Chat Room

1930026132 萧霄
1930026019 程仕杰
1930026165 张玉媛
1930026049 黄士鑫

Contents

Part 1 ： Introduction

Part 2 ： Data Structure

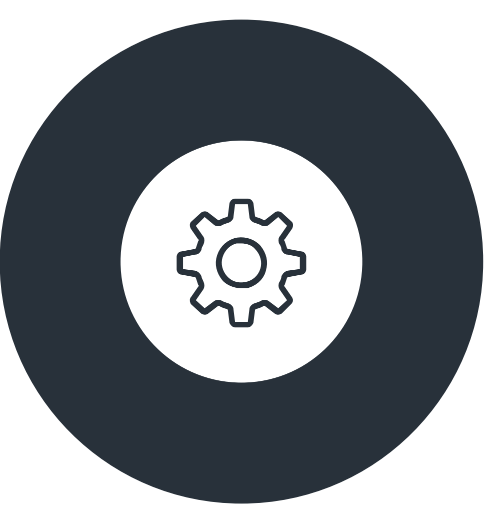
Part 3 ： Implementation

Part 4 ： Demonstration

How to create an online group chat room?

Basic network communication

Multithread processing



Concise message management

Basic network communication

bind():
Bind the master socket to a port number to let clients know where to locate the socket and connect to it.

listen(), accept():
Listen and receive connection request from client

WSAStartup(), socket():
Create a socket

connect():
Let clients connect to the server

send(), recv():
Send and receive message between client and server

Multithread processing

1. For client

```
CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE)receive, NULL, NULL, NULL);
while (1){
    memset(buff, 0, 1024);  // initialize
    printf("input character string:\n");
    fgets(buff, sizeof(buff), stdin);
    send(clientSocket, buff, strlen(buff), NULL);
}
```

```
void receive(){
    char recvBuff[1024];
    int r;
    while (1){
        r = recv(clientSocket, recvBuff, 1023, NULL);
        if (r > 0){
            recvBuff[r] = 0;
            outtextxy(0, count * 20, recvBuff);
            count++;
        }
    }
```

2. For server

```
void listenThread(int idx){
    // accepet the message from clients
    while (1){
        clientSocket[count] = accept(serverSocket, (sockaddr*)&cAddr, &len);
        if (clientSocket[count] == SOCKET_ERROR){
            printf("The server is crashed!\n");
            // close the socket
            closesocket(serverSocket);
            WSACleanup();
            return ;
        }
        printf("A client is connected to the server %s! user_id:%d\n", inet_ntoa(cAddr.sin_addr),count);

        addUser(count,0);
        CreateThread(NULL, NULL, (LPTHREAD_START_ROUTINE)communicate, (char*)count, NULL, NULL);
        count++;
    }
}
```

In order to deal with the synchronous communication between multiple clients

```
void communicate(int idx) {
    char buff[1024] = { 0 };
    char msg[1024] = { 0 };
    int r;
    while (1) {
        r = recv(clientSocket[idx], buff, 1023, NULL);
        if (r > 0) {
            buff[r] = 0;

            //if client is asking for some functions:
            if(buff[0] == '*'){
                std::vector<std::string> split_list;
                Stringsplit(buff, ' ', split_list);
                clientRequest(split_list,idx);
            }
            //if client send a common message:
            else {
                sprintf_s(msg, "%d:%s", idx, buff);
                printf("%s", msg);
                // Broadcast the data
                broadcast(idx, msg);
            }
        }
    }
}
```

Concise message management

All instructions and information mixed together in client: hard to recognize!

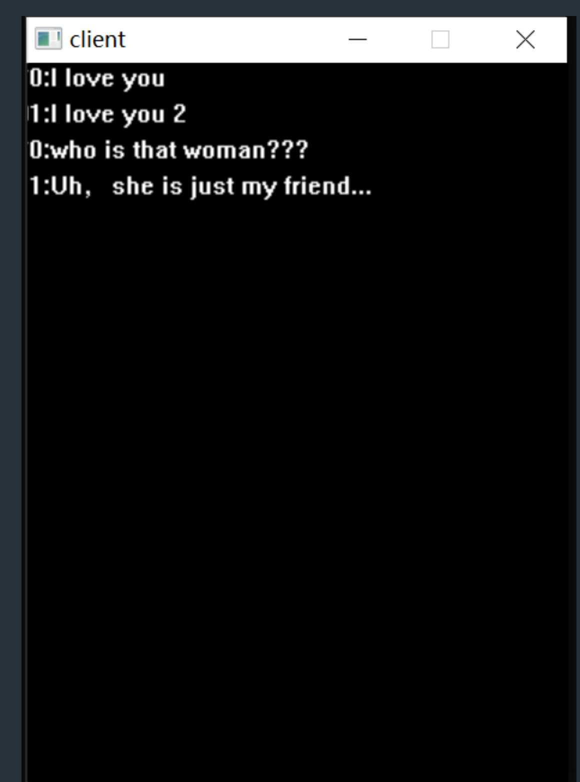
EasyX will be a perfect solution!

```
#include <graphics.h>//easyX
```

```
int main(){
    // initialize the chatting window
    hWnd = initgraph(300, 400, SHOWCONSOLE);
```

Create a chat window while a client is created.

```
void receive(){
    char recvBuff[1024];
    int r;
    while (1){
        r = recv(clientSocket, recvBuff, 1023, NULL);
        if (r > 0){
            recvBuff[r] = 0;
            outtextxy(0, count * 20, recvBuff);
            count++;
        }
    }
```



Demonstrate all the messages received on the chat window!

Contents

Part 1 ： Introduction

Part 2 ： Data Structure

Part 3 ： Implementation

Part 4 ： Demonstration

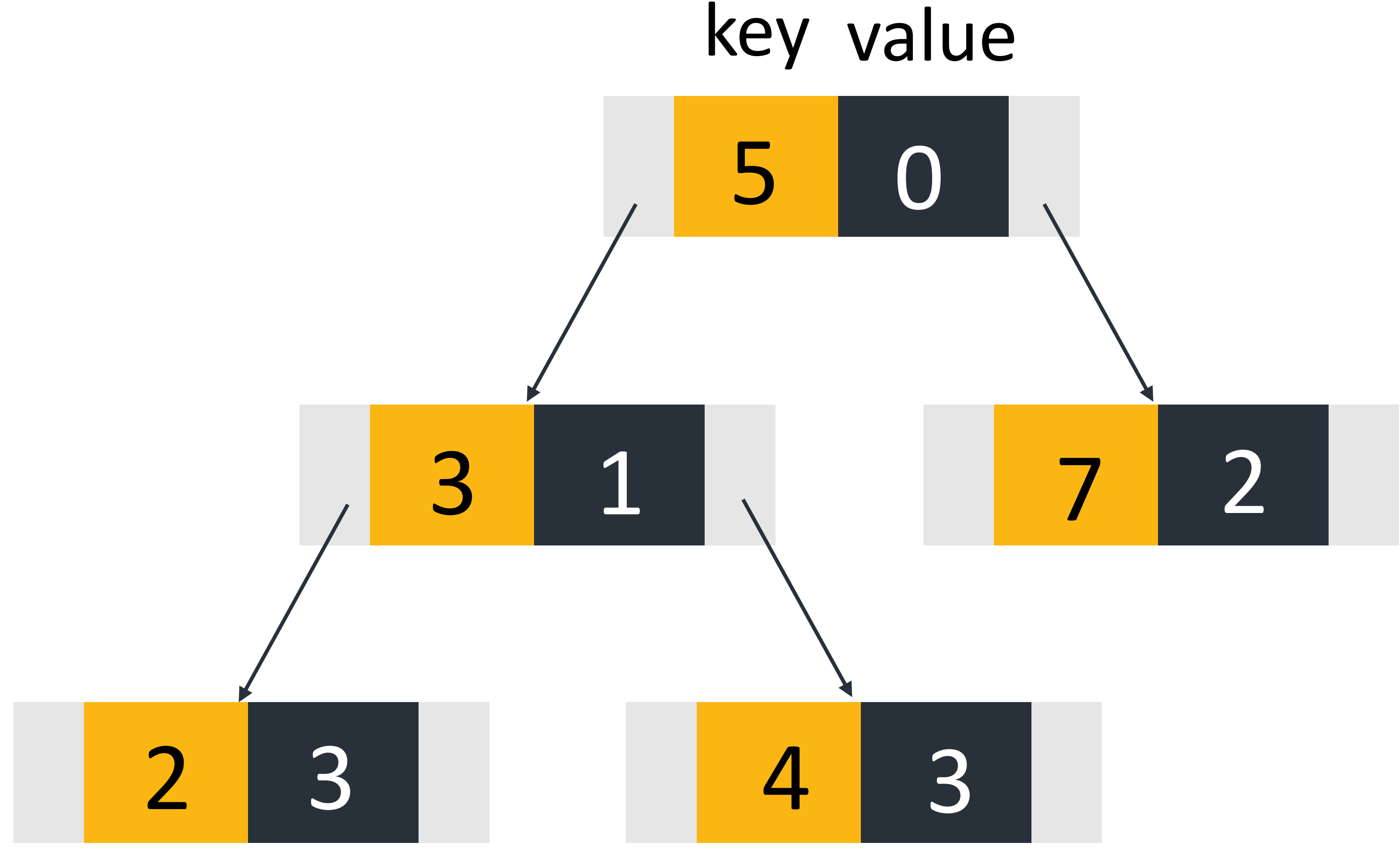
Data Structure

Map

An associated container from STL

Struct

For storing the private chat information



Contents

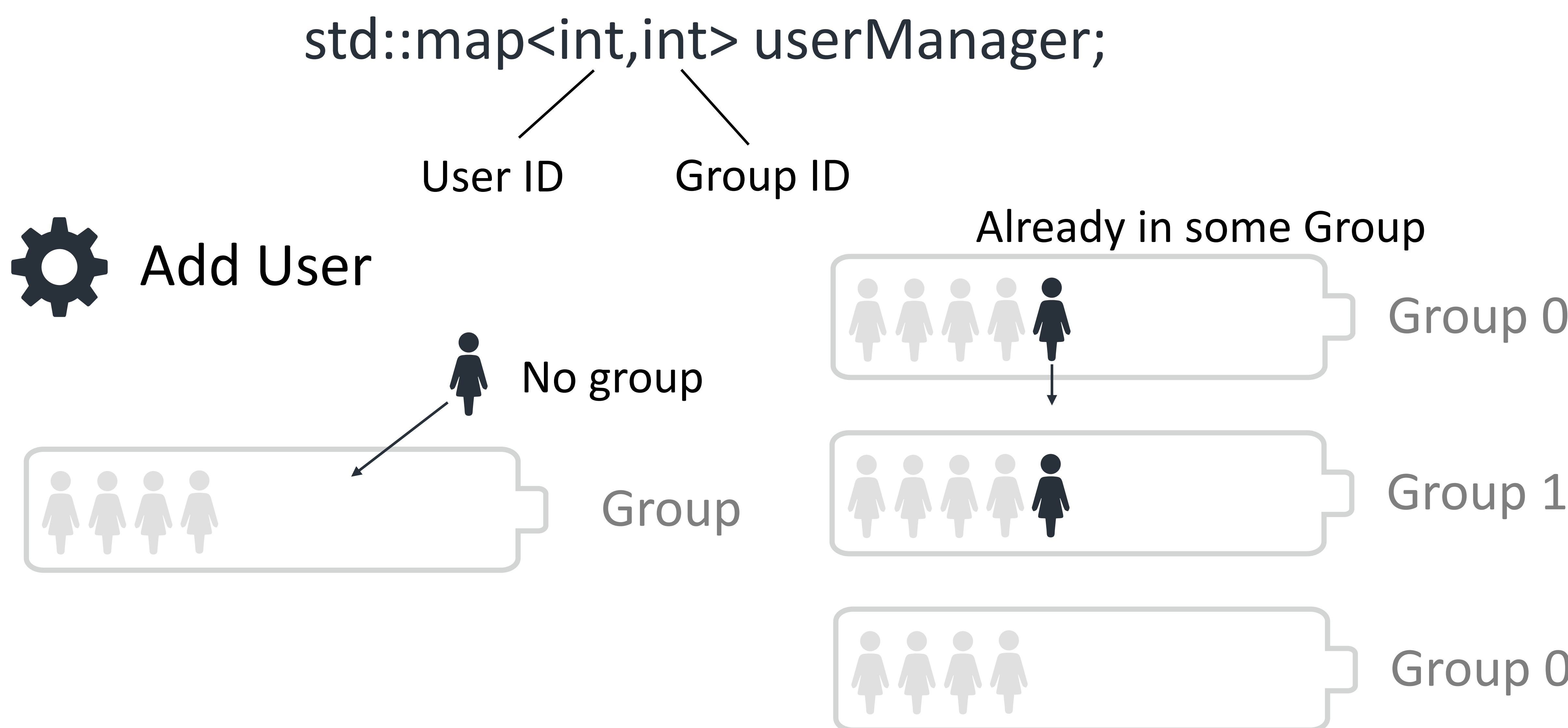
Part 1 : Introduction

Part 2 : Data Structure

Part 3 : Implementation

Part 4 : Demonstration

Implementation-Server



Implementation-Server

std::map<int,int> userManager;

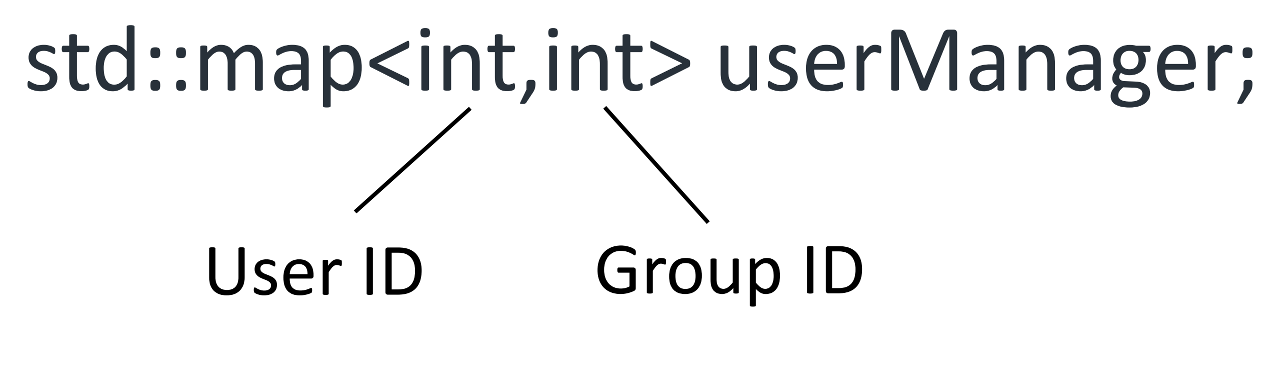
User ID Group ID

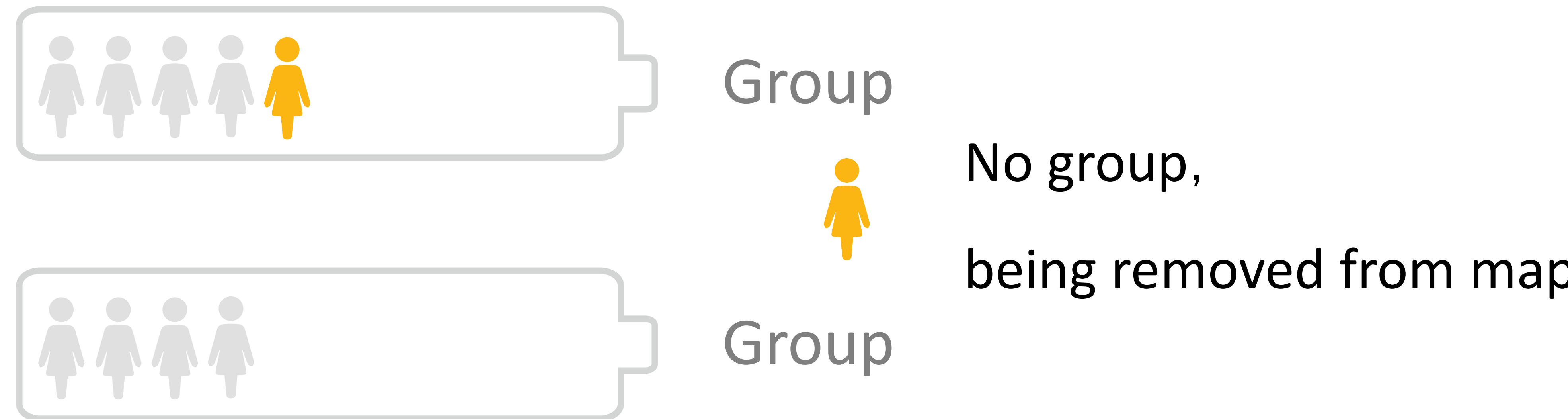
Add User

```
// add user(client)
void addUser(int idx,int group)
{
    std::map<int,int>::iterator it = userManager.find(idx);
    if(it == userManager.end())
    {
        std::vector<int> userVec;
        userVec.push_back(idx);
        userManager[idx] = group;
    }
    else
    {
        it->second = group;
    }

    printUserManger();
}
```

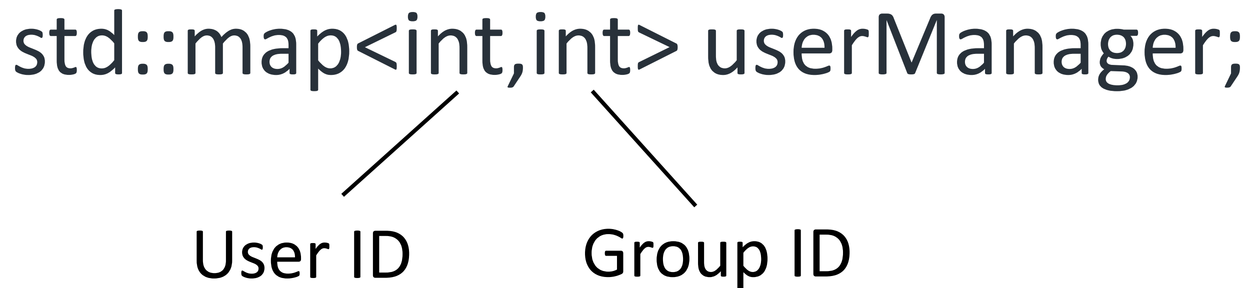
```
//print the user information
void printUserManger(){
    printf("user:group ");
    for(std::map<int,int>::iterator it = userManager.begin();it != userManager.end();++it)
    {
        printf("%d:%d ",it->first,it->second);
    }

    printf("\n");
}
```

Implementation-Server





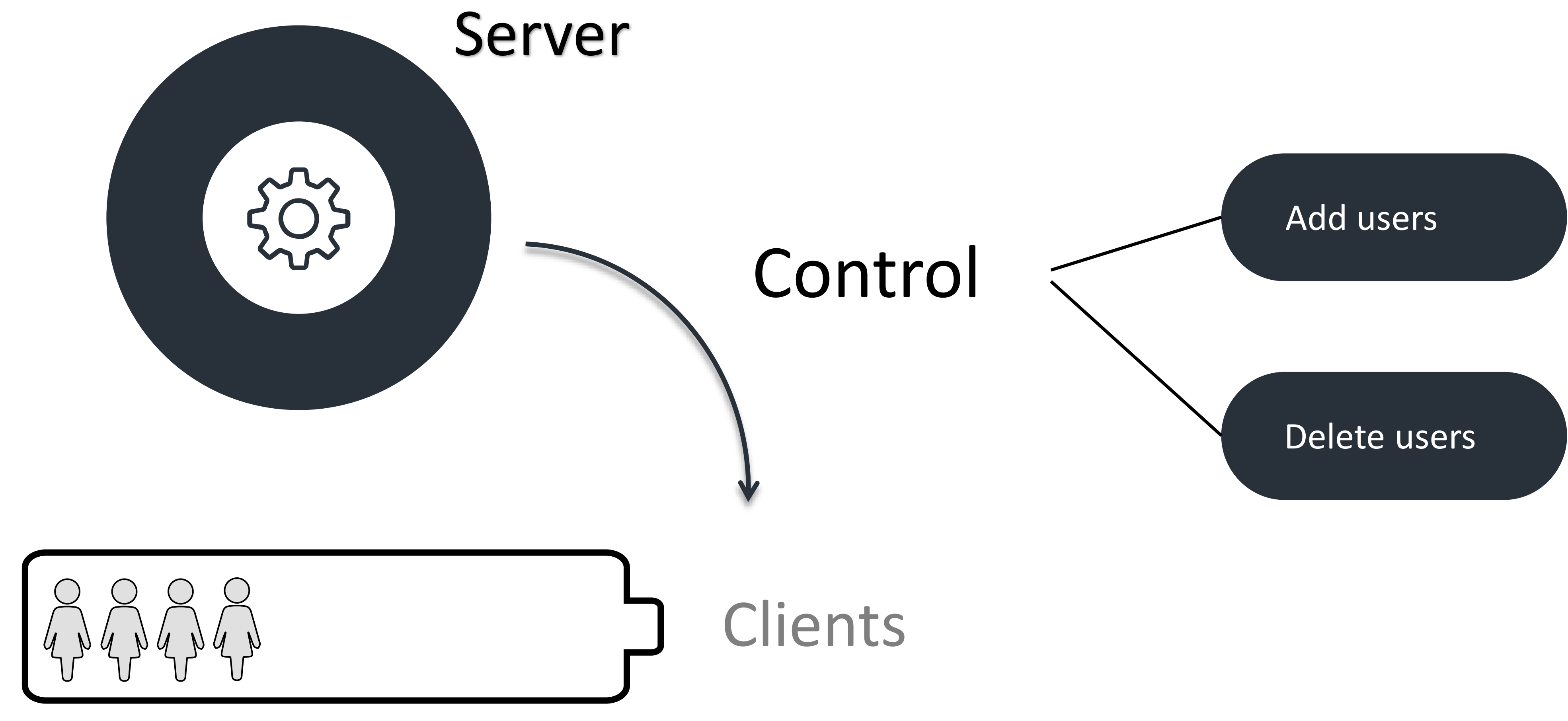
Implementation-Server



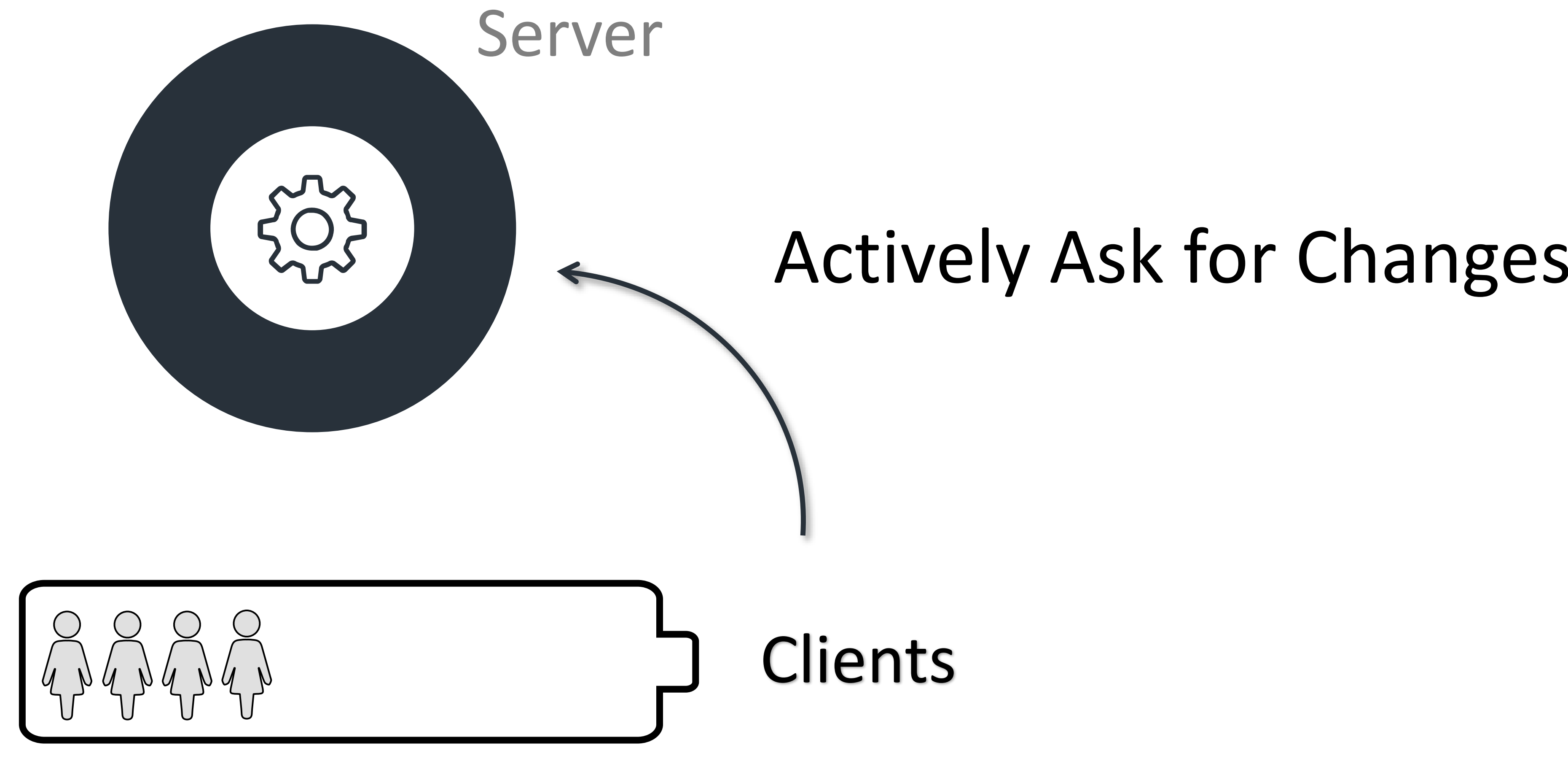
Delete User

```
    int user = atoi(split_list[1].c_str());
    std::map<int,int>::iterator it = userManager.find(user);
    if(it != userManager.end())
    {
        userManager.erase(it);
    }

    printUserManger();
}
```

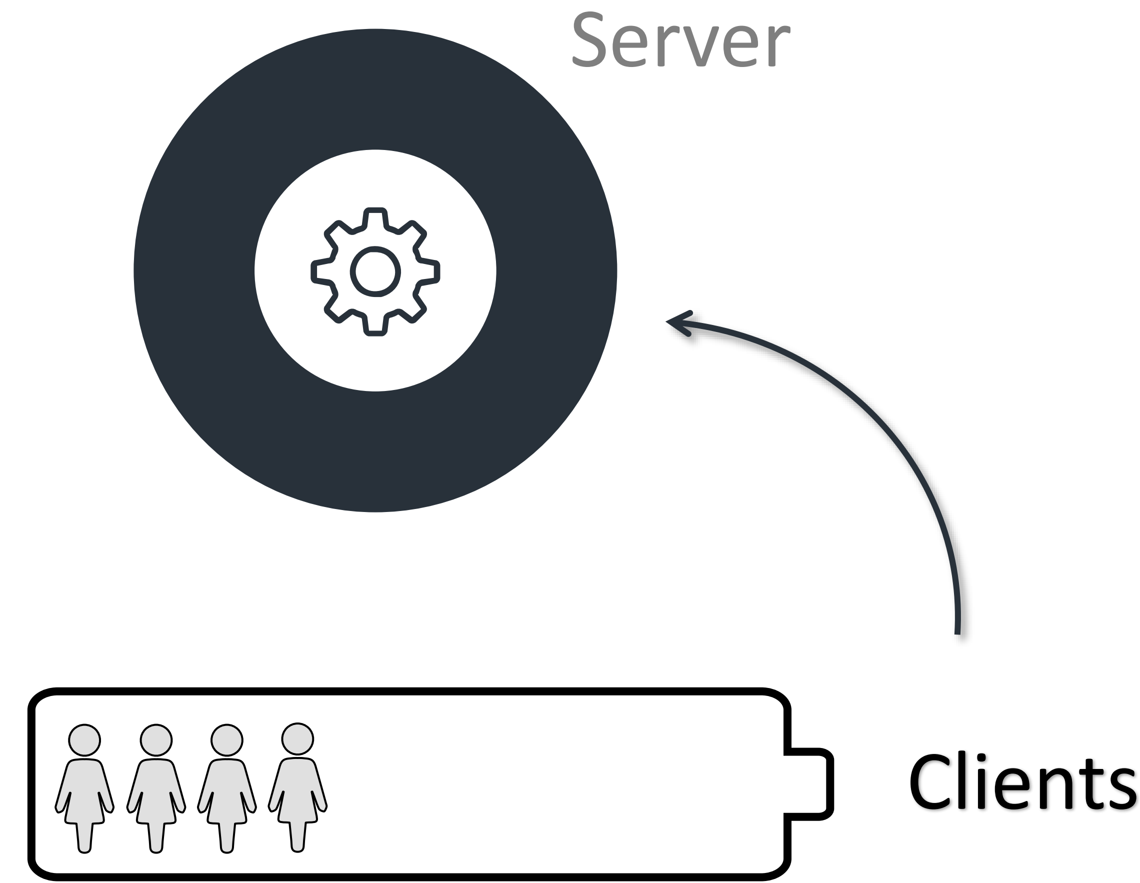
Implementation-Client



Implementation-Client



Implementation-Client



What client can do:

- Join the specified group
- Exit the current group
- Get the current group ID
- Connect with other clients for private chat
- Have private chat

Implementation-Client

```
...
r = recv(clientSocket[idx], buff, 1023, NULL);
if (r > 0) {
    buff[r] = 0;

    //if client is asking for some functions:
    if(buff[0] == '*') {
        std::vector<std::string> split_list;
        Stringsplit(buff, ' ', split_list);
        clientRequest(split_list,idx);
    }
    //if client send a common message:
    else {
        sprintf_s(msg, "%d:%s", idx, buff);
        printf("%s\n", msg);
        // Broadcast the data
        broadcast(idx, msg);
    }
}
```

Asterisk: ”*”

Special request

Common message

Implementation-Client

1. Judge which function it ask for

```
//1.*enter
if (cmd_str[0] == "*enter"){
//2.*exit
else if (cmd_str[0] == "*exit") {
//3. *groupinfo
else if(cmd_str[0] == "*groupinfo") {
//4. *private chat connect
else if(cmd_str[0] == "*connect") {
// *private chat
else if(cmd_str[0] == "*chat") {
//5.special error
else {
```

2. Change the needed string into integer

```
int user = atoi(cmd_str[1].c_str());/
int group = atoi(cmd_str[2].c_str());
```

- *enter as an example

Implementation-Client

*enter:

1. add(user, group)
2. Broadcast prompt message

```
addUser(user, group);
//give message to the new group
char msg[1024];
sprintf_s(msg, "%d joined the group! Welcome!", user);
broadcast(user, msg);
```

*exit:

1. Broadcast prompt message
2. Erase it from the map

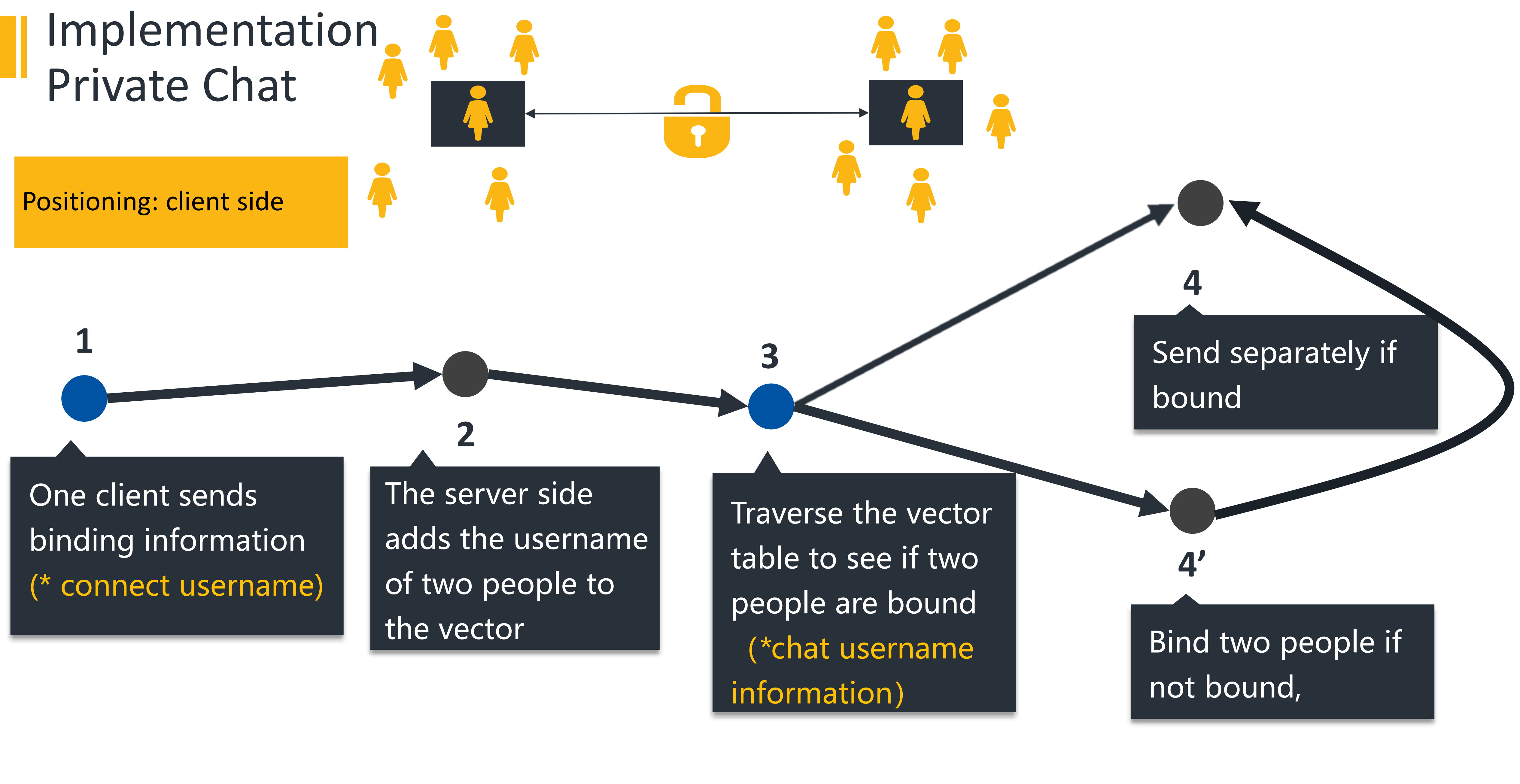
```
//give message in the former group
char msg[1024];
sprintf_s(msg, "%d left the group! Bye!", user);
broadcast(user, msg);
std::map<int, int>::iterator it = userManager.find(user);
if (it != userManager.end()){
    userManager.erase(it);
}
```

Implementation-Client

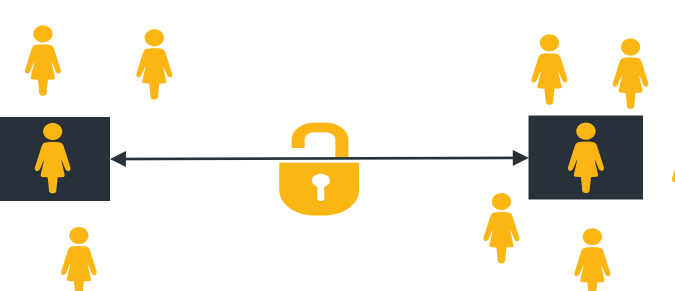
*groupinfo:

1. Search it in the map
2. Use its information of group id in map
3. Send private message to the client which ask for it

```
std::map<int, int>::iterator it_user = userManager.find(user);
char msg[1024];
sprintf_s(msg, "You are in group %d", it_user->second);
send(clientSocket[it_user->first], msg, strlen(msg), NULL);
```


Implementation
Private Chat
Positioning: client side
1
One client sends binding information
(* connect username)
2
The server side adds the username of two people to the vector
3
Traverse the vector table to see if two people are bound
(*chat username information)
4
Send separately if bound
4’
Bind two people if not bound,

Implementation Private Chat



● Realize the privacy of private chat

The information sent separately can only be seen by the private chat parties and the server

Advantage

● Use the vector（with the struct stored the user private chat information）

```
struct PriveChatUserInfo
{
    int user1;
    int user2;
};

std::vector<PriveChatUserInfo> priveChatVec;
```

● The redundancy of repeated two-way binding is avoid

```
void privateChat(int idx,int target, const char* buff) {
    bool is_exist = false;
    char msg[1024] = {0};

    for(std::vector<PriveChatUserInfo>::iterator it = priveChatVec.begin();it != priveChatVec.end();++it)
    {
        if((it->user1 == idx && it->user2 == target)||
            ((it->user1 == target && it->user2 == idx)))
        {
            is_exist = true;

            sprintf_s(msg, "%d(%d with %d):%s", idx, target, idx, buff);
            send(clientSocket[it->user1], msg, strlen(msg), NULL);
            send(clientSocket[it->user2], msg, strlen(msg), NULL);
        }
    }
```

Contents

Part 1 ： Introduction

Part 2 ： Data Structure

Part 3 ： Implementation

Part 4 ： Demonstration