Chương 10

Xây dựng ứng dụng mạng : phần Server

10.0 Dẫn nhập

10.1 Tổng quát về lập trình mạng trên Internet

10.2 Các bước hoạt động điển hình của ứng dụng server

10.3 Thí dụ về ứng dụng mạng cơ bản

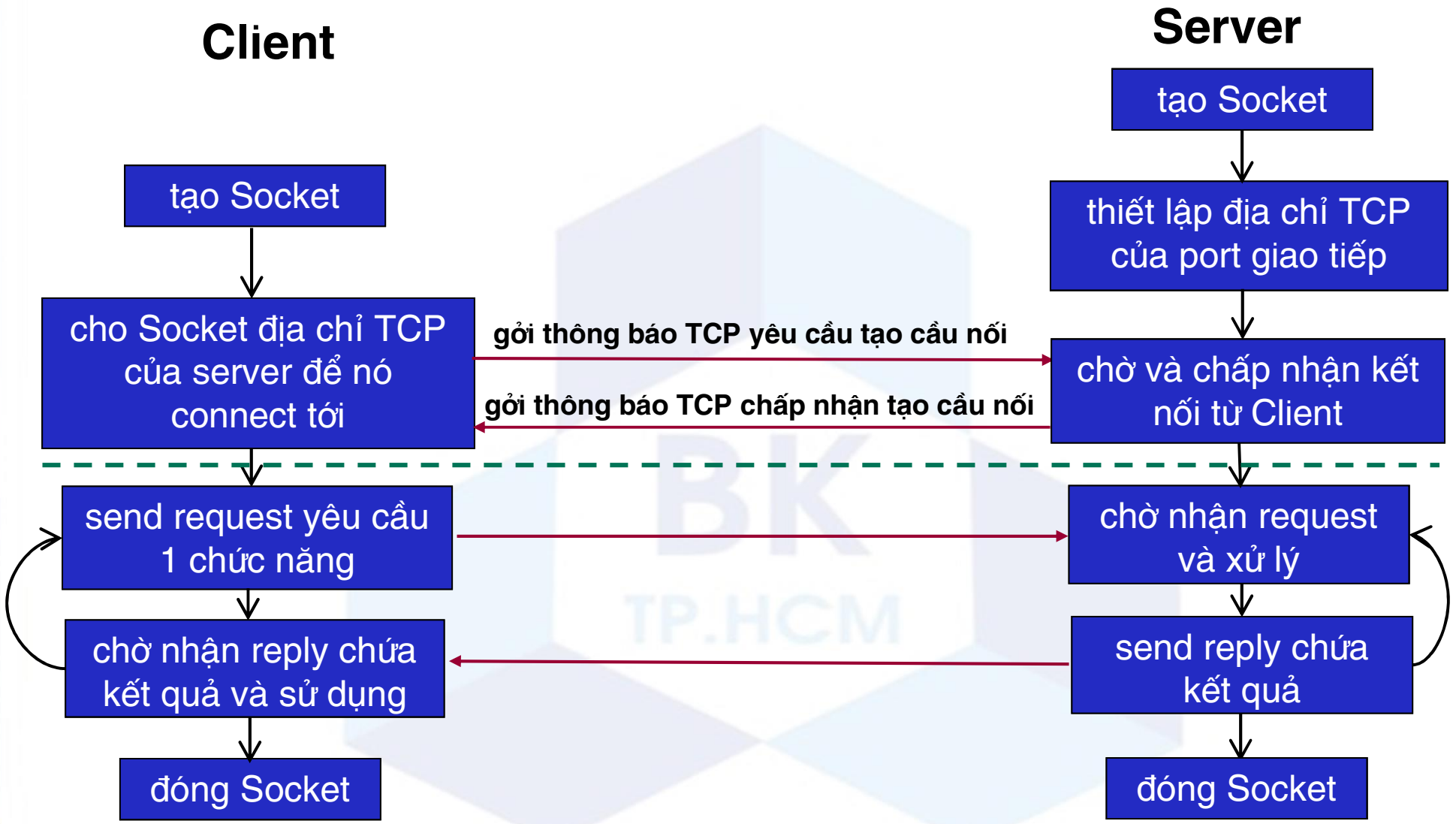
10.4 Các điểm chính về lập trình ứng dụng server

10.5 Kết chương

10.0 Dẫn nhập

- Chương này sẽ trình bày cơ chế hoạt động của module client và server để chúng giao tiếp tốt với nhau.
- Chương này cũng sẽ giới thiệu qui trình làm việc điển hình của module server và cách lập trình miêu tả từng bước hoạt động này.
- Chương này cũng sẽ giới thiệu cách xây dựng module server của ứng dụng Minichat cùng các đoạn code thực hiện các công việc chính yếu của module server này.

10.1 Tổng quát về lập trình mạng trên Internet



10.2 Các bước hoạt động điển hình của ứng dụng server

- Theo mô hình giao tiếp giữa module client và module server ở slide trước thì module server cần thực hiện các thao tác sau để chờ đợi yêu cầu kết nối tờ nhiều client và phục vụ đồng thời họ :
  1. tạo đối tượng Socket để quản lý port chờ yêu cầu kết nối từ các client
     ServerSocket Ssocket = new ServerSocket ();
  2. thiết lập địa chỉ TCP của server để các client biết và yêu cầu nối kết đến server. Thiết lập khả năng phục vụ đồng thời của server (số client max có thể phục vụ đồng thời). Thường trong lập trình Java, ta thực hiện bước 2 này luôn trong bước 1 :
     Ssocket = new ServerSocket(256, 100);
     Câu lệnh trên sẽ tạo đối tượng ServerSocket, thiết lập địa chỉ TCP của server là IPCur:256, trong đó IPCur là địa chỉ máy hiện hành chạy server.

10.2 Các bước hoạt động điển hình của ứng dụng server

3. gọi tác vụ accept() của đối tượng ServerSocket để chờ và chấp nhận yêu cầu kết nối từ client :

```
//chờ và chấp nhận yêu cầu kết nối từ client
cSocket[i] = Ssocket.accept();
```

4. Lặp lại bước 3 để chờ và chấp nhận yêu cầu kết nối từ client khác. Mỗi lần có 1 client yêu cầu kết nối, tác vụ accept() sẽ hoàn tất và 1 đối tượng Socket mới được tạo ra, ta lưu tham khảo này vào phần tử thứ i trong danh sách các Socket client đang phục vụ. Mỗi lần cần giao tiếp với client nào, ta dùng Socket tương ứng của client đó trong danh sách.

10.2 Các bước hoạt động điển hình của ứng dụng server

5. tạo các đối tượng quản lý gởi/nhận dữ liệu tới/từ client. Đối tượng Socket chứa 2 đối tượng gởi/nhận dữ liệu : OutputStream để gởi dữ liệu theo cơ chế từng byte nhị phân, InputStream để nhận dữ liệu theo cơ chế từng byte nhị phân (xem lại Chương 4). Mỗi thông báo request/reply thường là 1 dòng văn bản (được kết thúc bởi ký tự CRLF), để thuận lợi trong việc nhận/gởi từng dòng văn bản, ta thường tạo 2 đối tượng tương ứng như sau :

```
//tạo đối tượng hỗ trợ việc gởi từng dòng văn bản (reply)
PrintWriter writer = new PrintWriter(cSocket[i].getOutputStream());
//tạo đối tượng hỗ trợ việc nhận từng dòng văn bản (request)
cSocket[i].setSoTimeout(5000);
input = new BufferedReader(new InputStreamReader(
        cSocket[i].getInputStream()));
```

10.2 Các bước hoạt động điển hình của ứng dụng server

6. Khi cần ngắt kết nối với client nào, ta gọi tác vụ close() trên Socket tương ứng với client đó

```
//ngắt kết nối với client i trong danh sách
cSocket[i].close();
```

10.2 Các bước hoạt động điển hình của ứng dụng server

- Trong bước 3 ở slide trước, lệnh gọi sSocket.accept() có thể bị kẹt lâu dài (tùy thuộc vào thời điểm client yêu cầu nối kết đến). Nếu lệnh này nằm trong chương trình cần tương tác trực tiếp với người dùng thì trong khoảng thời gian tác vụ accept() chạy, chương trình sẽ không thể tương tác với người dùng. Để khắc phục vấn đề này, ta thường tạo thread con chạy song hành với chương trình chính, thread con này chỉ có nhiệm vụ thực hiện tác vụ accept(), khi nào nhận được yêu cầu kết nối của client nào đó thì nó sẽ cảnh báo cho chương trình biết để xử lý (bằng cách gởi thông điệp kích hoạt 1 tác vụ xác định trước của chương trình chính để tác vụ đó xử lý yêu cầu kết nối vừa nhận được).

10.2 Các bước hoạt động điển hình của ứng dụng server

- Trong việc giao tiếp với client nào đó, lệnh gởi reply về client không có gì đặc biệt, nhưng lệnh chờ nhận request có thể bị kẹt lâu dài (tùy thuộc vào thời điểm client đó gởi request). Nếu lệnh này nằm trong chương trình cần tương tác trực tiếp với người dùng thì trong khoảng thời gian chờ nhận request, chương trình sẽ không thể tương tác với người dùng. Để khắc phục vấn đề này, ta thường tạo thread con chạy song hành với chương trình chính, thread con này chỉ có nhiệm vụ thực hiện việc chờ nhận request, khi nào nhận được request thì nó sẽ cảnh báo cho chương trình biết để xử lý (bằng cách gởi thông điệp kích hoạt 1 tác vụ xác định trước của chương trình chính để tác vụ đó xử lý request vừa nhận được).

10.3 Thí dụ về ứng dụng mạng cơ bản

- Để thấy rõ chi tiết lập trình 1 server mạng, ta thử xây dựng server của hệ thống MiniChatter với 1 số tính chất sau :
  - Chức năng: cho phép nhiều user đăng ký vào các nhóm để trò chuyện với nhau.
  - Mô hình chọn lựa : client/server
  - Server: quản lý các nhóm và các user thuộc từng nhóm, phân phối các chuỗi thông tin từ một user đến các user khác cùng nhóm...
  - Client: giao tiếp với user, cho phép họ đăng ký nhóm, gởi/nhận thông tin lẫn nhau.

10.3 Thí dụ về ứng dụng mạng cơ bản

- Sau khi phân tích chức năng của chương trình chat, ta định nghĩa giao thức được dùng bởi hệ thống MiniChatter gồm 5 thông báo request sau :
  1. Lệnh GLIST <LF>
     Nhận danh sách các nhóm đang được server quản lý
  2. Lệnh ULIST <LF>
     Nhận danh sách user thuộc nhóm mà mình đang ở
  3. Lệnh LOGIN <tên group> "," <tên user> <LF>
     Đăng ký thành viên mới vào nhóm xác định
  4. Lệnh SEND <string> <LF>
     Gởi 1 dòng dữ liệu đến các user trong cùng nhóm
  5. Lệnh LOGOU <LF>
     Logout khỏi nhóm hiện hành (để đăng ký vào nhóm khác)

10.3 Thí dụ về ứng dụng mạng cơ bản

Và định dạng thông báo reply cho tất cả các request :

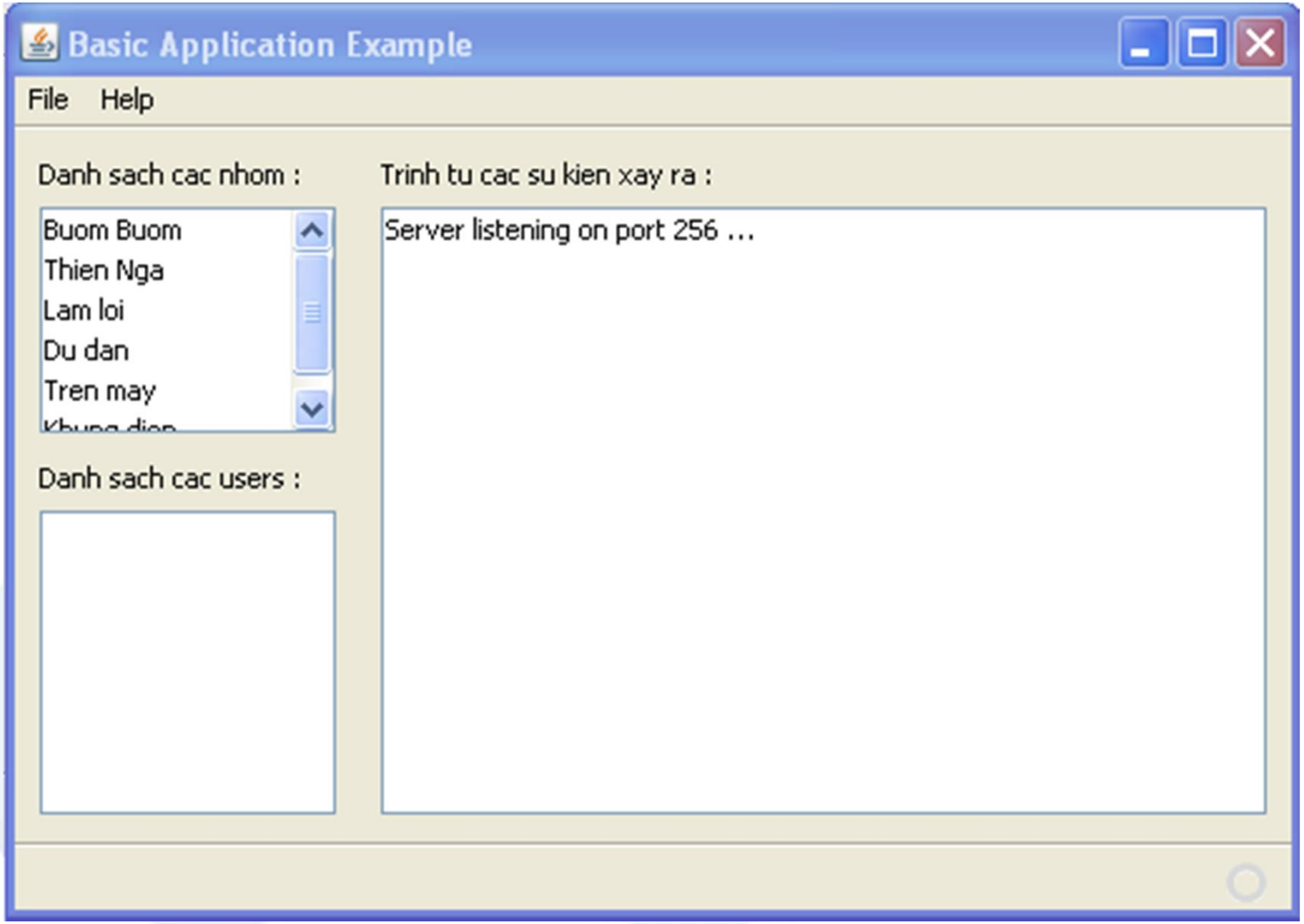
- n <chuỗi dữ liệu phụ trợ kèm theo> <LF>

  với n = 1 : thành công, n = 0 : thất bại.

- Bài thực hành số 10.1 sẽ trình bày qui trình viết module MiniChatter server bằng NetBeans và dùng kỹ thuật xử lý multi-thread để giúp module server có thể tương tác với người dùng trong khi hoặc chờ yêu cầu kết nối của client hoặc chờ nhận request của từng client.
- Chương trình server gồm 1 form giao diện chính cho phép admin giám sát trạng thái hoạt động của server, danh sách các user thuộc từng nhóm chat tại từng thời điểm.

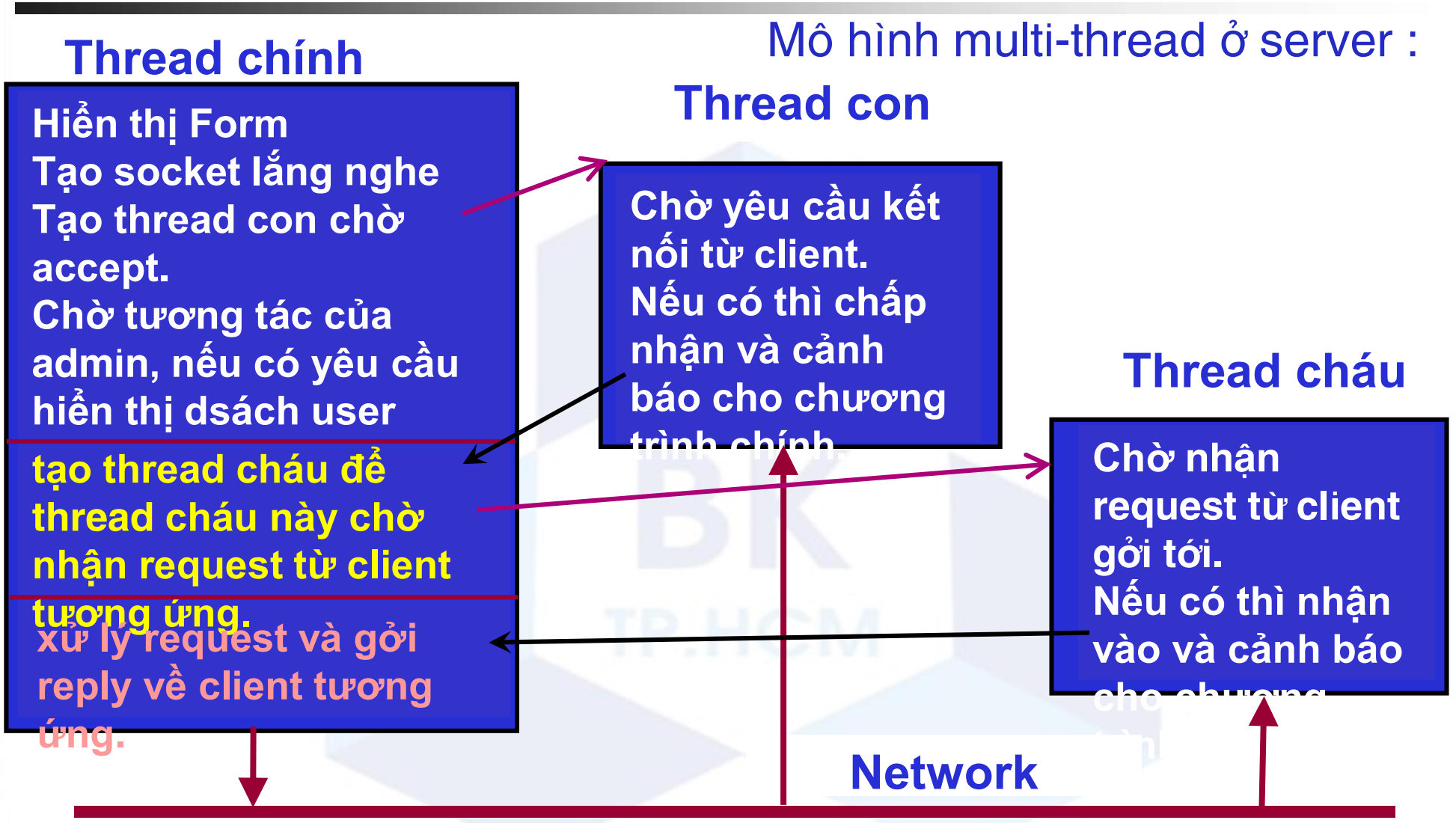
10.3 Thí dụ về ứng dụng mạng cơ bản

Form giao diện chính.



10.3 Thí dụ về ứng dụng mạng cơ bản

Mô hình multi-thread ở server :



10.4 Các điểm chính về lập trình ứng dụng server

Cấu trúc dữ liệu quản lý nhóm và các user trong từng nhóm :

```
//class miêu tả 1 nhóm
public class T_GroupList {
   String name;              //tên nhóm
   T_UserRec userlist;       //tham khảo đến user đầu trong danh
sách
}
//class miêu tả 1 user
public class T_UserRec {
    String name;             //tên user
    Socket sock;             //socket dùng để giao tiếp với user
    T_UserRec next;          //tham khảo đến user kế trong danh sách
}
```

10.4 Các điểm chính về lập trình ứng dụng server

Tạo sersersocket lắng nghe cho server :

```
try {
    serverSocket = new ServerSocket(SERVER_PORT, 100);
    //xuất thông báo lắng nghe lên Listbox
    DefaultListModel lmContent =
        (DefaultListModel) lbContent.getModel();
    lmContent.addElement("Server listening on port " +
SERVER_PORT + " ...");
    //tạo và chạy thread con chuyên chờ accept từ các client
    new ServerAcceptThread(this, serverSocket).start();
}
catch (IOException ioException) {
    //xử lý lỗi Exception
    ioException.printStackTrace();
}
```

10.4 Các điểm chính về lập trình ứng dụng server

Đọc thông tin các nhóm từ database và hiển thị lên Listbox :

```
private void ReadDisplayGroups() {
    //định nghĩa các biến cần dùng
    String conStr = "jdbc:odbc:GroupList";
    Connection con;
    String newSQL = "Select * from GroupList";
    String[] data = {"dummy"};
    DefaultListModel lmGroups = (DefaultListModel)
lbGroups.getModel();
    try {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        //1. Tạo connection miêu tả database cần truy xuất
        con = DriverManager.getConnection(conStr, "", "");
        //2. Tạo 1 đối tượng Statement liên kết đến connection
        java.sql.Statement stmt = con.createStatement();
```

10.4 Các điểm chính về lập trình ứng dụng server

```
//3. Tạo đối tượng recordset chứa kết quả của lệnh SQL
ResultSet rs = stmt.executeQuery(newSQL);
int i = 0;
lmGroups.clear(); //xóa nội dung cũ của ListBox
if (rs != null) {
//4. Duyệt recordset để xử lý các record của nó
    while (rs.next()) {  //khi còn record dữ liệu
        //tạo đối tượng quản lý nhóm tương ứng
        m_grouplist[i] = new T_GroupList();
        m_grouplist[i].name = rs.getString("groupname");
        lmGroups.addElement(m_grouplist[i].name);
        i++;
    }
}
```

10.4 Các điểm chính về lập trình ứng dụng server

```
        //ghi nhận số lượng nhóm cần quản lý
        m_groupcnt = i;
        //5. Đóng các đối tượng đã dùng lại
        rs.close(); stmt.close(); con.close();
    } catch (Exception e) {
        //Xử lý lỗi Exception
        System.out.println("Error : " + e);
    }
}
```

10.4 Các điểm chính về lập trình ứng dụng server

Thread chuyên chờ nhận và xử lý yêu cầu nối kết từ các client :

```
public class ServerAcceptThread extends Thread {
    ServerSocket serverSocket;
    MiniChatServerView serverChat;
    //hàm khởi tạo thread
    public ServerAcceptThread (MiniChatServerView server,
ServerSocket sock) {
        serverSocket = sock;  //lưu giữ socket sẽ chờ nhận yêu cầu
        serverChat = server;   //lưu giữ khách hàng nhờ
    }
    //hàm đặc tả thuật giải hoạt động của thread
    public void run() {
        T_UserRec puser;
```

10.4 Các điểm chính về lập trình ứng dụng server

```
try {
    // listen for clients constantly
    while (true) { //lặp chờ và xử lý từng yêu cầu nối kết
        //chờ 1 yêu cầu nối kết
        Socket clientSocket = serverSocket.accept();
        //tạo record đặc tả thông tin khách hàng
        puser = new T_UserRec();
        //chứa các thông tin khách hàng vào record
        puser.sock = clientSocket;
        //chèn record khách hàng mới vào đầu danh sách
        puser.next = serverChat.m_sock_no_user;
        serverChat.m_sock_no_user = puser;
        //tạo và chạy thread chờ nhận các request của socket này
        new ReceivingThread(serverChat, clientSocket).start();
```

10.4 Các điểm chính về lập trình ứng dụng server

```
            //hiển thị dòng thông báo về nối kết vừa xử lý
            DefaultListModel lmContent =
                  (DefaultListModel) serverChat.lbContent.getModel();
            lmContent.addElement("Connection received from: "
                  + clientSocket.getInetAddress());
            //gởi reply về nối kết vừa xử lý
            serverChat.SendMessage(clientSocket, "Request
accepted");
         } // end while
      } catch (Exception e) { e.printStackTrace(); }
   }
}
```

10.4 Các điểm chính về lập trình ứng dụng server

Lập trình thread chuyên chờ nhận reply :

```
//class miêu tả thread chuyên chờ nhận reply
public class ReceivingThread extends Thread {
    //định nghĩa các thuộc tính cần dùng
    private MessageListener messageListener;
    Socket socket;
    private BufferedReader input;
    private boolean keepListening = true;
    //định nghĩa hàm khởi tạo class
    public ReceivingThread(MessageListener listener, Socket clientSocket) {
        //kích hoạt cha chạy trước
        super("ReceivingThread: " + clientSocket);
        //ghi nhận khách hàng và Socket cần chờ
        messageListener = listener;
        socket = clientSocket;
```

10.4 Các điểm chính về lập trình ứng dụng server

```
        try {
            //thiết lập thời gian timeout cho socket
            socket.setSoTimeout(0);
            //tạo đối tượng nhận dữ liệu từ socket
            input = new BufferedReader (new InputStreamReader (
                    socket.getInputStream()));
        } catch (IOException ioException) { ioException.printStackTrace(); }
    }
```

10.4 Các điểm chính về lập trình ứng dụng server

```
//thuật giải hoạt động của thread chờ nhận dữ liệu
public void run() {
    String message;
    //lặp các hoạt động dưới đây cho đến khi hết yêu cầu
   while (keepListening) {
        try {
            //chờ nhận 1 dòng văn bản miêu tả reply được gởi đến
            message = input.readLine();
            //kiểm tra nếu chuỗi != rỗng thì gọi khách hàng xử lý
            if (message != null) //nếu != rỗng thì gọi khách hàng xử lý
                messageListener.messageReceived (socket, message);
        }
        catch (InterruptedIOException interruptedIOException) {
            continue; // tiếp tục chờ nhận reply
        }
```

10.4 Các điểm chính về lập trình ứng dụng server

```
            catch (IOException ioException) {
                keepListening = false; //yêu cầu kết thúc vòng lặp while
                //tạo reply đặc biệt và nhờ khách hàng xử lý
                messageListener.messageReceived (socket, "CLOSE ");
                break;
            }
        } //kết thúc vòng lặp while
        try {
            input.close(); //đóng đối tượng chờ nhận reply và đóng socket
        } catch (IOException ioException) {
            ioException.printStackTrace();
        }
    } //kết thúc hàm run
} //kết thúc class ReceivingThread
```

10.4 Các điểm chính về lập trình ứng dụng server

Lập trình phân tích và xử lý thông báo request :

```
public void messageReceived(Socket sock, String mesg) {
    int status;
    //xác định chuỗi đặc tả request
    String opcode = mesg.substring(0, 5);
    if (opcode.compareTo("LOGIN") == 0) { //nếu là request login
          Do_login(sock, mesg);
    } else if (opcode.compareTo("LOGOU") == 0) { //nếu là request logout
          Do_logout(sock);
    } else if (opcode.compareTo("GLIST") == 0) { //nếu là request GLIST
          Do_glist(sock);
    } else if (opcode.compareTo("ULIST") == 0) { //nếu là request ULIST
          Do_ulist(sock);
    } else if (opcode.compareTo("MESG") == 0) { //nếu là request MESG
           Do_MulticastMesg(sock, mesg.substring(5));
    }
}
```

10.4 Các điểm chính về lập trình ứng dụng server

Lập trình multicast 1 thông báo chat đến các thành viên của nhóm :

```
private void Do_MulticastMesg(Socket sock, String mesg) {
    //xác định chỉ số nhóm liên quan
    int i = Findgroup(sock);
    //nếu không có nhóm thì bỏ qua
    if (i < 0) return;
    //xây dựng chuỗi chat theo định dạng qui định : nickname : + nội dung
    mesg = uname + ":" + mesg;
    //xác định danh sách các user cùng nhóm
    T_UserRec pu = m_grouplist[i].userlist;
    //duyệt danh sách các user và gởi cho từng user
    while (pu != null) {
        SendMessage(pu.sock, mesg);
        pu = pu.next;
    }
}
```

10.4 Các điểm chính về lập trình ứng dụng server

Lập trình gởi thông báo (thường là reply) đến client :

```
//hàm gởi thông báo theo qui định của giao thức về client
public void SendMessage(Socket sock, String mesg) {
    try {
        //xác định đối tượng phục vụ gởi thông báo
        PrintWriter writer = new PrintWriter(sock.getOutputStream());
        //gởi thông báo
        writer.println(mesg);
        //yêu cầu hệ thống gởi ngay
        writer.flush();
    } catch (IOException ioException) {
            ioException.printStackTrace();
    }
}
```

10.5 Kết chương

- Chương này đã trình bày cơ chế hoạt động của module client và server để chúng giao tiếp tốt với nhau.
- Chương này cũng đã giới thiệu qui trình làm việc điển hình của module server và cách lập trình miêu tả từng bước hoạt động này.
- Chương này cũng đã giới thiệu cách xây dựng module server của ứng dụng Minichat cùng các đoạn code thực hiện các công việc chính yếu của module server này.