lead to a deadlock in a chat application with multiple rooms and clients.
6. How can deadlocks be avoided in a multithreaded server application? Discuss at least two strategies for preventing or mitigating deadlocks.
7. Explain the purpose of condition variables in multithreading. Provide an example in a chat application where a condition variable is used to coordinate threads.
8. How does a condition variable facilitate signaling between threads? Discuss a specific scenario in a chat application where signaling is crucial.
9. Describe how condition variables can be employed in a message queue system for a chat application. Why is signaling important in this context?
10. In the context of a chat application, how can a condition variable be used to coordinate file transfers between the sender and receiver threads?

Links

1. Internetworking With TCP/IP by Douglas E. Comer : https://www.homeworkforyou.com/static_media/uploadedfiles/Douglas%20E.%20Comer%20-%20Internetworking%20with%20TCP_IP%20Volume%20One.%201-Addison-Wesley%20(2013).pdf

2. Computer Systems: A Programming Perspective: https://github.com/iWangMu/Book-CSAPP/blob/master/_Attachments/Computer_Systems_A_Programmers_Perspective(3rd).pdf Chapters 11-12