

# BÁO CÁO ĐỒ ÁN THỰC HÀNH CUỐI KÌ

---o0o---

## *XÂY DỰNG CHƯƠNG TRÌNH CHIA SẺ FILE : PEER -TO- PEER*



### Nhóm Thực Hiện :

SV1 : Huỳnh Phi Hùng

MSSV : 0812195.

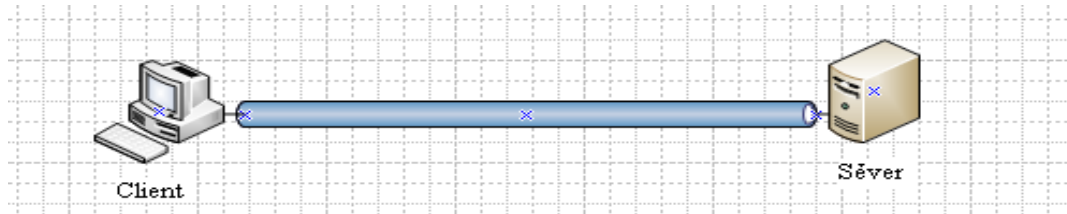
SV2 : Nguyễn Hoàng Đăng Khoa

MSSV : 0812231.

## **Mục Lục**

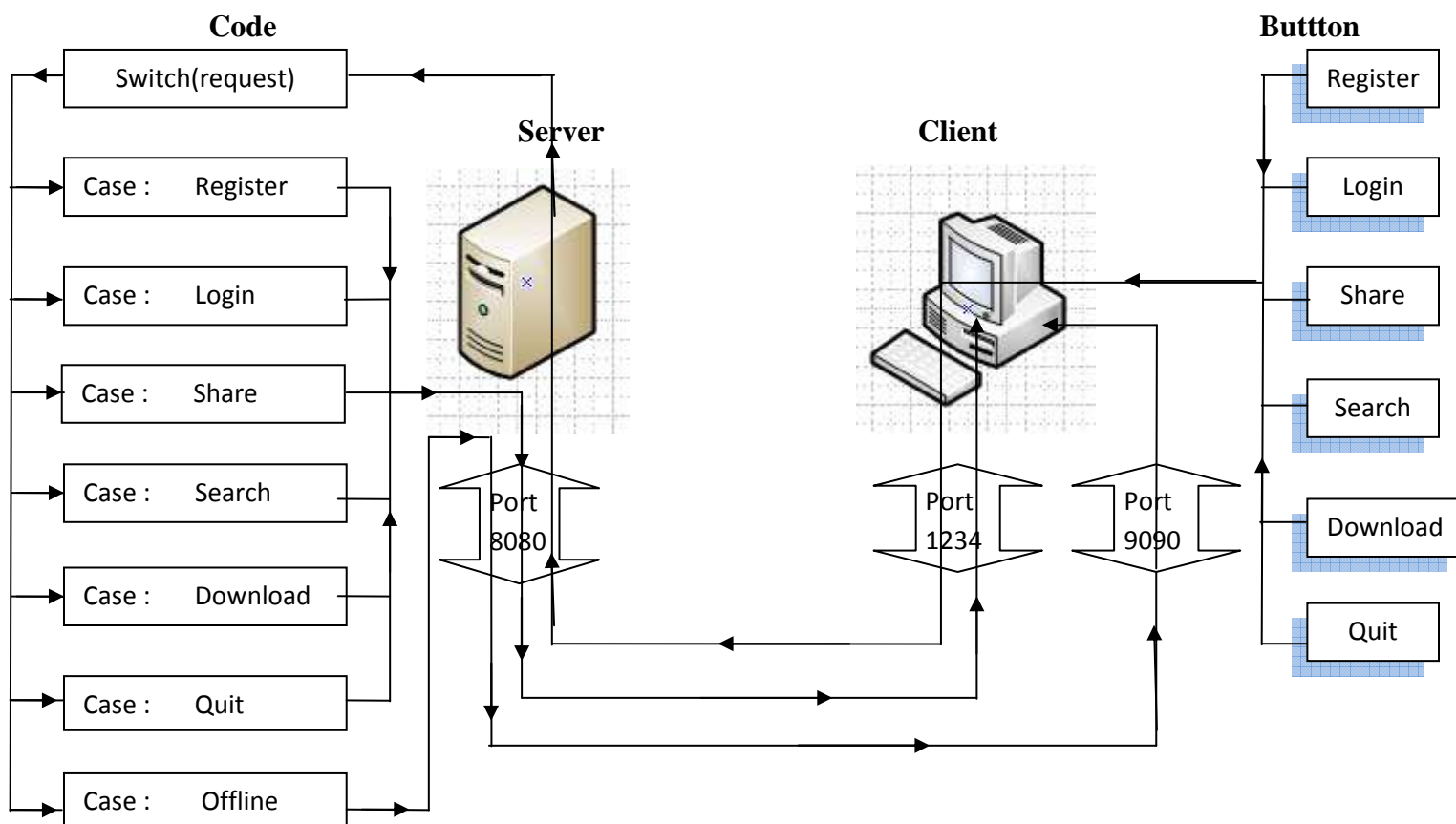
<b>A. MÔ HÌNH – THIẾT KẾ - CÀI ĐẶT :</b>	<b>4</b>
<b>I. Mô Hình Server-Client :</b>	<b>4</b>
<b>II. Mô Hình Client - Client :</b>	<b>5</b>
<b>III. Mô Hình Lưu Trữ Dữ Liệu :</b>	<b>7</b>
1. Khái quát chung về mô hình cài đặt dữ liệu :	7
2. Lưu Trữ Trên Server:	7
3. Lưu Trữ Trên Client:	8
<b>B. KỊCH BẢN TRAO ĐỔI DỮ LIỆU SERVER – CLIENT :</b>	<b>9</b>
<b>I. Kịch Bản Tiến Hành Đăng Nhập :</b>	<b>9</b>
<b>III. Kịch Bản Quá trình Chia Sẻ file:</b>	<b>11</b>
<b>IV. Kịch Bản Quá trình Tìm Kiếm file:</b>	<b>11</b>
<b>V. Kịch Bản quá trình Download :</b>	<b>13</b>
<b>VI. Kịch Bản quá trình Thoát:</b>	<b>15</b>
<b>C. ĐA TIỂU TRÌNH (Multi Thread):</b>	<b>16</b>
<b>I. Đa Tiểu Trình Trên Server :</b>	<b>16</b>
<b>II. Đa Tiểu Trình Trên Client :</b>	<b>17</b>
<b>D. ĐỒNG BỘ HÓA (Semaphore ):</b>	<b>18</b>
<b>I. Đặt vấn đề sử dụng Semaphore :</b>	<b>18</b>
<b>II. Cài Đặt Semaphore :</b>	<b>18</b>
a. Register:	18
b. Login:	19
c. Quit:	20
d. Share:	20
e. Download:	21
<b>E. LIÊN LẠC ĐA TIỂU TRÌNH:</b>	<b>22</b>
<b>I. Chia Sẻ Tài Nguyên (Static):</b>	<b>23</b>
<b>II. Trao Đổi Thông Tin :</b>	<b>23</b>
<b>F. XÂY DỰNG TESTCADE:</b>	<b>24</b>
<b>I. Đồng Bộ Hóa:</b>	<b>24</b>
<b>II. Đa Tiểu Trình:</b>	<b>25</b>
1. Đa Tiểu trình trên Server :	25

2.	Đa Tiêu trình trên Client :	25
III.	Liên Lạc Đa Tiêu Trình :	26
G.	BẢNG PHÂN CÔNG:	26
H.	NHẬN XÉT:	26
1.	Ưu điểm:	26
2.	Khuyết điểm:	26

**A. MÔ HÌNH – THIẾT KẾ - CÀI ĐẶT :****I. Mô Hình Server-Client :**

Mô Hình Server - Client là một mô hình xử lý tuần tự, tức là theo từng yêu cầu của bên Client khi gửi yêu cầu qua Server, khi Server nhận các yêu cầu từ Client và xử lý trả lời lại tuần tự các yêu cầu cho Client đó.

Do tính chất linh động về Port của chương trình cho người sử dụng nên trong chương trình, em sử dụng 2 Port trên Client (Port kết nối Server và Port lắng nghe Client khác kết nối đến), để người dùng có nhu cầu thay đổi được Port lắng nghe trên Client cho các kết nối từ máy tính khác đến máy mình Download. Còn về Server thì chỉ có 1 Port lắng nghe để thiết lập quản lý và đồng bộ quá trình Client Download File giữa các Client.

**Thiết Kế Mô Hình Server – Client :**

**Cài Đặt Mô Hình Server – Client :** Mô hình xử lý tuần tự giữa Server – Client được thể hiện qua thiết kế trên, mỗi yêu cầu của bên Client được xử lý bằng cách click vào nút Button ứng với mỗi yêu cầu, khi mỗi yêu cầu đó được xử lý bên Server và trả lời lại Client thì các yêu cầu khác mới có thể thực hiện tiếp tục .

**Download offline (Offline) :** là một dạng download khi Client share file không có trên mạng, yêu cầu này được lưu vào mảng `T_downloadWait[i]` cùng với `DataWait[i][3]` là dữ liệu lưu trữ cho Thread này, nó là một thuộc tính của form server, các Thread Client chia sẻ dễ dàng truy cập

Khi Client offline bắt đầu vào mạng thì Server báo hiệu các Client download các file của Client offline qua port lắng nghe của nó và khi nhận được tín hiệu thì các Client down liền kết nối tới Client kia nhưng theo tuần tự từng Client download xong thì sẽ giải phóng biến Semaphore.

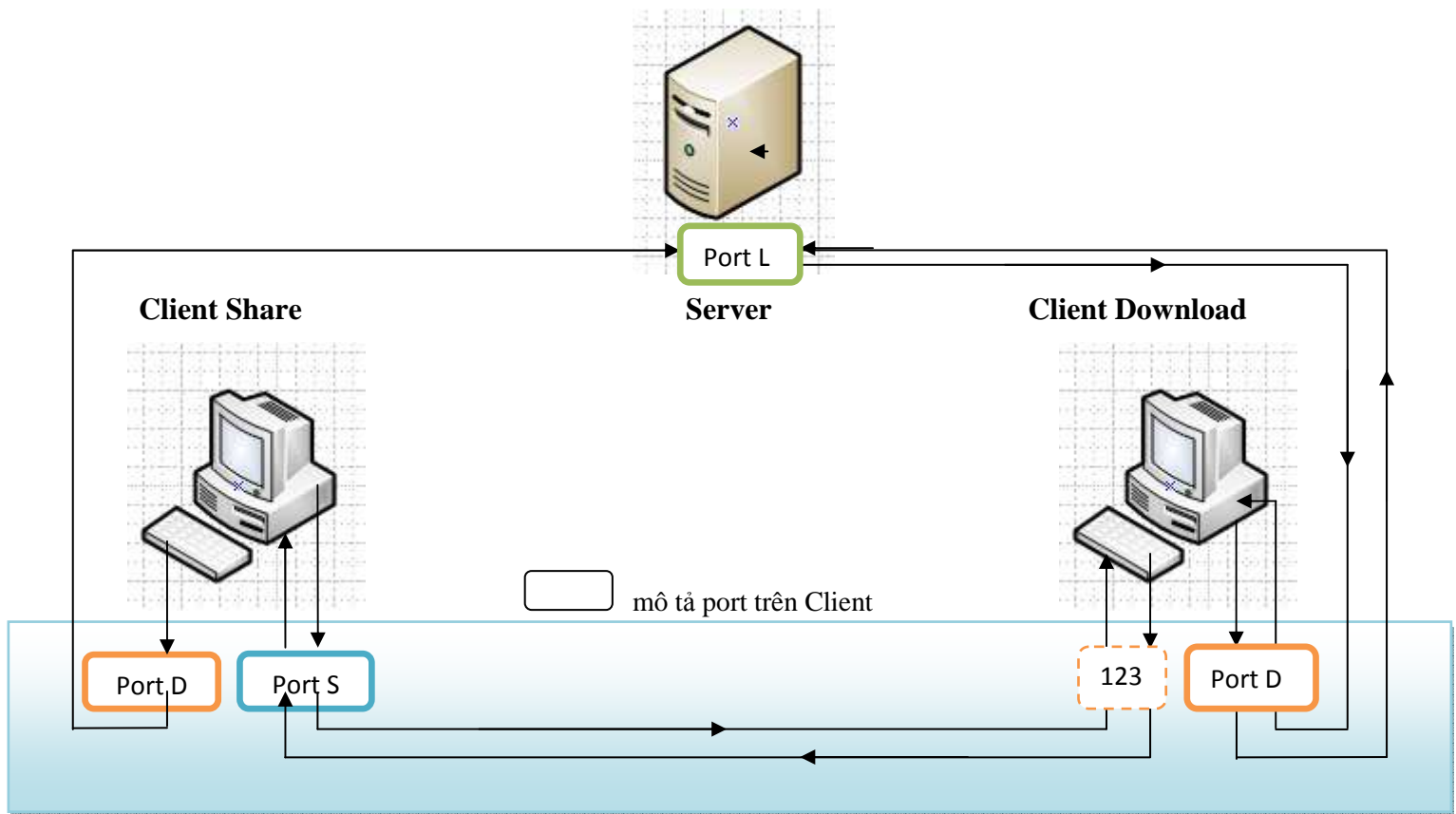
Khi nó được giải phóng thì các yêu cầu sau đó mới bắt đầu được Server gửi lại Client download offline đó chuỗi định dạng như trên.

Cách gửi dữ liệu và cách xử lý được nói rõ hơn trong phần **Kịch Bản Trao Đổi Server – Client**.

## II. **Mô Hình Client - Client :**

Mô Hình Client – Client là mô hình ngang hàng với nhau giữa Client và Client, mỗi bên đều có công việc như nhau là chia sẻ File và Download file. Việc Download giữa 2 Client đều o Server quản lý để đồng bộ quá trình chỉ download 1 file trên 1 Client chia sẻ.

**Thiết Kế Mô Hình Client – Client :**



Trên Client có 2 Port : **Port S** Port Lắng nghe và **Port D** Port kết nối đến Server , Client Share.

## Cài Đặt Mô Hình Client - Client :

- Khi Client Download gửi tín hiệu search lên Server để lấy dữ liệu trong Search File và tìm kiếm được các thông tin cần thiết sau đó thiết lập kết nối đến Client Share. Client Download sẽ connect đến Port đang lắng nghe trên Client Share đã cung cấp cho Server và gửi tên file.
- Client share chấp nhận kết nối và kiểm tra sự tồn tại của File mà Client Download yêu cầu. Nếu file tồn tại gửi tín hiệu “**Accept**” cho Client download :
- Hai phía bên mỗi Client đều gọi hàm thực thi quá trình gửi nhận dữ liệu ở thông qua Socket ở trên và thực hiện trong 2 class : ThreadListen và ThreadDownload.

```
//Bắt đầu Share dữ liệu :           //Bắt đầu Download dữ liệu :
this.sendFile(nameFile, this.S_listen);| this.result = ReceiveFile(Str_source,S_download);
```

- Khi truyền nhận dữ liệu có 2 tình huống xảy ra là :
  - + **Client Share file disconnect** : thì bên Client download gửi tín hiệu lên Server báo kết thúc download và hỏi người dùng có muốn download offline file này hay không ( ý nghĩa là resume download lại file này) ? nếu người dùng chấp nhận thì sẽ gửi chuỗi dạng download offline :

```
//Offline@nameFile@size@Username@IPofMe@PortofMe@null.
String Str_byDis = "Offline@" + Str_nameFile + "@@" + UserName + "@" + MeIP + "@" + MePort + "@null";
```

+ **Client Download file disconnect** : thì bên Client share thay Client download gửi tín hiệu kết thúc download file và Server giải phóng biến Semaphore đang đồng bộ quá trình download trên Server :

```
//Download@Finish@fileName@username@null.
String StrEn = "Download@Finish@" + MyName + "@" + Username + "@null";
byte[] byStrEn = System.Text.Encoding.ASCII.GetBytes(StrEn);
S_server.Send(byStrEn);
```

### III. Mô Hình Lưu Trữ Dữ Liệu :

1. **Khái quát chung về mô hình cài đặt dữ liệu** : Chương trình đã ứng dụng mô hình 3 lớp (three layers) : Presentation Layer, Business Logic Layer, Data Access Layer, để tiện trong quá trình lưu trữ và khai thác dữ liệu từ CSDL nguồn.

**Tại Server** : gồm 4 project nhỏ: Server, DTO, DAO, BUS. Trong đó DTO, DAO, BUS phục vụ cho các thao tác truy xuất vào CSDL bên dưới như Insert, Update, Delete dữ liệu khi cần và Server chứa form giao diện giúp giao tiếp với người dùng thì sử dụng các lớp có trong những project này để thực hiện các công việc như update thông tin, thêm/ xóa một đối tượng ví dụ như thêm Client, thông tin File chia sẻ.... Việc chia ra như vậy sẽ giúp dễ dàng quản lý code hơn, và có thể thêm bất kỳ một chức năng nào đó một cách dễ dàng.

**Tại Client**: chỉ có duy nhất 1 project là Client, nhưng trong đó ngoài class Client (Windows Forms Application) làm form giao diện giao tiếp với người dùng, thì thêm 3 class DTOInfor, DAOInfor, BUSInfor trong suốt đối với người dùng có chức năng tương tự như ở Server dùng để quản lý việc nhập/ xuất, sửa chữa thông tin về các File dữ liệu mà Client đã download.

#### 2. **Lưu Trữ Trên Server:**

Tập hợp các thông tin về Client, file chia sẻ được lưu trữ trong các table của CSDL sử dụng là Access theo cấu trúc như sau:

Thông tin Client:

CLIENT					
IDClient	Username	Password	IPAdr	NumPort	FlagSta

Trong đó:

- **IDClient**: mã của từng Client khi đăng kí với Server, được sinh ra theo kiểu Autonumber.
- **Username**: tên của Client đăng kí lên Server. Đây là tên duy nhất, mỗi Client phải có một Username khác nhau.
- **Password**: mật khẩu mà Client dùng để đăng nhập. Có phân biệt hoa thường.
- **IPAdr**: địa chỉ IP mà Client kết nối đến Server.
- **NumPort**: số port mà Client dùng để chia sẻ file.
- **FlagSta** : trạng thái online/offline của Client.

Thông tin Data :

DATA				
IDData	FileName	Size	IDClient	FlagSta

*Trong đó:*

- **IDData**: mã của từng file chia sẻ.
- **FileName**: tên file mà client muốn chia sẻ, giúp người dùng thuận tiện trong việc tra cứu và down file theo nhu cầu.
- **Size** : kích thước tương ứng của file nhằm sử dụng cho việc download file sau này khi cần so sánh kích thước file down được với kích thước file thực tế để có biện pháp giải quyết.
- **IDClient**: mã của Client chia sẻ file tương ứng.
- **FlagSta**: trạng thái file đang được down hay là không, nhằm dùng giải quyết vấn đề đồng bộ hóa sau này. Dựa vào đây để xác định được Client nào sẽ đưa vào hàng đợi và Client nào được phép down.

**3. Lưu Trữ Trên Client:**

Tương tự như Server, thông tin về file mà Client đã thực hiện việc download sẽ được cập nhật trong table Info, ở đây CSDL sử dụng cũng là Access. Có cấu trúc như sau:

INFO					
ID	FileName	Size	FromClient	PortClient	Flag

Dựa vào table này giúp ta có thể download lại những file trước đó đã down. Không cần gửi yêu cầu lên Server một lần nữa.

*Trong đó:*



- **ID**: số hiệu file đã download, tự động phát sinh.
- **FileName**: tên file đã download.
- **Size**: kích thước của file.
- **FromClient**: tên client chủ đã chia sẻ file tương ứng.
- **PortClient**: số port mà client chủ đã mở để chấp nhận kết nối và thực hiện việc cho phép các Client khác download.
- **Flag**: trạng thái file đã down hoàn tất hay là chưa.

**FromClient** và **PortClient** được lưu lại nhằm tái thiết lập lại kết nối nếu Client khách có nhu cầu muốn down lại.

→ Dữ liệu Share để trong thư mục : **C:\Share** và thư mục Downlaod về mặc định là **C:\Download**

## B. KỊCH BẢN TRAO ĐỔI DỮ LIỆU SERVER – CLIENT :

### I. Kịch Bản Tiến Hành Đăng Nhập :

SERVER	CLIENT
<ul style="list-style-type: none"> <li>- Tiến hành mở port lắng nghe tất cả các Client có nhu cầu kết nối đến Server.</li> <li>- Nhận được chuỗi thông tin, Server sẽ phân tích chuỗi bằng việc dựa vào ký tự “@” để tách từng từ đưa vào một mảng và dựa vào từ đầu tiên của chuỗi để thực hiện yêu cầu gì mà Client mong muốn. Ở đây, Client muốn “Login” nên Server sẽ tiến hành kiểm tra sự tồn tại của Client trong CSDL thông qua “ClientName” và “Password”. Nếu đúng sẽ cập nhật thông tin “NumPort” và “IPAddr”, trạng thái kết nối của Client bật lên là <b>true</b> trong CSDL, đồng thời gửi về</li> </ul>	<ul style="list-style-type: none"> <li>- Thiết lập kết nối đến Server thông qua port và Ip mà Server đã công bố.</li> <li>- Sau khi kết nối hoàn tất, gửi chuỗi thông tin: “Login@ClientName@Password@NumPort@IPAddr@null” lên Server để thực hiện quá trình login vào chương trình.</li> <li>- Khi nhận được chuỗi thông báo từ phía Server: “Login” hay “Bad”, Client sẽ</li> </ul>

Client chuỗi thông báo “Login” – Client đã đăng nhập thành công. Ngược lại, nếu sai sẽ không làm gì cả và gửi về Client chuỗi “Bad”_Client đã đăng nhập thất bại.	thông báo cho người dùng biết đăng nhập đã thành công hay là thất bại.  - Nếu thất bại thì thiết lập kết nối đó bị đóng và buộc Client muốn đăng nhập lại thì phải connect lại. Nếu thành công thì chuyển sang giao diện Download.
---	--

## II. Kịch Bản Tiến Hành Đăng Kí:

SERVER	CLIENT
<ul style="list-style-type: none"> <li>- Tiến hành mở port lắng nghe tất cả các Client có nhu cầu kết nối đến Server.</li> <li>- Nhận được chuỗi thông báo, Server sẽ phân tích và dựa vào từ đầu tiên của chuỗi để thao tác. Ở đây Client muốn “Register” nên Server sẽ tiến hành kiểm tra sự tồn tại của Client trong CSDL thông qua “ClientName” và “Password” mà Client gửi lên. Nếu thông tin Client cung cấp không tồn tại trong CSDL của Server, Server sẽ gửi về cho Client 1 chuỗi thông báo “Accept”, đồng thời cập nhật vào CSDL về thông tin Client đã cung cấp. Quá trình đăng kí của Client thành công, Ngược lại sẽ gửi về Client chuỗi “Cannot create Account” hoặc “Username has existed!”. Tức quá</li> </ul>	<ul style="list-style-type: none"> <li>- Thiết lập kết nối đến Server thông qua port và Ip mà Server đã công bố.</li> <li>- Sau khi kết nối hoàn tất, gửi chuỗi thông tin: “Register@ClientName@Password” lên Server để thực hiện quá trình register với Server.</li> </ul>

trình đăng kí thất bại. Và sau khi gửi thông báo Server sẽ ngắt kết nối với Client lúc này.	<ul style="list-style-type: none"><li>- Khi nhận được chuỗi thông báo từ phía Server: “<b>Accept</b>” Client sẽ thông báo cho người dùng biết đăng kí đã thành công. Và ngược lại sẽ thông báo thất bại.</li></ul>
---	--

**III. Kịch Bản Quá trình Chia Sẻ file:**

SERVER	CLIENT
<ul style="list-style-type: none"><li>- Nhận được chuỗi thông báo, phân tích và dựa vào từ đầu tiên của chuỗi để thực hiện. Server tiến hành cập nhật thông tin file chia sẻ khi nhận yêu cầu “<b>Share</b>”. Server sẽ kiểm tra các thông tin mà Client đưa lên “<b>FileName</b>” , “<b>FileSize</b>” và “<b>ClientName</b>”. Nếu các thông tin trên chưa có hoặc hợp lệ thì Sever sẽ cập nhật thông tin vào CSDL. Đồng thời gửi về chuỗi “<b>Next</b>” để yêu cầu Client tiếp tục gửi thông tin.</li></ul>	<ul style="list-style-type: none"><li>- Khi muốn đưa thông tin file chia sẻ lên Server, Client sẽ gửi một chuỗi với định dạng như sau : “<b>Share@FileName@FileSize@ClientName</b>” lên Server .</li><li>- Khi nhận được chuỗi “<b>Next</b>”, Client sẽ tiếp tục gửi thông tin file tiếp theo lên cho Server cho đến khi đã gửi hết toàn bộ thông tin về các file cho Server.</li></ul>

**IV. Kịch Bản Quá trình Tìm Kiếm file:**

SERVER	CLIENT
<ul style="list-style-type: none"> <li>- Dựa vào chuỗi nhận được, Server thực hiện nhiệm vụ “Search”. Đầu tiên, Server sẽ tiến hành kiểm tra tất cả các thông tin file có chứa chuỗi “StrSearch”, nếu tìm ra sẽ đưa vào một danh sách các file thỏa yêu cầu và loại bỏ những file nào do “ClientName” chia sẻ. Chỉ giữ lại những file do các Client khác cung cấp mà thôi. Sau đó gửi về cho Client chuỗi thông tin dạng “FileName@Size@IPAddr@NumPort@ClientName@Next” nếu như danh sách file mà Server tìm kiếm còn nhiều phần tử, và chuỗi “FileName@Size@IPAddr@NumPort@ClientName@End” nếu như là phần tử cuối cùng truyền đi.</li> <li>- Nếu không tìm thấy kết quả như mong muốn Server sẽ truyền về chuỗi “Null@Null@Null@Null@Null@Null” để thông báo không tìm thấy file như yêu cầu.</li> </ul>	<ul style="list-style-type: none"> <li>- Khi muốn tìm kiếm 1 thông tin về file mong muốn Client sẽ gửi lên cho Server chuỗi “Search@StrSearch@ClientName”</li> <li>- Khi nhận được chuỗi thông tin, Client cũng phân tích chuỗi thành từng thành phần và cập nhật thông tin của các file mà Client tìm kiếm, nếu nhận “Next” Client sẽ tiếp tục nhận thêm chuỗi mới từ Server, ngược lại nếu là “End” Client sẽ nhận biết là file cuối</li> </ul>

	<p>cùng và kết thúc vòng lặp nhận chuỗi thông tin từ Server.</p> <ul style="list-style-type: none"><li>- Nếu nhận được chuỗi đặc biệt “Null@Null@Null@Null@Null@Null” thì Client sẽ thông báo cho người dùng biết là không tìm thấy thông tin như yêu cầu.</li></ul>
--	--

**V. Kịch Bản quá trình Download :**

SERVER	CLIENT
<ul style="list-style-type: none"><li>- Tương tự như những phần trước Server sẽ phân tích chuỗi và kiểm tra tính hợp lệ của thông file mà Client có nhu cầu dowload. Ở đây Server sẽ thực hiện nhiệm vụ “Download”. Đầu tiên, Server sẽ kiểm tra xem “FileName”, “Size” có phù hợp với CSDL mà Server có hay không sau đó so sánh với thông tin “ClientChu” để biết được thông tin về “FileName” có thuộc về “ClientChu” hay không, cuối cùng kiểm tra trạng thái hiện tại của file là đang được sử dụng hay là không. Nếu tất cả đều phù hợp và kết quả kiểm tra trạng thái là false thì Server sẽ truyền về Client chuỗi “Start” để cho phép Client quá trình download của mình. Ngược lại, nếu kết quả kiểm tra là</li></ul>	<ul style="list-style-type: none"><li>- Khi có nhu cầu download một file sau khi tìm thấy Client sẽ gửi 1 chuỗi chứa thông tin file đó lên Server có dạng “Download@FileName@Size@ClientKhach@ClientChu@null”.</li></ul>

<p><b>true</b> tức file đang được sử dụng thì tiến trình này sẽ được đưa vào hàng đợi và truyền chuỗi “<b>Waiting</b>” về cho Client, cho đến khi file được giải phóng.</p>	<ul style="list-style-type: none"> <li>- Nếu nhận chuỗi “<b>Waiting</b>”, Client khách sẽ hiển thị thông báo cho người dùng biết phải đợi, nhưng có thể thực hiện việc download file khác trong khi đợi.</li> <li>- Khi nhận được chuỗi “<b>Start</b>” mà Server truyền về Client khách sẽ tiến hành kết nối với Client cho chia sẻ mà trước đó Client khách đã tìm thấy thông tin và gửi chuỗi thông tin “<b>Download@PathFile@null</b>” đến Client chủ. Tại đây Client chủ sẽ phân tích chuỗi và kiểm tra xem “<b>PathFile</b>” có tồn tại trong CSDL của mình hay không nếu có sẽ truyền chuỗi “<b>Accept</b>” bắt đầu cho Client khách download. Ngược lại, sẽ truyền chuỗi “<b>Close</b>” không cho download.</li> <li>- Sau khi download thành công Client khách sẽ hiển thị thông báo cho người dùng biết quá trình download thành công và kết thúc.</li> <li>- Khi quá trình Download đang diễn ra có thể xảy ra 2 trường hợp sau : <ul style="list-style-type: none"> <li>+ <b>Client share file bị disconnect</b> thì ngay lập tức Client download gửi thông báo kết thúc download file : <b>Download@finish@FileName@Username@null</b> và một thông báo Resum để Client có thể chọn download lại hoặc không, đó là dạng download offline (khi Client share online thì quá trình download bắt đầu) có</li> </ul> </li> </ul>
---	--

<p>- Khi Server nhận được chuỗi kết thúc: <b>Download@finish@FileName@UserName@null</b>. Download của Client thì nó sẽ giải phóng biến Semaphore để các Thread khác trong hàng đợi (Tức là các yêu cầu Download của các Client khác lên file này nhưng cùng một Client share file nên nó được tạo ra các List Data và Thread để lưu lại và đi vào trong hàng đợi của Semaphore) tiếp tục gửi các yêu cầu Download đến Client Share và được Server quản lý việc Download này một cách tuần tự, lần lượt theo chiến thuật điều phối FIFO.</p>	<p>dạng như :</p> <p><b>Offline@FileName@SizeFile@UserName@PortMe@IPMe@null</b></p> <p>.+<b>Client download file bị disconnect</b> thì ngay lập tức bên client share gửi thông báo kết thúc thay cho Client Download để báo Server biết đã download xong và giải thoát Thread nằm trong hàng đợi của semaphore:<b>Download@finish@FileName@UserName@null</b>.</p>
---	---

## VI. Kịch Bản quá trình Thoát:

SERVER	CLIENT
<p>- Nhận được chuỗi thông báo, phân tích và dựa vào từ đầu tiên của chuỗi để thực hiện. Trong trường hợp “Quit”, Server sẽ cập nhật thông tin trạng thái của</p>	<p>- Khi muốn thoát khỏi chương trình, Client sẽ gửi chuỗi “<b>Quit@ClientName@IPAddr</b>” lên Server .</p>

<p>Client có “<b>ClientName</b>” và “<b>IPAddr</b>” mà Client gửi lên. Nếu Server đồng ý cho thoát sẽ gửi về Client chuỗi thông báo “<b>Quit</b>”, cập nhật trạng thái lúc này của Client là <b>false</b>, ngược lại không làm gì cả và gửi chuỗi thông báo về cho Client là “<b>No</b>”.</p>	<ul style="list-style-type: none"> <li>- Khi nhận được chuỗi thông báo từ phía Server: “<b>Quit</b>” Client sẽ ngắt các kết nối, kết thúc phiên làm việc hiện tại, thoát khỏi chương trình. Nếu nhận được chuỗi “<b>No</b>”, Client không được phép làm gì cả.</li> </ul>
---	---

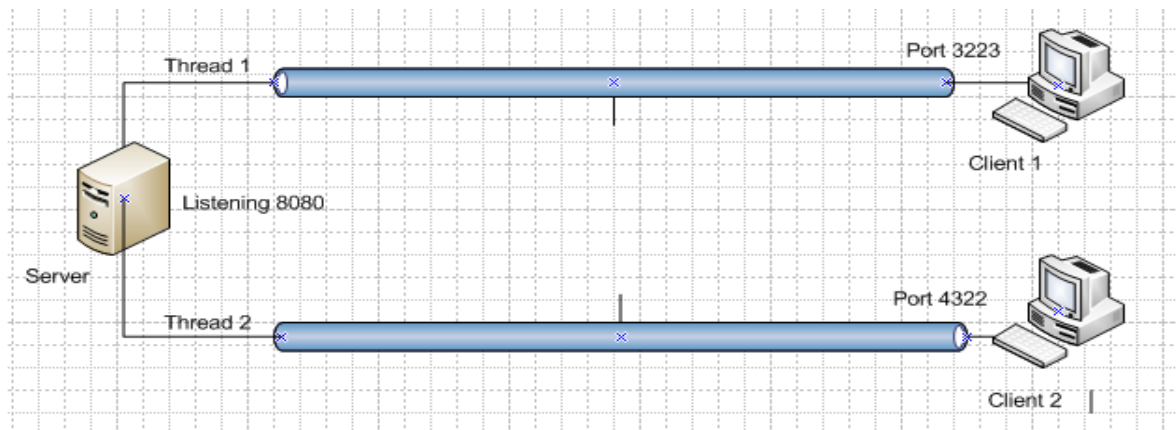
### C. ĐA TIỂU TRÌNH (Multi Thread):

#### I. Đa Tiểu Trình Trên Server :

Server phải xử lý các công việc đồng hành như lắng nghe các kết nối đến và xử lý các yêu cầu của mỗi Client. Để có thể xử lý đồng hành các công việc như trên thì Server phải có nhiều dòng xử lý cùng một lúc, nghĩa là một Server đa chương, đa tiểu trình giúp Server hoạt động .

- Trên Server để có thể lắng nghe một lúc nhiều kết nối của các Client, trước hết tạo thread lắng nghe các kết nối đến .
- Nhiệm vụ của thread này là lắng nghe các kết nối của Client đến và tạo ra các thread chạy cùng với Socket đó để trao đổi thông tin.
- Khi mỗi Client connect đến thì sẽ tạo một Thread chạy cùng với Socket mới tạo đó, sau khi được gọi chạy Start() thì Thread Client này sẽ **new ThreadClient(S\_sockWork, this)** . Class ThreadClient chứa các hàm xử lý trao đổi với Client, có chức năng lắng nghe các yêu cầu từ Client.
- Class này được mô tả cách hoạt động đầy đủ trong **Mô Hình Server – Client**, trong mô hình đó đã nêu đầy đủ chức năng của các trường hợp case trao đổi với Client.





## II. Đa Thread Trên Client :

Client có các công việc sau mà phải xử lý song song đó là :

- + Dùng kết nối ban đầu của Login để gửi các yêu cầu cho Server quá trình chia sẻ, download file .
- + Lắng nghe các kết nối đến từ các Client khác để nó download các file mà Client này chia sẻ, và khi Client có yêu cầu download các File từ các Client khác chia sẻ.

Vấn đề đặt ra là khi trao đổi thông tin với Server thì Client chỉ cần Thread main giao diện xử lý các Button, nhưng khi có yêu cầu download và share File thì Client không chỉ download và chia sẻ 1 file mà là nhiều file khác nhau, phụ thuộc vào yêu cầu người dùng. Do vậy quá trình đa thread trên Client được chia ra làm 2 bước xử lý như sau :

- a. **Tạo Thread lắng nghe các kết nối đến từ các Client khác :** để có thể tạo Socket lắng nghe các kết nối đến để yêu cầu download thì ngay khi Load form Download lên thì ta thực hiện như sau:
  - Khởi tạo Thread T\_clientListen chạy hàm void downloadListen(), mục đích là để thread này chạy lắng nghe các kết nối của các Client khác.
  - Sau đó tạo Socket với cổng lắng nghe được chọn để trạng thái chờ kết nối đến, mỗi khi có kết nối connect từ các Client khác thì tạo ra thread T\_threadListen. Sau khi tạo được Thread thì sẽ gọi hàm : void ThreadListen()
  - Khi Thread mới T\_threadListen chạy sẽ gọi hàm dựng new class ThreadListen và truyền tham số Socket vào class này để tiếp tục trao đổi thông tin của file :
- b. **Tạo Thread download khi có yêu cầu download file :** khi người dùng Click vào ButDownload thì Client gửi tín hiệu download file lên Server, khi đó Server sẽ kiểm tra xem file của Client đó có rảnh không, nếu có rảnh thì gửi yêu cầu tín hiệu bắt đầu thì Client bắt đầu thực hiện như sau :

- Khởi tạo Thread T\_clientDownload chạy hàm void connectClient(), mục đích là tạo Thread xử lý việc download file này, Thread này chạy hàm connectClient :
- Khi Thread T\_clientDownload chạy hàm connectClient, nó bắt đầu gọi hàm dựng new của class ThreadDownload để truyền các dữ liệu cần thiết cho Thread khi nó bắt đầu thực thi class ThreadDownload.

➔ Quá trình Download và Share File được nói rõ trong **Mô Hình Client – Client**.

## **D. ĐỒNG BỘ HÓA (Semaphore):**

### **I. Đặt vấn đề sử dụng Semaphore :**

Như đã biết việc nhiều tiểu trình cùng truy xuất vào một miền dùng chung (CS) nào đó sẽ gây ra xung đột dữ liệu, dẫn đến kết quả sẽ sai lệch hoặc chương trình treo.... Để giải quyết vấn đề này cần phải thực hiện việc đồng bộ việc truy xuất vào miền CS này nhằm đảm bảo sao cho các tiểu trình đều diễn ra theo đúng yêu cầu nhưng vẫn đảm bảo không có sự xung đột ở đây.

Ta có thể thực hiện việc đồng bộ ở Server hay Client, tùy vào ý đồ của người lập trình, nhưng vẫn đảm bảo cho chương trình hoạt động ổn định.

Cụ thể, chương trình sẽ tiến hành đồng bộ tại các bước từ khi Register, Login, Share, Download và Quit.

Trước hết, để giải quyết bài toán đồng bộ hóa, chương trình đã sử dụng những đối tượng Semaphore với khai báo tương tự có biến đếm Count khởi đầu là 1 và Max =1 nhằm xác định duy nhất 1 tiểu trình được phép truy xuất vào miền CS. Để đăng kí truy xuất tài nguyên dùng chung, hàm **WaitOne()** sẽ được gọi và tùy vào trạng thái của Semaphore mà tiểu trình có nhu cầu sẽ được thực hiện tiếp công việc của mình hay là vào hàng đợi. Khi tiểu trình đầu tiên truy xuất vào miền CS, biến Count = 1 > 0 và sẽ cho tiểu trình đó vào miền CS , đồng thời biến Count sẽ giảm đi 1 đơn vị (Count = 0). Những tiểu trình khác khi vào sẽ thấy rằng biến Count = 0 và Max = 1 nên sẽ bị đưa vào hàng đợi của đối tượng semaphore, đợi cho đến khi nào tiểu trình đang hoạt động gọi hàm **Release()**, trả lại miền CS và biến Count tăng lên 1 đơn vị (Count =1 ) thì những tiểu trình đang nằm trong hàng đợi sẽ được gọi dậy và tiếp tục công việc của mình theo chiến lược FIFO : tiểu trình nào ở đầu hàng đợi sẽ được gọi dậy đầu tiên bất kể có độ ưu tiên thấp hơn các tiểu trình vào sau đó.

### **II. Cài Đặt Semaphore :**

#### **a. Register:**

- Sử dụng 1 biến semaphore toàn cục :

```
static Semaphore S_updateClient = new Semaphore(1, 1);
```

- Đặt vấn đề: Khi có một yêu cầu gửi lên từ Client muốn đăng kí tài khoản trên Server, Server sẽ kiểm tra nếu hợp lệ sẽ tiến hành cập nhật thông tin của Client vào CSDL của mình. Chính tại đây, việc cập nhật thông tin này đã tạo ra 1 miền CS đó là CSDL mà Server dùng lưu trữ thông tin Client. Nếu nhiều Client cùng lúc yêu cầu thì Server sẽ cập nhật như thế nào?
- Giải quyết vấn đề: lúc này biến `S_updateClient` sẽ được dùng ngay khi quá trình cập nhật chuẩn bị được thực hiện. Cụ thể:

```
S_updateClient.WaitOne();  
bool rs = BUSClient.Insert(client);  
S_updateClient.Release();
```

`S_updateClient.WaitOne()` được gọi lên để khóa những tiểu trình khác nếu đã có 1 tiểu trình đang thực hiện việc cập nhật dữ liệu thông qua hàm `BUSClient.Insert(client)`, đây chính là miền CS mà chúng ta cần phải đồng bộ, sau khi hoàn thành việc Insert thì Client đang thực thi sẽ gọi hàm `S_updateClient.Release()` để trả lại miền CS và đánh thức những tiểu trình khác theo chiến lược FIFO.

**b. Login:**

- Tương tự như khi Register, ta sẽ dùng một biến Semaphore toàn cục khác để giải quyết:

```
static Semaphore S_updateClient = new Semaphore(1, 1);
```

- Đặt vấn đề: khi Client login, sẽ gửi thông báo lên cho Server và Server sẽ cập nhật thông tin trạng thái trong CSDL. Và tương tự như Register thì việc update lại thông tin trạng thái của từng Client cũng đang sử dụng chung một tài nguyên. Vậy cần phải đồng bộ ra sao?
- Giải quyết vấn đề: ta sẽ dùng biến semaphore `S_updateClient` kẹp vào phần miền CS đã nói trên.

```
S_updateClient.WaitOne();  
BUSClient.Update(client);  
S_updateClient.Release();
```

Cách thức hoạt động cũng tương tự như của Register. Tiểu trình đầu tiên sẽ gọi `S_updateClient.WaitOne()` để đăng kí sử dụng miền CS, đó là update vào CSDL của Server. Với cách thức hoạt động của semaphore đã trình bày ở trên thì những tiểu trình khác khi vào sẽ được đưa vào hàng đợi của đối tượng semaphore này cho đến khi tiểu trình hiện tại gọi hàm `S_updateClient.Release()` thì những tiểu trình hiện trong hàng đợi sẽ được gọi dậy và tiếp tục công việc của mình cũng theo chiến lược FIFO.

**c. Quit:**

- Tương tự như phần Login , hay Register ta sẽ dùng 2 biến semaphore để đồng bộ:

```
static Semaphore S_getClient = new Semaphore(1, 1);  
static Semaphore S_updateClient = new Semaphore(1, 1);
```

Biến **S\_getClient** sẽ giúp đồng bộ trong việc truy xuất vào dữ liệu để lấy danh sách Client và biến **S\_updateClient** giúp đồng bộ trong việc cập nhật lại danh sách đó trong CSDL.

- Đặt vấn đề: Khi các Client có nhu cầu thoát khỏi chương trình và báo lên Server để cập nhật lại trạng thái disconnect, thì Server sẽ tiến hành cập nhật trong CSDL của mình và vấn đề lại tiếp tục xảy ra tại đây. Nếu cùng lúc nhiều Client có nhu cầu như vậy thì sao?
- Giải quyết: Việc đồng bộ cũng sẽ được tiến hành như khi làm Login hay Register. Ở đây ta sẽ đồng bộ tại 2 vị trí:

- o Truy xuất vào CSDL để lấy dữ liệu.

```
S_getClient.WaitOne();  
List<DTOClient> listClient = BUSClient.getListClient();  
S_getClient.Release();
```

- o Update lại trạng thái của từng Client.

```
S_updateClient.WaitOne();  
BUSClient.Update(client);  
S_updateClient.Release();
```

Cách thức hoạt động của các semaphore ở 2 vị trí trên cũng tương tự như khi đồng bộ Login. Tiểu trình đầu tiên vào miền CS sẽ gọi **WaitOne()** để đăng kí sử dụng và những tiểu trình sau sẽ đưa vào hàng đợi cho đến khi tiểu trình đang hoạt động hiện tại gọi hàm **Release()** để giải phóng tài nguyên, đánh thức các tiểu trình khác.

**d. Share:**

- Quá trình này cũng cần đồng bộ khi muốn truy xuất lấy thông tin trong cùng CSDL và cập nhật thêm đối tượng vào bảng thông tin các file chia sẻ. Ở đây ta dùng 2 semaphore:

```
static Semaphore S_getClient = new Semaphore(1, 1);  
static Semaphore S_updateClient = new Semaphore(1, 1);
```

- Đặt vấn đề: Khi thông tin các file chia sẻ của các Client đưa lên Server thì việc cùng truy cập vào CSDL của Server sẽ xảy ra xung đột. Vì lúc này những thông tin này sẽ được cập nhật vào CSDL nhưng nếu như có nhiều Client cùng lúc đưa thông tin lên thì việc cập nhật thông tin đó sẽ như thế nào?
- Giải quyết: Biến **S\_getClient** được dùng để độc quyền trong việc lấy ra danh sách các Client từ CSDL của Server mà không gây ra việc tranh chấp giữa các tiểu trình sau này.

```
S_getClient.WaitOne();
List<DTOClient> listClient = BUSClient.getListClient();
S_getClient.Release();
```

Cách thức hoạt động của công việc này cũng tương tự như ở phần Quit. Tiếp theo sử dụng biến `S_updateClient` để giúp độc quyền trong vấn đề thêm thông tin của một đối tượng file chia sẻ vào CSDL.

```
S_updateClient.WaitOne();
BUSData.Insert(share);
S_updateClient.Release();
```

Vì việc thêm này sẽ gây ra xung đột hệ thống khi các Client có cùng nhu cầu tại cùng 1 thời điểm. Chúng ta phải đảm bảo sao cho tại 1 thời điểm nhất định chỉ có duy nhất một tiểu trình thực sự được sử dụng CSDL, khi tiểu trình đầu tiên truy cập vào CSDL sẽ gọi hàm `S_updateClient.WaitOne()` để khai báo muốn sử dụng tài nguyên và hoạt động cho đến khi gọi `S_updateClient.Release()` để giải phóng tài nguyên. Những tiểu trình khác vào sau bị đưa vào hàng đợi khi kiểm tra `S_updateClient.WaitOne()` và được đánh thức khi `S_updateClient.Release()` được gọi bởi tiểu trình hiện đang hoạt động.

#### e. Download:

- Vấn đề nảy giải quyết hơi phức tạp hơn vì các Client có nhu cầu down cùng nhiều file cùng một lúc thì sẽ xảy ra vấn đề tranh chấp tài nguyên dùng chung chính là những file dữ liệu mà Server đang lưu trữ. Do vậy, cứ một file ta sẽ sử dụng một biến semaphore để quản lý và đồng bộ. Ta sẽ tạo ra một mảng các semaphore và một mảng các tên tương ứng với từng semaphore đó để dễ quản lý từng semaphore mà ta đã tạo .

```
static List<Semaphore> listSemaphore = new List<Semaphore>();
static List<String> listNameSemaphore = new List<String>();
static Semaphore S_getClient = new Semaphore(1, 1);
static Semaphore S_updateClient = new Semaphore(1, 1);
```

- Đặt vấn đề: Khi các Client có nhu cầu download các file cùng một lúc thì việc sẽ xảy ra nếu như cùng một file mà các Client cùng down?
- Giải quyết: Trước tiên chương trình sẽ dùng biến semaphore `S_getClient` để đồng bộ trong việc truy xuất vào dữ liệu. Cách thức hoạt động đã được nói rõ ở những phần trên.

```
S_getClient.WaitOne();
List<DTOClient> listClient = BUSClient.getListClient();
List<DTOData> listData = BUSData.getListData();
S_getClient.Release();
```

Tiếp đến là dùng biến semaphore **S\_updateClient** để đồng bộ trong việc cập nhật thông tin dữ liệu vào CSDL. Ở đây chúng ta sẽ cập nhật lại trạng thái của file trước và sau khi hoàn tất quá trình download. Công việc cũng tiến hành như những phần trên.

```
S_updateClient.WaitOne();  
BUSData.Update(data);  
S_updateClient.Release();
```

Khi file đang ở trạng thái đang được sử dụng, những tiểu trình khác có nhu cầu sẽ tiến hành đưa những tiểu trình đó vào một hàng đợi riêng để đảm bảo cho những tiểu trình khác của những Client có tiểu trình đang đợi vẫn hoạt động một cách bình thường. Ở đây, ta sử dụng biến **i\_Semaphore** dùng để xác định đối tượng semaphore tương ứng của từng file trong danh sách các đối tượng semaphore mà ta đã dùng. Đây là biến toàn cục và được sử dụng chung cho mọi tiểu trình cho nên việc dùng chung sẽ không thể tránh khỏi, do đó ta sẽ tiến hành đồng bộ việc sử dụng biến này.

Cứ một tiểu trình có nhu cầu download một file đang hoạt động thì sẽ được tiến hành kiểm tra sự tồn tại của file đó trong danh sách các semaphore, nếu tồn tại tại đối tượng semaphore tương ứng thì sẽ thực hiện việc đưa tiểu trình đó vào hàng đợi như đã nói. Việc biến **i\_Semaphore** dùng chung nhằm đảm bảo cho tất cả các tiểu trình đều hoạt động một cách ổn định, tuần tự, không xảy ra quá trình xung đột khi có nhu cầu cùng download 1 file.

*Tóm lại toàn bộ quá trình Download sẽ diễn ra như sau: Đầu tiên chương trình sẽ kiểm tra sự tồn tại của đối tượng semaphore có tên tương ứng với tên mà client gửi lên hay không. Nếu đã có sẽ không tạo thêm một đối tượng semaphore mới, ngược lại sẽ tạo thêm một đối tượng vào **listSemaphore**. Tiếp đến nếu trạng thái file là **false**, semaphore tương ứng của file này sẽ gọi **WaitOne()**, gửi thông báo về cho tiểu trình đang muốn download và tiến hành cập nhật lại trạng thái cho file tương ứng (đồng bộ cập nhật trạng thái file). Ngược lại trạng thái file là **true** thì sẽ tiến hành đưa tiểu trình tương ứng (thông qua việc kiểm tra bởi biến chung **i\_Semaphore**: đồng bộ sử dụng tài nguyên chung) vào danh sách các hàng đợi tiểu trình. Khi có thông báo kết thúc từ Client, Server sẽ cập nhật lại trạng thái file lần nữa (tiếp tục đồng bộ trạng thái file) và đồng thời lấy ra đối tượng semaphore tương ứng để gọi **Release()** giải phóng tài nguyên. Kết thúc.*

## **E. LIÊN LẠC ĐA TIỂU TRÌNH:**



**I. Chia Sẻ Tài Nguyên (Static):**

Các tiểu trình để có thể hoạt động tốt, thực thi các công việc với nhau cần có thông tin liên lạc với nhau, đó chính là sử dụng các không gian địa chỉ chung trên cùng 1 tiến trình.

Khi các Thread cùng hoạt động xử lý các tác vụ yêu cầu của các Client thì có dùng một số tài nguyên như biến Static tĩnh hoặc CPU để xử lý từng dòng code . Trên Server, em có dùng các biến `static` dùng chung cho các đối tượng `class ThreadClient`, các biến này dùng để lưu dữ liệu download offline và quá trình đồng bộ hóa download :

```
//du lieu download offline :
static int numThread = 10;
static DTOClient LoginClient = new DTOClient();
static DTOData DownData = new DTOData();
static Thread[] T_downloadWait = new Thread[10];
static Semaphore S_offlineDown = new Semaphore(1, 1);    //(x,y)

// Du lieu cho dong bo cho download :
static int i_Semaphore;
static String nameSemaphore;
static List<Semaphore> listSemaphore = new List<Semaphore>();
static List<String> listNameSemaphore = new List<String>();
static List<bool> listCheckFile = new List<bool>();
static Queue<String>[] listDataSemaphore = new Queue<String>[10];
```

- ➔ Là cơ chế trao đổi thông tin với nhau dễ dàng nhưng khi nhiều tiểu trình truy xuất tài nguyên dùng chung mà tiểu trình khác cùng truy xuất vào, do đó các biến static rất cần được bảo vệ khỏi sự truy cập không đồng bộ giữa các tiểu trình. Cơ chế đồng bộ đã được tìm hiểu và trình bày trong phần **D. Đồng Bộ Hóa (Semaphore).**

**II. Trao Đổi Thông Tin :**

Vấn đề trao đổi thông tin giữa Server và Client hoặc Client và Client với nhau là trao đổi thông tin giữa 2 máy với nhau, do đó cần một cơ chế trao đổi thông tin hợp lý giữa 2 chương trình trên 2 máy tính.

Trong chương trình chia sẻ file này sử dụng diện lập trình Socket, cụ thể trong bài làm này là trao đổi giữa 2 Thread với nhau dùng cơ chế liên lạc Socket, Socket thiết lập kênh truyền thông tin từ 2 yếu tố là : địa chỉ máy tính và Port của máy tính, Thông qua địa chỉ và port thì 2 Thread có thể trao đổi thông điệp với nhau.

Cài đặt mô hình Socket ta có thể dùng các hàm mà Socket có hỗ trợ như : `Socket.Connect()`, `Socket.Bind()`, `Socket.Send()`, `Socket.Receive()`, `Socket.Close()` ....

**Một vài quá trình kết nối sử dụng Socket :**

- Client kết nối đến Server để đăng ký 1 định danh, đăng kí xong thì kết nối này sẽ bị Server đóng .
- Client kết nối đến Server để đăng nhập, nếu thành công thì kết nối này được giữ xuyên suốt quá trình trao đổi giữa Server và Client đến khi Client ngắt kết nối này.
- Client download file từ Client share , quá trình thiết lập kết nối này chỉ diễn ra trong quá trình bắt đầu download đến khi download kết thúc thì kết nối này cũng kết thúc theo .
- Server Connect đến Client qua Port lắng nghe của Client để gửi thông tin download offline khi client share file online đăng nhập vào mạng .

**F. XÂY DỰNG TESTCASE:****I. Đồng Bộ Hóa:**

Quá trình đồng bộ đã được nêu rõ và khá chi tiết trong phần Mô hình đồng bộ hóa. Ở đây sẽ nói về các trường hợp xảy ra trước và sau khi đồng bộ.

	Trước khi đồng bộ	Sau khi đồng bộ
Register	<ul style="list-style-type: none"><li>- Cùng tranh đoạt miền CS ( CSDL) nên có thể treo máy, thông tin cập nhật sai.</li><li>- Khó debug lỗi sai trong xử lý đồng hành.</li></ul>	<ul style="list-style-type: none"><li>- Thông tin cập nhật chính xác.</li><li>- Đảm bảo duy nhất 1 tiến trình hoạt động trọn vẹn quá trình truy xuất tài nguyên chung trước khi có tài nguyên khác trước khi có tiến trình khác can thiệp.</li></ul>
Login	Treo máy, đôi lúc client đã đăng nhập thành công nhưng trạng thái không cập nhật trên Server.	Client nào đăng nhập, thông tin cập nhật tương ứng của Client đó luôn đảm bảo, không sai sót. Trạng thái Client khi login sẽ luôn là connect khi Login với Server (nếu hợp lệ).
Quit	Thoát đôi lúc báo lỗi, treo máy, hoặc thoát thành công nhưng trạng thái trên Server vẫn còn là connect. Khi muốn login lại không được.	Đảm bảo Client nào có nhu cầu thoát đều hoàn thành chính xác. Không xảy ra treo máy, trạng thái được cập nhật trên Server. Đảm bảo việc Login tiếp đó có thể thực hiện.



Share	Update thông tin dữ liệu file chia sẻ vào CSDL không thực hiện hoàn toàn chính xác. Có thể file của Client này nhưng lại ghi vào CSDL thành file của Client khác và ngược lại. Hoặc có đôi lúc máy bị treo vì không xử lý kịp thời việc truy xuất cùng lúc của nhiều Client.	Thông tin update lên CSDL được thực hiện một cách tuần tự, chính xác, rõ ràng. Tại một thời điểm đảm bảo duy nhất có một tiến trình được thực hiện. Xung đột dữ liệu không xảy ra. Đảm bảo hệ thống hoạt động ổn định, không gây lỗi.
Download	Nhiều Client cùng muốn down 1 file của 1 Client khác sẽ có thể down cùng lúc nhưng vi phạm nguyên tắc FIFO, tại một thời điểm chỉ có nhiều Client down cùng file đó. Hoặc đôi lúc máy bị treo, thông tin file down không chính xác, thiếu toàn vẹn dữ liệu, hỏng file.	Đảm bảo nguyên tắc FIFO: tại 1 thời điểm có duy nhất một tiến trình được thực hiện việc download file dùng chung, quá trình download file luôn luôn hoàn thành với độ chính xác cao. Không gây thất thoát dữ liệu. Hệ thống hoạt động tốt. Không gây die hệ thống đột ngột.

## **II. Đa Tiểu Trình:**

### **1. Đa Tiểu trình trên Server :**

- Khi server không có mô hình đa tiểu trình thì khi chấp nhận nhiều kết nối của các Client khi kết nối để đăng nhập và chia sẻ file, quá trình đó diễn ra một cách tuần tự, hết client này đến Client khác khiến sự chờ đợi diễn ra.
- Đa tiểu trình trên Server được thể hiện qua việc thiết lập được nhiều kết nối với các Client khi các Client connect đến Server, khi Client connect đến Server thì Thread S\_threadWork sẽ lần lượt được khởi tạo và chạy cùng với Socket mới được thiết lập đó. Khi mỗi Client kết nối đến, thì Server đều tạo ra một luồng xử lý hay Thread để trao đổi thông điệp với Client đó.

### **2. Đa Tiểu trình trên Client :**

- Trên Client khi không có mô hình đa tiểu trình thì chỉ xử lý được một cách tuần tự để trao đổi thông tin với Server, Client vẫn download và chia sẻ file được nhưng quá trình xảy ra một cách tuần tự, các công việc không có song song cùng làm,

- Khi xử lý quá trình đồng bộ trên Client thì các công việc sẽ diễn ra cùng nhau, song song do HĐH điều phối các luồng xử lý theo chiến lược điều phối. Quá trình diễn ra trong suốt với người dùng, các Thread hoạt động theo ngắt của HĐH.

### **III. Liên Lạc Đa Tiểu Trình :**


Mô hình hoạt động đa tiểu trình trên Client và Server liên lạc với nhau qua Socket, thể hiện qua việc test trên 2 máy kết nối với nhau bằng socket, quá trình trao đổi dữ liệu hoàn tất.

### **G. BẢNG PHÂN CÔNG:**

<b>Huỳnh Phi Hùng (0812195)</b>	<b>Nguyễn Hoàng Đăng Khoa (08122 31)</b>
<b>1. Thiết kế cài đặt mô hình Server – Client.</b>	<b>1. Thiết kế , cài đặt cơ sở dữ liệu lưu trữ.</b>
<b>2. Cài đặt mô hình đa tiểu trình và liên lạc đa tiểu trình.</b>	<b>2. Cài đặt quá trình đồng bộ hóa trên Server và Download File của các Client.</b>
<b>3. Xây dựng các Test Case đa tiểu trình.</b>	<b>3. Xây dựng các Test Case đồng bộ hóa.</b>
<b>4. Design giao diện.</b>	<b>4. Design giao diện.</b>

### **H. NHẬN XÉT:**

#### **1. Ưu điểm:**

- Hoàn thành tất cả các yêu cầu mà bài tập đã đưa ra.
- Có xây dựng thêm chức năng resume khi cần và pause khi muốn dừng quá trình download.
- Cho phép Client có thể chỉ ra Client mà mình muốn download và 1 file có thể được chia sẻ bởi nhiều Client khác.
- Tạo được tray icon khi thu nhỏ màn hình.
- Ứng dụng mô hình 3 lớp để quản lý, truy xuất CSDL một cách hiệu quả nhất.
- Việc thoát khỏi chương trình đã được xây dựng một cách tối ưu, có thể nhấp chọn nút  (Close ), hoặc chức năng Quit của chương trình.
- Nhiều Client hoạt động độc lập với nhau trên cùng một máy nhưng số port kết nối phải khác nhau.

#### **2. Khuyết điểm:**

- Tuy nhiên, trong quá trình xây dựng chương trình có những đoạn code chưa thực sự tối ưu hoàn toàn.
- Chưa thực sự hoạt động tốt trên WIN 7, đôi lúc xảy ra lỗi ngoài ý muốn nhưng chưa xác định được là gặp vấn đề gì.

+++ HẾT +++